
# Game Prototyping

## Building 100s of Games using HTML5 & Phaser.js Gaming Frameworks

( 6th Edition includes v2.x.x & v3.24+)

By Stephen Gose

# Phaser Game Prototyping

Building 100s of games using HTML5 & Phaser.js Gaming Frameworks (6th Edition includes v2.x.x & v3.24+)

Stephen Gose

This book is for sale at http://leanpub.com/LoRD

This version was published on 2021-05-28

ISBN 978-1-952635-04-5

# Tweet This Book!

Please help Stephen Gose by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm making HTML games using Phaser Game Prototyping workbook.

The suggested hashtag for this book is #PBMCube.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#PBMCube

*For my students*

*@ Culpeper Public Schools, Culpeper, VA;*

*@ ITT Technical Institute, Tempe, AZ;*

*@ Early Career Academy, Tempe, AZ; and*

*@ University of Advancing Technology (UAT), Tempe, AZ*

# Contents

# Part II: Mechanisms vs. Mechanics . . . . . 127

# Distribution Permission



## Supporting website

**https://makingbrowsergames.com/**

Thank you for your patronage! I truly appreciate it.

# Forwards

- ***by Terry Paton:*** — "Copying or imitating is an awesome way to learn how to do something, traditional artists have done it for centuries. This practice was generally considered a tribute, ***not forgery***[1], — **If you want to get better at something, then trying to do it like those who already have mastered it.** Look at the choices they have made and consider why they made those decisions, often important things are hidden in subtlety, and the only way we learn those subtleties is by creating the same thing. The balance here is `stealing` versus `inspiration`. **"Ripping off" ideas from someone else in a way that harms their hard work compared to producing something which is `inspired by` their work.** If you plan on publicly releasing something, I recommend you should inject some of your own vision into any game you make, take a concept but then extend or change it to create something of your own."

[1] http://en.wikipedia.org/wiki/Art_forgery

# Disclosures

- ***Stephen Gose LLC reserves the right, at any time and without notice, to change modify or correct the information contained in this publication.***
- I refer to "Phaser v3.16+" under a ***moniker of "Phaser III"*** to distinguish it as a clear demarcation from previous versions.

In this book, I am not paid to recommend any of the tools or services presented but I do use affiliate links. Here's how it works. When I find a tool, service, author's content, idea, or product I admire, I investigate if they have an affiliate program. If it exists, I get a special link and when you click it or confirm a purchase I receive a small percentage from that activity. In short, it's the same methods everyone finds on any typical website; only now, those links are available inside ebooks as ***substitute for "crowd-funding".***

I think everyone, with ***any business savvy***[2], should do this too; especially when you recommend your own books, services, and tools. Amazon and others offer affiliate links. Whenever you recommend anything **(hopefully this book? hint, hint!), use your affiliate links.**

By law, I must disclose that I am are using affiliate links. Amazon, in particular, requires the following.

> *"We are a participant in the Amazon Services LLC Associates Program, an affiliate advertising program designed to provide a means for us to earn fees by linking to Amazon.com and affiliated sites."*

[2] https://www.inc.com/neil-patel/11-mental-habits-that-will-improve-your-business-savvy.html

# Disclaimer

- All the information, contained within, is for the convenience of its readers. It is accurate, as can be reasonably verified, at the time of original publication. However, this content may not reflect migrating industry recommendations ***after the original publication date*** for ***ECMA-262***[3] (also known as ***(aka)*** "JavaScript", ES5, ES6, ES7, ES8, ES9 or ES10) or for any version of the ***Phaser.JS Gaming Frameworks.***
- All websites listed herein **are accurate at the time of publication but may change in the future or cease to exist.** It is best to research these "dead websites" links in ***"The WayBack Machine"***[4]. The website references does not imply my endorsement of a site's contents.
- There are no guarantees nor warranties stated nor implied by the distribution of this educational information. Using this information is at the reader's own risk, and no liability shall carry to the author. Any damage or loss is the sole responsibility of the reader.

***Warning:*** The Phaser newsletter dated 21 September 2018 includes projected development on Phaser III. In August 2017, many features in pre-v3.16.x were removed. There were many business decisions on why they were removed based on financial support and sponsorship deadlines imposed. Phaser v3.14.0 (released OCT 2018) saw the return of many deleted features. In other words, Phaser v3.14.0 returns to the original vision of January 2017 after several rewrites. Phaser v3.15+ was the next massive re-write (released OCT 2018); followed by v3.16.0 DEC 2018. Phaser v3.5 is nearing completion with more "breaking changes".

My best guess is that ***any books, tutorials, blogs, and "how-to" articles — written before to Phaser v3.16+ (NOV 2018) — are not fully functional with Phaser III and should be re-written to the Phaser v3.24+ minimum standard baseline. Hence the reason this book is dedicated and updated to the official Phaser III (release v3.24+) and has removed any references to previously released versions. (See newsletter #139 dated 20190211) "Breaking Changes"***[5]

[3] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[4] https://web.archive.org/
[5] https://madmimi.com/p/f0b3bd

# About this Workbook:

This 6th edition offers additional production methods, beyond the former editions, I call the ***Game Design System™*** from which we'll create ***Game Recipes™.*** Expert game developers understand the "Don't Repeat Yourself" (D.R.Y.) concept, yet few have taken a step back to the ***"10,000-foot view"***[6] (i.e., "Executive level thinking") on their game production pipelines. We'll do that aerial view in this book as we build gaming prototypes, linked to ***"Headless" Game Design***[TM7] mechanics, and artwork. Then, **"nose dive head-first"** directly into game prototyping. I believe you will be surprised how quickly and easily we can build 100s of games using this ***Game Design System™*** with its ***Game Recipes™ tools.***

This workbook is intended to be a hands-on guide for "HTML5 Game Development" with an emphasis on the ***Phaser JavaScript Gaming Frameworks.*** Yet, our game production and project management could apply ***to any browser-based gaming framework!*** It's not exclusive to the Phaser III JS Gaming Framework.

I'm assuming you already have a "working knowledge" of the ***"Front-end" technologies***[8] — specifically ***HTML5***[9], ***CSS3***[10], and ***JavaScript***[11] — in your arsenal. I know that many *senior "full-stack" and "front-end" developers* do; but, I have received dozen of emails about this book's former editions as being **"... too difficult for those just starting their own game studios."** Therefore, ***if learning any of these web development technologies***[12] ***is what you're initially seeking,*** then I recommend a quick visit to ***W3Schools***[13] as your **first FREE starting point.** By following their instructions, you will learn a complete foundation in HTML5, CSS, and JavaScript **in a matter of hours! ... then, return to this workbook and learn how to combine those technologies into your own game creations.**

[6] https://en.wikipedia.org/wiki/Business_acumen#Executive_level_thinking
[7] https://leanpub.com/hgd
[8] https://roadmap.sh/frontend
[9] http://amzn.to/2nAYjxr
[10] http://amzn.to/2mG01Zv
[11] https://amzn.to/2Iw9RZj
[12] https://github.com/kamranahmedse/developer-roadmap
[13] http://www.w3schools.com/js/default.asp

## Viewing the Source Code

This e-Book includes source code which is optimally viewed **in single-column, landscape mode** with the font size adjusted to a comfortable setting.

## Links and References

The Internet is a living, dynamic resource of information ***that doubles every 35 days! There are several reasons this book points to external content. Because —***

1. It provides you the "research path" I took to develop and present my ideas. It takes all the guess-working and personal research out of it. This saves you 100s of hours of your personal free-time searching for supporting facts and opinions.
2. It ***avoids copyright infringements*** and provides the **required acknowledgments** to "Open-source authors" for use of their contributions and resources under the various **licenses and EULA**[14]**.**
3. It provides external authors an opportunity ***to recant or update their content. (Kindly review Phaser v3 DevLogs as an example.)*** Technology is a fast-moving target, and what was once "cutting edge" becomes obsolete. For example, the use of **"`window.onload`"** was recanted by its originator back in 2014 as an unsafe method for launching browser applications. (see ***Phaser Game Design Workbook, pages 15-22***[15] for complete details. )
4. It **reduces your initial purchase price** from the reams of "padded source code content" **— don't make me embarrass those authors who do this —** while keeping your investment in this book's information **"fresh".** This book would be triple the size and "4x" the price if I had embedded all of the source code tutorial's as many others do.

All the source code is written in **"pure"** JavaScript (JS) and the ***Phaser.js Gaming Frameworks;*** it doesn't use any additional "abstraction layers" such as "TypeScript", "CoffeeScript", or "JQuery" for obvious reasons.

You'll find your ***Bonus content, source code, and references*** in ...

- the footnotes links,

[14] https://choosealicense.com/licenses/
[15] http://leanpub.com/phaserjsgamedesignworkbook/

- external reference links, and in **the file** which are available directly from the supporting website — without registration nor private logon thus keeping your personal information safe!
- or from the latest and most current updates inside your **LeanPub.com personal library (assuming that you're a LeanPub patron[16]).**

## Who should use this workbook?

This workbook targets both the ***learning novices***[17] — those who enjoy "learning by doing" using ***"deliberate practice"***[18] — and the ***experienced expert programmers***[19] in web-application development; and, of course, those who want a finished game from their own designs and efforts. If you are interested in making browser games, ***especially for the Mobile/WebXR markets***[20]***,*** then this book is a perfect choice along with its companion volumes: the ***Headless HTML5 Game Design***™[21], ***Phaser Game Starter Kit Collection***[22], ***Phaser Game Design Workbook***[23], and ***Phaser Game Prototyping***[24]. With this in mind, you will do a lot of writing, thinking, and coding in JavaScript throughout this workbook. You may prefer using paper (external physical- or soft-"paper") to organize your development ideas and processes.

I've ***"gone to great lengths"***[25] to make this book "skim-friendly" ***— even for my International customers*** by emphasizing important concepts in bold font type. I have provided links to ***"English (American) Jargon phrases"***[26] that will help translate this content directly into your native language. I use "Notes", "Tips", "Warning" and "Best Practices" icons to encapsulate those ancillary topics for your further education from **other experts** in the gaming industry.

I further assume that many readers will want to use this book to quickly build their ***bespoke***[27] game products. So, I've included references to other similar game examples, gaming engines, frameworks, ***GUI game kits***[28], indie developers, authors, their "open-source" contributions, articles, books, ***artwork***[29], application tools, and their wisdom.

[16] https://leanpub.com/u/pbmcube
[17] https://www.nap.edu/read/9853/chapter/5
[18] https://www.nateliason.com/notes/peak-anders-ericsson-robert-pool
[19] https://www.nateliason.com/blog/become-expert-dreyfus
[20] https://immersiveweb.dev/
[21] http://leanpub.com/hgd
[22] https://leanpub.com/p3gskc
[23] https://leanpub.com/phaser3gamedesignworkbook
[24] https://leanpub.com/phaser3gameprototyping
[25] https://dictionary.cambridge.org/us/dictionary/english/go-to-great-lengths
[26] https://www.smartling.com/blog/40-american-slang-words-and-phrases-you-need-to-know/
[27] https://www.urbandictionary.com/define.php?term=bespoke
[28] https://craftpix.net/categorys/2d-game-kits/?affiliate=112238
[29] https://www.gamedevmarket.net/?ally=GVgAVsoJ

In summary, if you are a ***hobbyist, independent game developer, student, teacher, or start-up game studio*** you will find a wealth of information on both product and project management, game design, game mechanisms, game mechanics, and "insider's tips" about the ***Phaser JavaScript Gaming Frameworks.***

**If you'd like a link to your finished games in future updates, please use my contact information on LeanPub.com[30] or Amazon.com Author's page[31].**

## Your newly obtained skills...

By the end of this workbook, you'll have integrated into your own ***bespoke***[32] game design:

- ***Step-by-step methods migrating older Phaser v2.x.x into III.***
- ***Built "future-proof" and flexible game architectures.***
- Used the ***"Headless" Game Design System™*** which creates ***Game Recipes™***[33] from automated tools.
- Demystified **Web Sockets for optimized game deployments.**
- Adopted processes for business product/project management and ***"extreme programming"***[34].
- Organized a standardized file structure for general game developments;
- Used a blank game template to scaffold further game projects;
- ***Converted and adopted new changes in the Phaser III API.***
- ***Managed groups and layers of game objects with Phaser Frameworks;***
- Imported resources and game assets;
- Displayed, animated, and moved game avatars on various screen renderings;
- Incorporated sound effects (sfx) and theme music across various game phases;
- Deployed "heads-up display" (HUD) on game scenes both inside and outside the canvas;
- Used customized web fonts;
- Incorporated multiple game-inputs (touch, multi-touch, accelerometer, mouse, and keyboard);
- Implemented several physics systems in both Phaser Gaming Frameworks;
- Created and managed various game deployments (CMS, PWA, & SWPA);

[30] https://leanpub.com/u/pbmcube

[31] https://www.amazon.com/kindle-dbs/entity/author/B01N780CUF?_encoding=UTF8&node=283155&offset=0&pageSize=12&sort=author-pages-popularity-rank&page=1&langFilter=default#formatSelectorHeader

[32] https://www.urbandictionary.com/define.php?term=bespoke

[33] https://www.nateliason.com/blog/become-expert-dreyfus

[34] https://en.wikipedia.org/wiki/Extreme_programming

- Managed permanent cache assets across game phases;
- Optimized games for various mobile devices;
- Integrated several 3rd-party scripts and services.

# Game Design System™

***"Game Recipe™" Courses*** (purchased separately on specific gaming mechanics) using the ***Game Design System™ management method and concepts.*** These courses enhance your skills and are available from my educational websites (***hosted by LeanPub.com***[35]) or ***Training By Blackboard, Books, and Browsers***[36].

You can earn your ***Game Development Certifications*** from ***my online courses***[37], from ***Udemy (102-pages of online courses!)***[38] ***or Zenva (my personal favorite!)***[39]; to enhance your resume.

## Game Studio - Book Series

**Learn to build a Gaming Studio for passive (or secondary) income!**

- ***Game Studio Starter Kit Bundle***[40] — Start your own Game Studio for passive or secondary income! This bundle shows you how to develop product and project management in the gaming industry from my 23 years of experience. You'll learn how to quickly build game prototypes in any genre, launch, and then distribute your games. You'll also have 16+ popular game genres to choose for your product line with 19 subgenres to to expand upon. Learn how to capture various game industry market shares.
- ***Headless HTML5 Game Design (Vol. I)***[41] — Creating Cloud-based **"Content-as-a-Service" (CaaS)** games ***for Any Gaming Device.***
- ***Making Massive Multi-Player Online games (Vol. II)***[42] — Creating Multi-Player Online games ***using the Full-stack, White-labeled, and "Content-as-a-Service" (CaaS) Architectures.***

[35] https://leanpub.com/u/pbmcube
[36] http://tbcube.com./
[37] https://leanpub.com/u/pbmcube
[38] https://click.linksynergy.com/deeplink?id=pmlyJRiRsYE&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fcourses%2Fdevelopment%2Fgame-development%2F%3Fsearch-query%game%2Bdevelopment
[39] https://academy.zenva.com/?a=47&campaign=RPGCaaS
[40] https://leanpub.com/b/gssk
[41] https://leanpub.com/hgd
[42] https://leanpub.com/mmmog

- ***Making Multi-Player Online games***[43] — A Game Development Workbook ***for any*** Phaser JavaScript Gaming Framework. This book is a thorough review of MMoG mechanics for both client- and server-side APIs using Block-chain, WebRTC, RPC, MoM, SSE, Cloud Services, and Web Sockets (Berkeley). ***I do not recommended for entry-level developers;*** mastery of several ***IT technologies***[44] is required.
- ***Phaser JS Game Design Workbook***[45] — 6th Edition for v2.x.x ***and*** v3.16+ — guidance on project and product management in the gaming industry.
- ***Phaser Game Prototyping***[46] — 6th Edition for v2.x.x ***and*** v3.16+
- ***Phaser Game Starter Kit Collection***[47] — 6th Edition for v2.x.x only.
- ***Phaser III Game Prototyping***[48] — 6th Edition for v3.16+ only.
- ***Phaser III Game Starter Kit Collection***[49] — 6th Edition for v3.16+ only.

## Game Studio - Online Courses

- ***Phaser Game Design Workshop Course***[50] — guidance on programming your first game in v2.x.x.
- ***Phaser Starter Kit Game Collection***[51] for either Phaser v2.x.x ***or*** Phaser III.
- ***Phaser III Game Design Workshop Course***[52] — guidance on programming your first game in v3.16+.
- ***Game Studio Starter Kit Collection (basic)***[53] — 3 courses are included in this Business starter kit for Game Studios ... “Making Dating & Quiz Browser games”, “Making Online Dress-UP Fashion games”, and “Making Puzzle Browser games” with Phaser v2.x.x.
- ***Ultimate Game Studio Starter Kit Collection***[54] — 6 course-set are included in this Business starter kit for Game Studios. Build your own Game Studio business for as little as $17.99.

[43] https://leanpub.com/rrgamingsystem
[44] https://github.com/kamranahmedse/developer-roadmap
[45] https://leanpub.com/phaserjsgamedesignworkbook
[46] https://leanpub.com/LoRD
[47] https://leanpub.com/pgskc
[48] https://leanpub.com/phaser3gameprototyping
[49] https://leanpub.com/p3gskc
[50] https://leanpub.com/c/phasergamedesignworkshop
[51] https://makingbrowsergames.com/p3gskc/
[52] https://leanpub.com/c/p3gdc
[53] https://leanpub.com/set/leanpub/gsskit
[54] https://leanpub.com/set/leanpub/ugsskitc

# "Making Browser Games" - Books Series

Individual Chapters **— *sold separately from the "Phaser Starter Kit Game Collections books"***[55] **—** contain **both** the Phaser v2.x.x **and** Phaser "III" (3.16+) examples, source code, and game license. Find and select your favorite game genre.

Chapter 1 — ***Action & Arcade***[56]
Chapter 2 — ***Adventure Mazes & Story Plots***[57]
Chapter 3 — ***Collapsing Blocks***[58]
Chapter 4 — ***Connect 4 & Go***[59]
Chapter 5 — ***Dating Simulations & Quizzes***[60]
Chapter 6 — ***Defensive Towers***[61] — the typical tower-defense constructions **with two innovative variations.**
Chapter 7 — ***Dress-Up & Fashion***[62]
Chapter 8 — ***Hidden Objects***[63]
Chapter 9 — ***"Jump to Capture"***[64]
Chapter 10 — **MahJong — available only** in the ***"Phaser Starter Kit Game Collections"***[65] volumes or the "Memory Match" ***mega-chapter.***
Chapter 11 — ***Match-3 & Trace 3+***[66]
Chapter 12 — ***Memory Match***[67] for Pairs (either "Open" or "Hidden") & Sequence matching — a **"mega-chapter" with 5 games and licenses.**
Chapter 13 — ***Music & Rhythm***[68]
Chapter 14 — ***Puzzle (both Jigsaw & Sliders)***[69]
Chapter 15 — **Role-Playing Character Development — available only** in the ***"Phaser Starter Kit Game Collection"***[70] volumes. ***Role Playing Content-as-a-Service (CaaS)***[71] — a **"mega-chapter"** developing content for B2B, Affiliate Syndicates, and clients is **NOT available** in the ***"Phaser Starter Kit Game Collections"***[72] volumes.
Chapter 16 — **Simulations — available only** in the ***"Phaser Starter Kit Game***

[55] https://makingbrowsergames.com/p3gskc/
[56] https://leanpub.com/mbg-action-arcade
[57] https://leanpub.com/mbg-adventure
[58] https://leanpub.com/mbg-collapse
[59] https://leanpub.com/mbg-connect4
[60] https://leanpub.com/mbg-dating
[61] https://leanpub.com/mbg-towers
[62] https://leanpub.com/mbg-dressup
[63] https://leanpub.com/mbg-hidden
[64] https://leanpub.com/makingjump2capturebrowsergames
[65] https://makingbrowsergames.com/p3gskc/
[66] https://leanpub.com/mbg-match3
[67] https://leanpub.com/mbg-memory
[68] https://leanpub.com/mbg-music
[69] https://leanpub.com/mbg-puzzle
[70] https://makingbrowsergames.com/p3gskc/
[71] https://leanpub.com/mbg-rpg
[72] https://makingbrowsergames.com/p3gskc/

***Collections"***[73] volumes.
Chapter 17 — ***Strategy & Tactics***[74]

## "Making Browser Games" Series - online Courses

- ***Making Browser games — Tower Defense***[75] with Phaser v2x.x and v3.16+.
- ***Making Dating & Quiz Browser games***[76] with Phaser v2x.x.
- ***Making Online Dress-UP Fashion games***[77] with Phaser v2x.x.
- ***Making Peg Solitaire Browser games***[78] with Phaser v2x.x.
- ***Making Phaser III Peg Solitaire Browser games***[79] with Phaser v3.16+.
- ***Making Puzzle Browser games***[80] with Phaser v2x.x.
- ***Making RPG Browser games***[81] with Phaser v2x.x.

## Programming Courses

See the growing catalog of courses for ***college credit, home schooling or personal skills development*** at ***Training by Blackboard, Books & Browsers***[82]

- ***Using JavaScript OLOO in game development***[83] (learn JavaScript development)

## "Walk-Thru Tutorial" Series - Online Courses

These courses are "step-by-step" guides to create specifically designed games with some explanation as to why we do this (which is typically found in most online tutorials).

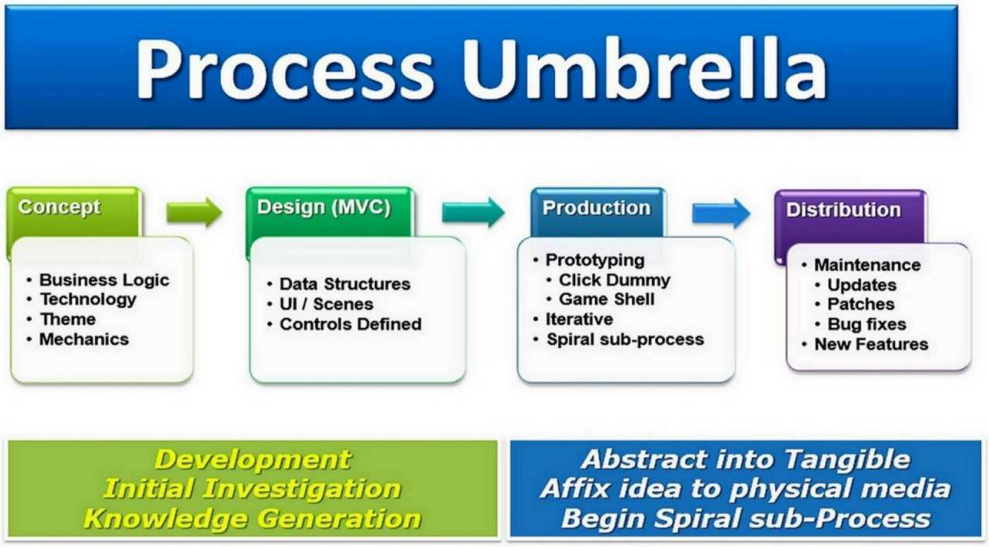
- ***"Walk-Thru Tutorial Series" - Blood Pit™ (IGM)***[84]

[73] https://makingbrowsergames.com/p3gskc/
[74] https://leanpub.com/mbg-strategy
[75] https://leanpub.com/c/mbg-p2p3-towerdefenses
[76] https://leanpub.com/c/mbg-dating
[77] https://leanpub.com/c/mbg-dressup-p2
[78] https://leanpub.com/c/mbg-peg-p2
[79] https://leanpub.com/c/mbg-peg-p3
[80] https://leanpub.com/c/mbg-puzzle-p2
[81] https://leanpub.com/c/mbg-rpg-p2
[82] https://www.tbcube.com/
[83] https://leanpub.com/c/jsoloo
[84] https://leanpub.com/c/bloodpit-wtts

- ***"Walk-Thru Tutorial Series" - Blood Pit II™***[85]
- ***"Walk-Thru Tutorial Series" - Mozart's Music Match™***[86]
- ***"Walk-Thru Tutorial Series" - Rogue Prince Quests™ (IGM)***[87]

[85] https://leanpub.com/c/wtts-bloodpit2-mmog
[86] https://leanpub.com/c/wtts-mmm
[87] https://leanpub.com/c/arrp

# Part I: Product Management




**Part I Introduction to Game Design System™**

I'm skipping to my 3rd "Product Management Phase" and we'll create a single-player prototype in both Phaser v2.x.x & v3.24+. More detailed information about the "Product/Project Management" phases **— *Concept, Design and Distribution* —** are available now only in these "Game Studio" series companions:

- ***Phaser.js Game Design Workbook (6th edition)***[88].
- ***Headless Game Design***[89] a "Product Management" workbook for "Content-as-a-Service".

[88] https://leanpub.com/phaserjsgamedesignworkbook
[89] https://leanpub.com/hgd

***Part I*** is a brief excerpt from ***Phaser.js Game Design Workbook (6th edition)***[90] explaining the ***Game Design System™*** and its ***Game Recipe™*** construction. Our goal is to build several fully-functional "Phaser Game Prototypes". There are several step-by-step instructions and file downloads. We'll catalog and create various ***"Game Framework Mechanism" (GFM)***[91] components — those ***"visual elements"*** that are separate from our core ***"Game Mechanics" (GM) genres*** and Artwork theme components. From this simple foundation, we'll learn to combine these "building blocks" into various game products as easily as a child would use "Lego"™ blocks to construct a toy castle. Furthermore, you can review this construction process throughout ***"Phaser Game Starter Kit Collections"***[92] — a workbook of 16+ popular game genre mechanics and 19 sub-genres. By the end of "Part I", we'll have created everything a game prototype uses:

- Creating visual avatars and their associated metadata structures;
- Collecting a player's input;
- Detecting collisions and interactions among various elements;
- Migrating to dynamic game phases;
- Monitoring the gaming loop;
- Creating and Updating heads-up displays (aka "HUD").
- Reviewing the massive overhaul of the "Tween" System in the v3.19+;
- Events and Call-backs &mdash where and when to use;
- Sound effects (sfx); and
- Sprite Animation(s).

[90] https://leanpub.com/phaserjsgamedesignworkbook

[91] http://www.michael-richardson.com/processes/rup_for_sqa/core.base_rup/guidances/concepts/component_A2E2B3B1.html

[92] https://leanpub.com/p3gskc

# 1. Game Studio & Project Preparations

This chapter is focused on organizing the project's file structure and setting up your workstation environment. It will allow us to:

- have the software tools available for game production
- maintain an organized file structure;
- facilitate project creation, and
- test various aspects of our game

The first impressions you'll develop while reading this chapter is: ***THERE'S NO FRONT-END BUILD TOOLS*** "Why is that?", you're thinking? My goal is to provide a direct "no non-sense" approach in game construction. In many competing tutorials and books, you'll read chapter***S(!)*** on working with ***"`node.js`", "`grunt`", "`bower`", "`yeoman`", "`webpack`", "`brunch`", "`gulp`", etc. (ad nauseam)***[1]. The shame, of all this, is that folks are beginning to write such articles as ***"I finally made sense of front end build tools. You can, too."***[2] Another popular game developer, Andrezej Mazur of Enclave Game, stated my same sentiments in this way ...

> "Everything changed so much over the years. I feel like grandpa right now — back in my days, we typed pure CSS and used jQuery in a Notepad. Right now, to start a new project, you need ***a few days*** to chose the right tools, configure the build process, have proper configs and settings, preprocessors in place, etc. I would really need ***at least a few solid months*** to go through the most popular tool-chain. ... That's why I think I'll end up using the "quick and dirty" approach — I'll do a research about using a common template ***or a starter (kit)***[3] ***and will expand from it."*** (quoted from https://dev.end3r.com/2018/11/gamedev-versus-front-end/)

### *ACTUALLY START THE DAMN GAME ...*

"Writing your idea down is not starting the damn game. Writing a design document is not starting the damn game. Assembling a team is not starting the damn game. ***Even doing graphics or music is not starting the damn game.*** It's easy to confuse

[1] https://en.wikipedia.org/wiki/Ad_nauseam
[2] https://medium.freecodecamp.org/making-sense-of-front-end-build-tools-3a1b3a87043b
[3] https://makingbrowsergames.com/book/standalone.zip

> ***"preparing to start the damn game" (ed.: and all those "FRONT-END BUILD TOOLS"!) with "starting the damn game".*** Just remember: a damn game ***can be played, and if you have not created something that can be played, it's not a damn game!***
>
> So dammit, even creating a ***game engine*** is not necessarily starting the damn game. Which brings me to the next tip..." Read more such advice ***here ...***[a]
>
> [a] http://makegames.tumblr.com/post/1136623767/finishing-a-game

***Well, ... Ok then,*** let's get our game "prototype" started and finished; then, we'll address those ***"Front-end Building Tools"***[4] in later chapters. BUT first, we need a workstation to build stuff ...

***Hint:*** Google and Mozilla both provide excellent resources for ***Game***[5] and ***Web Developers.***[6]

## 1.1 Workstation Setup

Let's take an inventory of what you currently have on-hand. Do you have:

- A browser that is ***HTML5 compliant***[7]; now-a-days, most browsers are compliant. But you can double-check using this site: ***https://caniuse.com/#search=ES6*** (***NOTE:*** bookmark this site for use latter.)
- A separate directory (i.e., Microsoft "folder") to save and review each game projects' development files. ***Check you disk space***[8] now. This becomes your ***"software workspace"***[9]. (See Project Setup below)
- A "text editor" or "Integrated Develop Environment" (IDE) of your choice. (See Development Tools below)
- An operational web server? (Coming next ...)

[4] https://developers.google.com/web/tools/setup/setup-buildtools
[5] https://developer.mozilla.org/en-US/docs/Games
[6] https://developers.google.com/web/tools/
[7] https://caniuse.com/#search=html5
[8] https://www.wikihow.com/Check-Your-Hard-Disk-Space
[9] https://en.wikipedia.org/wiki/Workspace

## Batteries not included ... Web Server Required

### "Why do I need a web server? Can't I just open the `html` files with my browser?"

***Answer:*** All JavaScript games, that load assets and files, require launching itself from a web server — either locally inside your workstation or ***remotely from the Internet.***[10] It's all about browser security and the ***same-origin policies***[11] — prohibiting files loading from different ***"domains"***[12] and the protocols used to access your locally stored files. When you request anything from the Internet you're typically using the "hypertext transfer protocol" ***("`http://`" or "`https://`").*** From a web server, security policies ensure you only access files that you are authorized to use. When you open any HTML file from your local operating system, your browser uses the ***"`file://`"*** protocol. (***technically a different protocol***[13] than ***"`http://`"),*** massive restrictions are triggered inside your browser, for the following obvious reasons. Under the ***"`file://`"*** protocol, no concept of domains nor "server-level security policies" exists, just your computer's "raw file system" and its operating system — identified as the ***localhost***[14] — using a ***local IP address(es) (0.0.0.0 or IPv6 ::1 or IPv4 127.0.0.0/8 to 127.255.255.255/8)***[15] per ***RFC 990, November 1986.***[16] This means that your HTML pages ***are not running on any domain nor public Internet IP address at all,*** and thus JavaScript is ***unable to load any game assets dynamically.*** Do you really want JavaScript to have that much control — to load anything from anywhere — off your computer? Well, I'm guessing your answer should be "... not ever!". If JavaScript had unrestricted access using the ***"`file://`"*** protocol, nothing could stop it from tapping into your information and sending it anywhere to anyone.

***Exercise:*** Read more about browser security from the ***Chromium Blog***[17]

***Hint:*** "127.0.0.1" or "localhost" are IP addresses that default to your local workstation. The packet path ***never reaches the Network Interface Card (NIC). This is an important concept when creating web sockets and Multi-Player games.***

[10] http://gose-internet-services.net/data-centers/uk-data-center/
[11] https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
[12] https://en.wikipedia.org/wiki/Domain_Master_Browser
[13] https://www.w3.org/Addressing/
[14] https://en.wikipedia.org/wiki/Localhost
[15] https://www.lifewire.com/network-computer-special-ip-address-818385
[16] http://tools.ietf.org/html/rfc990#page-6
[17] https://blog.chromium.org/2008/12/security-in-depth-local-web-pages.html

## Deeper Dive: Testing *"MMoGs"* Locally??!

There's a trend to give your ***"`localhost`"*** a "top-level domain" (TLD) name of ***". `dev`". Stop! Don't! WHY?!?*** Because in 2017, Google has the TLD registration of several thousand of these commonly used domains that developers once used. ( See ***this article***[18]. )

***Exercise:*** Learn which TLD names are still available ***here***[19].

### Locally Testing *MMoG*s

***Wikipedia states***[a], "The processing of any packets sent to a loop-back address is implemented ***in the link layer of the TCP/IP stack. Such packets are never delivered to any network interface controller (NIC) or device driver, which permits testing of software in the absence of any hardware network interfaces.***

Like any other packets traversing the TCP/IP stack, looped-back packets convey the IP address and port number they were addressed to. Thus, the services that ultimately receive them can respond according to the specified loop-back destination. For example, an HTTP service could route packets addressed to 127.0.0.99:80 and 127.0.0.100:80 to different Web servers, ***or to a single server that would return different web pages. (ed.: one browser window to another browser window which is NOT true testing of any MMoG game.)*** To simplify such testing, the host's file can be modified to provide appropriate aliases for each such address."

[a] https://en.wikipedia.org/wiki/Localhost

"So then! What's a girl to do?!" The solution really is simple. Run your game development files **from a local web server or remotely from the Internet**[20]**.** Depending on your workstation operating system (and what you have installed already), there are several ways to launch web pages from a ***"web service"***[21].

- Some text editors and Integrated Development Environments (IDE) **already include a local web server.** ***Brackets***[22] is one such example.

[18] http://fossilgeek.jasonbaier.com/posts/google-chrome-63-forcing-dev-domains-to-https
[19] https://iyware.com/dont-use-dev-for-development/
[20] http://gose-internet-services.net/data-centers/uk-data-center/
[21] http://www.webopedia.com/TERM/W/Web_Services.html
[22] http://brackets.io/

- Another simple solution I discovered is the ***Google's Chrome Web Server.***[23] Once you install this application, you can launch any web page(s) directly from Chrome. Simply point it at your ***URI path and project folder.***[24]
- The official Phaser instructions (and sanctioned method) are ...

> ***Quote:*** "We would recommend either ***WAMP Server***[25] or ***XAMPP***[26] and both have easy set-up guides available. WAMP specifically installs an icon into your system-tray from which you can stop and restart the services, as well as modify Apache settings such as creating a new folder alias for a project." ***Read more details here***[27]. (***overlooked was MAMP or MAMP Pro***[28] ***available for those with MAC OS X.***)

***Exercise:*** Use this ***Google Chrome Web Server Install Instructions (movie: 1:51 minutes)***[29] to setup a webserver.

## Development Tools

Although Phaser web site has a well-documented section on ***"how to get started"***[30]. I recommend you test and develop your game using an Integrated Development Environment (IDE) editor of your choice. I use several: ***Mighty Editor (now open source)***[31], ***Phaser Editor 2D***[32], the ***(FREE) NotePad++***[33]; although, ***Brackets***[34] editor runs a close second in my daily web-site construction. *Do not use any word processing applications such as Microsoft Word*; this is not a "hate statement" against Microsoft. Word processing applications add invisible formatting to your source code that will lead to problems. If you do not have a favorite text editor, I have some recommendations based on your status:

[23] https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en
[24] http://uri.thephpleague.com/4.0/components/overview/
[25] http://www.wampserver.com/en/
[26] http://www.apachefriends.org/en/xampp.html
[27] https://phaser.io/tutorials/getting-started-phaser3/part2
[28] https://www.mamp.info/en/
[29] https://www.youtube.com/watch?v=AK6swHiPtew
[30] https://phaser.io/tutorials/getting-started-phaser3
[31] https://github.com/TheMightyFingers/mightyeditor
[32] https://phasereditor2d.com/
[33] https://notepad-plus-plus.org/
[34] http://brackets.io/

- ***StarUML v2***[35] and its ***v2 JavaScript extension***[36] is a "***Forward Engineering***[37]" tool that directly converts "UML Model diagrams" directly into JavaScript code. You can also create supporting documentation website by thoroughly defining your object models. If you can create UML models you can create gaming code in JavaScript.
- **FREE Access** to the online ***Game Designer Tool Kit***[38] — from which you can:
  - Randomly generate game ideas.
  - Collect game prototypes.
  - Automatically generate game design documentation and source code.
- ***Phaser Game mechanics Explorer***[39] is a limited set of 9 free examples.
- ***Remote Web Server***[40] pre-installed with ***Node.js, PHP, and Python 3.5.***
- Are you a student or instructor? Then obtain a FREE copy of ***PHPStorm***[41] or ***WebStorm***[42] — a savings of $199 per year; other enticements and discounts available for indie start-ups. ***OVER one third of all professional "full-stack" developers use this IDE. See the proof here***[43]
- Are you an indie game development start-up or hobbyist with little cash assets? Then obtain a FREE copy of ***NotePad++***[44]; it's the editor I use. Or, use ***Microsoft Visual Source / Code***[45] if you are inclined to develop in ***Typescript***[46] which transpiles to ***ECMAScript 3 by default!!!***[47] ***Only One fourth of all professional web developers use Typescript! See the proof here***[48]***.*** If you're having problems with Microsoft Visual Studio / Code, you're not alone. Read what others have done to ***solve MS Visual Studio / Code problems here***[49].
- Do you have money to spend? Then pick any of the thousands of software source code editors available online, and then send me the remaining money for my own development (wink, wink, my poor attempt at humor?!) On a more serious note, save your cash for ***game artwork***[50], visual assets, and ***"Graphical User Interfaces" (GUI)***[51] gaming kits

[35] http://staruml.io/
[36] http://staruml.io/extensions
[37] https://www.techopedia.com/definition/19445/forward-engineering
[38] https://makingbrowsergames.com/gameDesigner/
[39] https://gamemechanicexplorer.com/
[40] http://gose-internet-services.net/wordpress-web-hosting/
[41] https://www.jetbrains.com/phpstorm/buy/#edition=discounts
[42] https://www.jetbrains.com/webstorm/buy/#edition=discounts
[43] https://www.jetbrains.com/research/devecosystem-2018/javascript/
[44] https://notepad-plus-plus.org/
[45] https://code.visualstudio.com/
[46] https://en.wikipedia.org/wiki/TypeScript
[47] https://www.typescriptlang.org/docs/handbook/compiler-options.html
[48] https://www.jetbrains.com/research/devecosystem-2018/javascript/
[49] https://www.html5gamedevs.com/topic/30978-how-to-use-the-phaser-in-visual-studio-2017/
[50] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[51] https://craftpix.net/categorys/2d-game-kits/?affiliate=112238

## 1.2 Project Setup

Create the following directories/folders for this first project on your computer. You can quickly set-up the project by copying the "Project GamePrototype StarterKit" from inside the Bonus Content directory — the Bonus Content was an additional download available from your LeanPub.com personal library. For **other patrons**, you can download this project template from the book's website — ***Phaser template kit***[52].

***https://makingbrowsergames.com/book/_basic-phaser-template.zip***

### Deeper Dive: Project Data Structure

The arrangement of directories (***i.e.***[53] folders) and files is an important consideration. If you use some of those ***"Front-end Build tools",***[54] your project file structure is dictated; this provides less security since your game follows a known directory structure that everyone knows and uses.

I have read dozens of game development authors who literally dictate a rigid organization up to 10+ layers deep — they should reads Microsoft's warning about doing that in Windows O/S. "Why should I follow this?", has always been my question; **How does their organizational structures help or hinder the final game product?** I recommend that you avoid this non-sense of 10-levels deep directories. ***Create an organization of directories***[55] that make sense to you and labeled as you please. A side benefit of doing so, is a security feature, because it becomes harder to guess ***your directories and naming schema.***[56]

On the other hand, if you are working on a game development team, then directory structure takes on a new meaning as: ***"Name Space"***[57] for local and public variables. Consistency and standardization rules as King ***in software collaborative efforts.***[58] Many game developer turn to ***tools such as Vagrant***[59] for help.

When you are creating a game project (aka Blueprints, templates, or Game Starter Kits, make-up your own "marketing nickname") its "File Structure" should be consistent

[52] https://makingbrowsergames.com/book/_basic-phaser-template.zip
[53] https://www.grammarly.com/blog/know-your-latin-i-e-vs-e-g/
[54] https://frontendmasters.com/books/front-end-handbook/2018/tools.html
[55] https://addyosmani.com/blog/essential-js-namespacing/
[56] https://namingschemes.com/
[57] https://en.wikipedia.org/wiki/Namespace
[58] https://en.wikipedia.org/wiki/Collaborative_software
[59] https://www.vagrantup.com/

across all your projects. ***"Why's that?" Well, I'll tell you***[60]; because when you create another new project and "re-factor" everything (i.e., rename stuff) to coincide with the new project, it is easier to "find & replace" consistent formatting and file names. I'll show you how to do this in Part II and walk through that process in Part III.

Coming next is the structure I use, which makes the most sense for me, when creating my games. ***Having consistency across all your projects, makes it easier for other staff members to know where everything is located as they work on — and switching between — projects .***

## Deeper Dive: And its name shall be called …

### *What is a "Namespace"?*

Namespace is a container for a collection of identifiers, functions, methods, and variables deployed ***away from the global setting (browser window object).*** They are used to organize blocks of functionality into logical groups having a unique identity. Unfortunately, JavaScript doesn't provide namespacing by default. So anything (function, method, object, variable) created in JavaScript appears in the window object (aka global object from which all primitive members reside), and additional software structures continue polluting that global namespace by adding more to it. To solve this problem you can create a single object in the global scope for our game, and make all the game's functions, variables, and properties inside that secluded object. Read more ***here***[61] and ***here***[62].

***Hint:*** Refer to ***https://addyosmani.com/blog/essential-js-namespacing/*** for an excellent review.

***Why is it poor practice to have variables and functions on a global level?***
Because, ***if*** you are supplementing your code with 3rd party libraries and scripts, ***— which I shy away from for several reasons: 1) it distracts from development time to learn someone else's code; 2) your game is only as good as its "weakest imported library" —*** those additions all share the same global object. Furthermore, there is ***a chance*** those additional libraries ***might use*** similar naming conventions for their variables and functions; this ***could cause*** "name collisions" and override your code's logic. If that all sounds ***"totally paranoid"***[63] and ***"psychotic"***[64] — what, what?!

[60] https://www.youtube.com/watch?v=x2Y7_1dILlQ
[61] http://stackoverflow.com/questions/8862665/what-does-it-mean-global-namespace-would-be-polluted/13352212
[62] http://stackoverflow.com/questions/9773964/understanding-the-javascript-global-namespace-and-closures
[63] https://www.mind.org.uk/information-support/types-of-mental-health-problems/paranoia/effects-of-paranoia/
[64] https://medlineplus.gov/psychoticdisorders.html

Software engineers psychotic? Go figure! — ***you're right!*** The chances are even higher when libraries that use "$" as an alias (e.g.: JQuery, Prototype, PHP, and others).

### Concerns using Browserify with Phaser

***Quotes from "Phaser 2.7.5 Browserify"***[a], "Phaser was never written to be modular. **Everything exists under one single global namespace,** and you cannot require selected parts of it into your builds. It expects 3 global vars to exist in order to work properly: ***Phaser, PIXI, and p2.*** The following is one way of doing this:

```
window.PIXI = require('phaser/build/custom/pixi')
window.p2 = require('phaser/build/custom/p2')
window.Phaser = require('phaser/build/custom/phaser-split')
```

If you build a ***custom version*** of Phaser it will split the 3 core libs out into their own files, allowing you to require them as above. (ed.: I strongly DO NOT recommend doing this. )

***We appreciate this is just a band-aid, and not a proper use of modules, but please understand it was never built to be used this way.*** You're trying to fit a square peg in a round ***browserify-shaped hole,***[b] so compromises have to be made. Please don't open GitHub issues about it as we've no intention of changing Phaser at this stage of its life. ***Full module based development is available in Phaser v3 http://labs.phaser.io/***

[a] https://photonstorm.github.io/phaser-ce/index.html
[b] https://github.com/browserify/browserify-handbook

If you use 3rd party libraries, you might consider using ***namespace to isolate your code.*** A perfect example comes from ***"Mighty Editor"***[65]. Refer to the line numbers available ***here***[66]:

**Example: Creating Namespace for game**

```
118     // MightEditor namespacing: http://mightyfingers.com/
119     //   free on GitHub: https://github.com/TheMightyFingers/
120     "use strict";
121     window.GAMEAPP = {
122         // reference to the Phaser.Game instance
123         game: null,
124 
125         //Canvas dimensions: world and viewports' Height and Width
126         //**TODO** adjust for your game deployment
```

[65] http://mightyfingers.com/
[66] https://makingbrowsergames.com/book/demos/_index.html

```
        viewportWidth:  800,    //game view using Golden Ration
        viewportHeight: 500,
        worldWidth:     800,    //world view using Golden Ration
        worldHeight:    500,
        ...
        // main function
        main: function(){
            this.game = new Phaser.Game(
                this.viewportWidth, this.viewportHeight,
                Phaser.AUTO, document.body,
                window.GAMEAPP.state.boot);
        },
        // we'll store all game phases
        //  as the js files load.
        state: {}
    };

    /** DEPRECATED METHOD - NEVER EVER USE THIS AGAIN!
    * See Phaser.js Game Design Workbook for complete explanation
    * http://leanpub.com/phaserjsgamedesignworkbook
    * window.onload = function () {
    *    let game = new Phaser.Game(0, 0, Phaser.AUTO, document.body);
    * };
    */
    // preferred lauch method for BOM.
    window.addEventListener('DOMContentLoaded', function(){
        window.GAMEAPP.main();
    }, false);
```

## Project Directories & Files

```
.URI/<PROJECT NAME>/ //game root directory (single player)
├── favicon.ico      //game logo
├── index.html       //game front-door entrance
├── license.txt      //game EULA @ http://renown-games/License.txt
├── manifest.json    //game mobile app
├── package.json     //for Progressive Web Applications
├── purchaseOrd.pdf  //how to buy your game
├── ReadMe.md        //game info, contact and metadata
|
├── assets/          //game unique © resources
```

```
11      |   ├── audio/
12      |   ├── images/
13      |   └── spriteSheets/
14      |
15      ├── css/            //game content styling
16      |   └── main.css
17      ├── fonts/          //game font styling
18      |   └── fonts.css
19      |
20      └── js/             //game behaviors
21          ├── lib/        //game external source code libraries
22          ├── plugins/    //game enhancements
23          ├── prefabs/    //game prefabrication objects
24          ├── states/     //game phase
25          └── utilities/  //game helpers
```

- ***index.html*** — Main game container file, your example game should be viewed from within this page.
- ***.htaccess*** — The default web server configs are for Apache. For more information, please refer to the Apache Server Configs documentation[67]. Host your site on a server other than Apache? You're likely to find the corresponding server configs project listed here[68]
- ***apple-touch-icon-precomposed.png*** — If you want to use different Apple Touch Icons for different resolutions refer to this documentation[69].
- ***crossdomain.xml*** — A template for working with cross-domain requests. ( ***more about crossdomain.xml here***[70] ). ***WARNING:*** WebSockets **can AND do cross-domain communication,** it follows the same cross-origin sharing ***CORS methodologies***[71] and **they are not limited** by the "Same Origin Policy (SOP)", as JavaScript is traditionally inside the browser. Because of this, WebSockets have the same exposure to the types of cross-domain attacks. I won't go into detailed descriptions on WebSocket security, simply said, it's up to the server authentication to validate their client's origin and for WebSocket frame tampering. If you'd like to restrict browsers' communication to same-domain servers, you will modify the header policies in the browser Content-Security-Policy header. This will lock down the WebSocket to your originating domain. Naturally, you should always use "wss://" (Secure WebSockets ), to ensure a stronger level of encryption.
- ***favicon.ico*** — refer to Hans' handy HTML5 Boilerplate Favicon and Apple Touch

[67] https://github.com/h5bp/server-configs-apache/tree/master/doc
[68] https://github.com/h5bp/server-configs/blob/master/README.md
[69] https://github.com/h5bp/html5-boilerplate/blob/v4.3.0/doc/extend.md#apple-touch-icons
[70] https://web.dev/samesite-cookies-explained/
[71] http://en.wikipedia.org/wiki/Cross-origin_resource_sharing

Icon PSD-Template[72].

- ***human.txt*** — Edit this file to include the team that worked on your site/app, and the technology powering it.
- ***license.txt*** — describe how you permit the use of your game.
- ***purchaseOrder.pdf*** — never know how a consumer obtained your game release; provide them with a means to remain honest.
- ***readme.txt***[73] (***FILE_ID.DIZ***[74], or ***readme.md***[75]) should have a customer-friendly welcome, project introduction, installation instructions, license, and contact information.
- ***robots.txt*** — Edit this file to include any pages you need to be hidden from search engines.
- ***assets/*** — Any copyrighted assets (purchased or created) specifically for this game, or referenced in the index.html file. Everything should be in this folder.
  - ***audio/*** — Home for any audio files. You could simply name this directory "sounds" or "sound effects" (sfx). You might consider building two sub-directories for game theme "music" and another for "sound effects (sfx)". Remember that not all browsers support every audio format (.wav, .ogg, mp3/4). Try creating ***your own music here***[76] Research ***more demos***[77] from ToneJS — A Web Audio framework for making interactive music in the browser at https://tonejs.github.io. ***Learn about the differences between HTML5 audio and Web Audio here***[78]***.***
  * ***data/*** — Any data files (e.g. JSON, atlas) that pertain directly utilized by these assets.
  * ***fonts/*** — Any unique font-sets you have licensed
  * ***images/*** — Home for any visual files. You could simply name this directory "images", "sprites" or " graphics effects" (gfx). I stuff all the visuals here — including spriteSheets.
  * ***maps/*** — the information about tile-maps used in this game.
  * ***misc./*** — any additional files such as dialogs, run-time scripts, language XML/json, etc.
- ***data/*** — configurations, static data templates, tile maps, game board dimensions, etc.
- ***docs/*** — This directory contains all the HTML5 Boilerplate documentation and might contain any extra documentation about the Blueprint. You can use it as the location and basis for your own project's documentation.
- ***js/*** — Source JavaScript files for your game. You could simply name this directory "scripts" or "source (src). You could include Libraries, plugins, and custom code; or all can be included in a separate sub-directory or directory. It includes some

[72] https://drublic.de/archive/html5-boilerplate-favicons-psd-template/?file=blog/html5-boilerplate-favicons-psd-template/
[73] https://en.wikipedia.org/wiki/README
[74] https://en.wikipedia.org/wiki/FILE_ID.DIZ
[75] https://en.wikipedia.org/wiki/GitHub_Flavored_Markdown
[76] https://learningmusic.ableton.com/index.html
[77] https://tonejs.github.io/demos
[78] https://developer.mozilla.org/en-US/docs/Web/Guide/Audio_and_video_delivery/Cross-browser_audio_basics

initial JS to help get you started
* ***gameObjects/*** — Any core Game Objects (such as player.js, avatar.js, treasure.js, etc.) should be contained here.
* ***states/*** — All Game Phases/States menus used by your game.
* ***utils/*** — Folder containing any Utility Methods/Objects.
* ***game.js*** — The main JavaScript game mechanics logic file.

- ***lib/*** — External Libraries that are required/used should be contained here. This includes any JavaScript Framework and addons / extensions (a.k.a., ***Plugins***).
  * ***phaser.min.js*** or simply use from one of the content delivery networks.
  * ***plugins/*** — Any Plugins that are used.
- ***themes/*** — Folder containing any formatting for the overall hosting website.
  * ***CSS/*** — cascading styles sheets for the overall website theme. It should follow a ***"structured approach"***[79] creating separate cascading style sheets during development. Upon deployment, all of these collapse into a single file. This directory should contain all your project's CSS files. It includes some initial CSS to help get you started from a solid foundation.
  * ***gfx/*** — graphics effects for the hosting website.

***NOTE:*** Separate your style sheets for better management.[80]: typography, color, layouts, navigation, general formats styles

***Hint:*** "Development code is what you read and write, and "check-in" to your source control system. It should be highly modular (split over many files), extensively commented, and should make liberal use of whitespace to indicate structure. On the other hand, ***Machine code*** is what gets served up to a browser. It should consist of a small number of ***merged files,*** and should be stripped of comments and unnecessary whitespace. Your ***build process*** is a method to which you apply these transformations; many developers use the automated "Grunt"[81]. Finally, your web server should deliver the machine code with gzip compression for extra speediness." Read more tips here[82]

***Exercise: Download this Phase Game Prototype starter kit here (35 MB zipped)***[83] ***https://makingbrowsergames.com/book/standalone.zip***

***Exercise:*** Read what others say about ***"How Do I Organize Files in A Phaser.js Project?"***[84]

[79] https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/
[80] https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/
[81] https://24ways.org/2013/grunt-is-not-weird-and-hard/
[82] https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/
[83] https://makingbrowsergames.com/book/standalone.zip
[84] https://glcheetham.name/2016/03/18/organise-files-phaserjs-project/

## 1.3 Game Project "Concept & Design"

***"The time something will take depends on how much time you allot to it." — Parkinson's law***[85]

By the end of this section, you will have walked through the ***Game Design System™*** method of building a ***Game Recipe™.*** Later in this book, we'll automate this process and develop those tools to do so. But for now, let's step through the "design process" of creating a ***Game Recipe™.***

***Hint:*** The ***Game Design Appendix*** offers many suggestions from experts in the gaming industry. Hopefully your product/project manager has already thought about the follow guidelines. **The following is an excerpt from the Phaser Game Design Workbook (6th edition).**

### Introduction to Game Design System™

***"Why study a "systems-based" design?", you say?***

The earliest decisions about what kind of games a studio will build impacts the following development and production activities for that game project. It affects

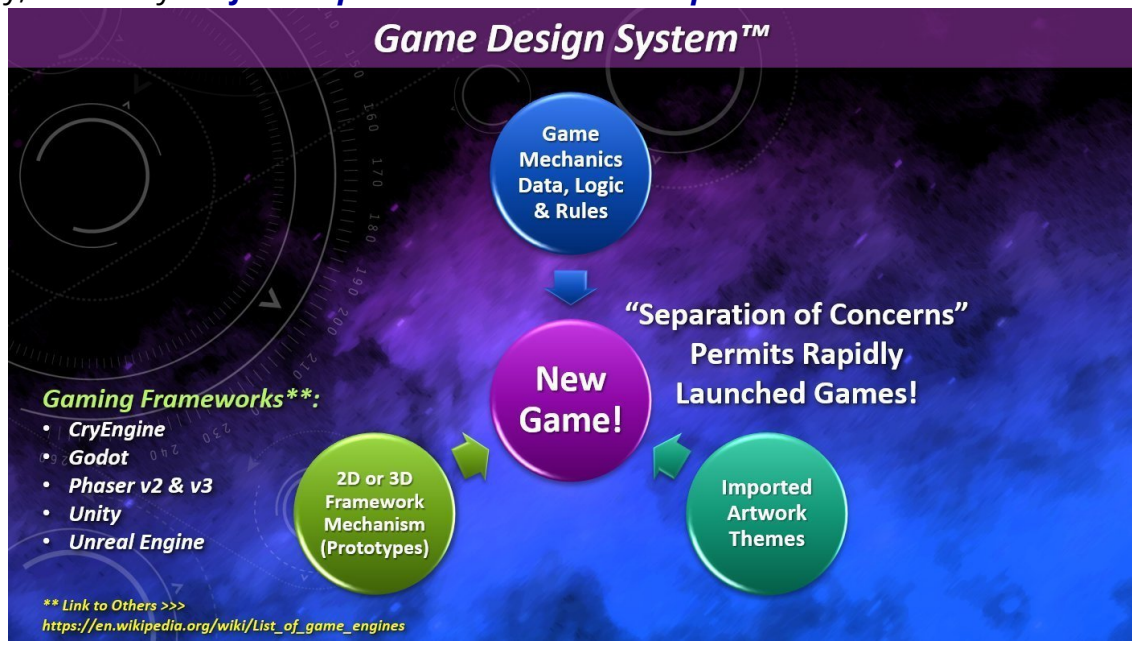
- how the programmers encode features,
- how the designers construct levels and optimize game mechanics, and
- how the time-consuming animations and "cut-scenes" are handled … just to mention a few!

There's also a "heavy-cost" associated with how much creative freedom is permitted. Historically, games with "open-ended" possibilities tend to be much more difficult to accurately schedule. ***Game Design System™*** addresses those short-comings in this new game prototyping approach.

Creating your own game is an exciting adventure in creativity using the Phaser III JavaScript Game Framework ***— any version beyond v3.16+ —*** (or ***with any Gaming***

[85] https://en.wikipedia.org/wiki/Parkinson%27s_law

***Framework***[86] for that matter); and, at the same time, it's fun! However, dealing with all those ***"full-stack" technical details***[87] — such as web pages, ***artwork production***[88], collisions, sprites, "game phases" (and there's more!) — can be ***intimidating;*** especially, if this is your ***first experience with such components.***[89]




**Phaser Game Design Workbook**

JavaScript ***is NOT the end-all*** for online gaming! To ***"future-proof"***[90] your time spent developing a game, you should begin reading about the ***"Internet-of-Things" (IoT)***[91], cloud-based ***"DevOps"***[92] and ***"web assembly"***[93]. You should begin using "Separation of Concerns" (SoC) on your game pipeline to segregate those "Game Mechanic" (GM) from their "visual elements" governed by the "Game Framework Mechanisms" (GFM) and further isolate your "Artwork" themes from their "visually displayed mechanisms" which includes them.

***JavaScript Gaming Frameworks***[94], in general, are resource libraries that run inside a browser or an Internet device. Any of these "frameworks" simplify and abstract the ***gaming patterns and algorithms***[95] into an easy-to-use "interface" of JavaScript functions. Using them, you can quickly build various two-dimensional ( aka "2D" or ***"2.5D"***[96]

[86] https://ourcodeworld.com/articles/read/308/top-15-best-open-source-javascript-game-engines
[87] https://www.w3schools.com/whatis/default.asp
[88] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[89] http://www.michael-richardson.com/processes/rup_for_sqa/core.base_rup/guidances/supportingmaterials/use_component_architectures_CBC2F6B5.html
[90] https://medium.com/@george3d6/stop-future-proofing-software-c984cbd65e78
[91] https://www.sas.com/en_us/insights/big-data/internet-of-things.html
[92] https://azure.microsoft.com/en-us/overview/what-is-devops/
[93] https://developer.mozilla.org/en-US/docs/WebAssembly
[94] https://blog.bitsrc.io/9-top-js-gaming-engines-and-libraries-for-2020-81707d9f095
[95] https://web.archive.org/web/20200516234514/http://gameprogrammingpatterns.com./
[96] https://en.wikipedia.org/wiki/2.5D

) and even ***"3D"(!!)***[97] games inside a simple ***HTML5 "<canvas>" tag***[98]. The ***Phaser JavaScript Gaming Frameworks***[99] does ***90% of all that work for us;*** and beyond that, all you need is your imagination and some basic JavaScript programming knowledge that you can learn ***for FREE from W3Schools***[100]**.** So, let's begin by creating simple game prototypes and mechanisms while exploring many of the basic concepts found in the ***Game Design System™*** from the former ***Phaser III Game Design Workbook***[101] and newest ***Headless Game Design Workbook***[102].

## What makes a Good Game?

***MMMM, something smells good … What's cookin'?***

A ***"good game" (18-page Bonus Content!)***[103] is a matter of **personal preference.** If you intend to market your game and earn your "just rewards", then you need to research those game genres people enjoy consuming. The hard reality becomes what others enjoy may not be what inspires you. Starting a small simple game release isn't the real problem. The problem truly is finding an idea that …

1. nobody's tried before (i.e., ***technology break-through*** ), or
2. improves upon an existing game genre with *"better enhancements"* (i.e., ***innovative*** and **again, "better" is a matter of taste**), or
3. is distinct from anything currently on the market (i.e., ***novelty***).
4. focused on business drivers for a target market.

> "Don't copy blindly, but try to **do things differently instead of doing what everyone else is doing.** Think about how you can stand out from the hundreds of thousands of other games. Surprise and delight. **It doesn't cost anything to surprise people.** That said, learn as much as you possibly can from others. Then do your own thing." (Peter Vesterbacka, pg 401 ***"Building javaScript Games"***[a])
>
> [a] https://amzn.to/2D8c7qM

[97] https://en.wikipedia.org/wiki/Three.js
[98] https://www.w3schools.com/graphics/game_canvas.asp
[99] http://phaser.io
[100] http://www.w3schools.com
[101] https://leanpub.com/phaser3gamedesignworkbook
[102] https://leanpub.com/hgd
[103] https://makingbrowsergames.com/book/WhatMakesaGoodGame.pdf

***HINT:*** Generally speaking, if a game has over 50% of a ***market's share,***[104] it'll be difficult to take on and defeat that ***"Boss"***[105].

The best place, to begin our ***Game Recipe™,*** is to ***jot down***[106] your ideas on paper — that's right, draw what you're visualizing — what is your mind "***cooking up***[107]"? This will help clarify your ideas into a tangible form (i.e., **securing your Copyrights!** That topic's coming up!) and figure out what needs to be done "in what order" and "with what priority". You're building a game prototype; you can think of a prototype as a **"recipe" for an gaming thing!** That's what this whole ***Game Design System™*** is all about. Your doodles become a true skeletal framework/engine with **"new muscles (game prototypes), organs (game mechanics) and flesh ( the artwork!**[108] **).** When everything comes together, you'll start shouting ***— in your best Dr. Frankenstein accent —*** "IT'S ALIVE!" However, you might also find, as details are "fleshed out", there are some inconsistencies or discover perhaps things that need more clarification.

***Exercise:*** Read ***Making Games Fun by Burak Kanber***[109] for some excellent suggestions.

***Exercise: Download the Terry Paton mind-map on "What makes a Great Game"***[110] or find it in your Bonus Content (file name: WhatMakesAGreatGame.pdf).

***https://makingbrowsergames.com/design/WhatMakesAGreatGame.pdf***

## 1.4 Preparing a "Gaming Product"

***otherwise known as "Planning your Game Project"***

If you want to develop game recipes, you should know how to program in JavaScript; but more importantly, you ***must know*** how to create common solutions for various programming problems. In addition to the problem solving skills, some of those

[104] https://economictimes.indiatimes.com/definition/market-share
[105] https://en.wikipedia.org/wiki/Boss_(video_gaming)
[106] https://idioms.thefreedictionary.com/jot+down
[107] https://www.merriam-webster.com/dictionary/cook%20up
[108] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[109] http://buildnewgames.com/making-games-fun/
[110] https://makingbrowsergames.com/design/WhatMakesAGreatGame.pdf

solutions may be more generic than others, and some of the solutions may become more efficient than others. In this regard, game programming is sometimes a trade-off between solving ***a specific*** game issues quickly or taking ***more time to resolve a whole category of problems*** at once. In game development, there's often less time to solve game construction issues because of tightly-mandated deadlines. So, we need to think about our development approach very carefully. Our ultimate goal in game prototyping is to write ***"nice"***[111], reusable code which won't always take more time than writing ***"quick-and-dirty" code.***[112] As you gain more experience in this ***Game Design System™***, you'll find that you'll start developing a mindset that lets you quickly gauge the kind of solutions that are required for a certain gaming problems.

## Why are you doing this?

Before we go any further, let's determine **"Why"** you want to create your game. Our next workbook exercise question is:

***Exercise:*** Do you plan to create this game:

- **As a hobbyist**? In other words, generating income **is NOT your primary motivation**. You simply want to "add onto" your skills or seek the challenge of creating a similar popular game currently in the "apps stores".
- **As an academic pursuit?** In other words, your primary motivation is to **study and experiment with the most cutting-edge technology**.
- **As a way of generating revenues?** In other words, your **primary motivation is to supplement or replace your current income source**.

Answering this exercise question will guide many of the following production decisions. So, go fetch some paper or open a file and record your answers from the question above. Write your answer down. Become an active participant, and learn the most important concept — **"Journaling and logging"**. By doing so, you are developing a time-line of your activities (i.e., what is easy for you to do, what poses difficulties for you, and who to hire for additional staff support). This helps determine the amount of time it takes to put a profitable game into your distribution channels. When your customers ask you, "When can I have the finished product?"; you have proven empirical evidence based your project's development schedule — not some "pie in the sky", ***UN-realistic time-frame*** to which so many business fall victim!

[111] https://en.wikipedia.org/wiki/Nice_(Unix)
[112] https://www.urbandictionary.com/define.php?term=quick-and-dirty

***Exercise:*** Read why so many business fall victim to poor time management in this article: ***Scrum makes you dumb***[113] ...

> (an excerpt) ***If your software developers are able to accurately estimate how long something will take, you should fire them.*** If they've done something so many times before that they know exactly how long it'll take them to do it again, then ***they should have made a reusable solution by now. (ed.: a game prototype!)***

## What are you making?

When I first started game production in the mid-70s, I found myself constantly thinking of new game ideas and ***jotting them down.***[114] I became addicted to the "creation process" and the power of bringing my thoughts into something physically tangible. There was so much I wanted to make. If you identify with that sensation, then you probably have some game ideas already on what you'd like to make. Do you have your own list?

## What technology will you use?

***You must be thinking ... "DUH?!? Phaser of course! Why even consider this?"***

We must wisely choose a gaming framework to use, since we ***simply don't have the time*** to study and master every new "bleeding-edge" library, framework, or ***"brain-fart"***[115] appearing on the ***technology horizon — such as game-based learning***[116]? To help us maintain focus and guide us in our selection, here a list of probing questions:

- Is the framework or library ***well-used?***[117] If it has a forum community following its development, then it becomes more likely that it also has contributors, frequent improvements on its key features, and rapid software bug resolutions. Furthermore, it is more likely to have ***"staying power"***[118] and stamina.

[113] https://www.linkedin.com/pulse/scrum-makes-you-dumb-daniel-jones/
[114] https://www.collinsdictionary.com/us/dictionary/english/jotting
[115] https://www.urbandictionary.com/define.php?term=Brain%20Fart
[116] https://www.researchgate.net/publication/216566471_What's_on_the_Technology_Horizon
[117] https://www.thesaurus.com/browse/well-used
[118] https://www.merriam-webster.com/dictionary/staying%20power

- Who comprises its supporting community? Are they corporations, universities, or passionate hobbyists? How does the community respond to each other … with civility or impassioned fanatical opinions or ***one-upmanship?***[119] What is their ***"welcome wagon"***[120] for new users.
- How often is community content initiated and updated? It would be a sad day to discover a software bug and wait for a response to come years later.
- Are there frequently released versions with dramatic changes in API or architecture? You don't want to revisit your entire product line and refactor, republish, and redistribute your entire collection portfolio for frequently "breaking changes"! Furthermore, backward compatibility may pose significant problems within your marketing channels and worldwide distributions.
- What currently active features make this framework (or library) better than its competitors? What technological innovation is it based upon? Is that technology widely available to your client-base?
- Does the framework match your development team's capabilities? For example, if you have junior/student developers, does the framework provide tutorials, ***completed*** documentation and architecture ***explanations.***
- Speaking of supporting materials, is the documentation consist of quality professional content compared to ***naive descriptions easily deduced*** from merely reading the source code?
- Is the framework "open-source" or commercial? Do you have access to the ***raw annotated source code*** or to a compiled release only? Are you able to extend, modify or supplement the framework ***legally?***
- Is the framework truly ***performant***[121] or merely just ***an abstraction layer?***

***Exercise:*** Read this article about ***"Shiny New Objects"***[122]

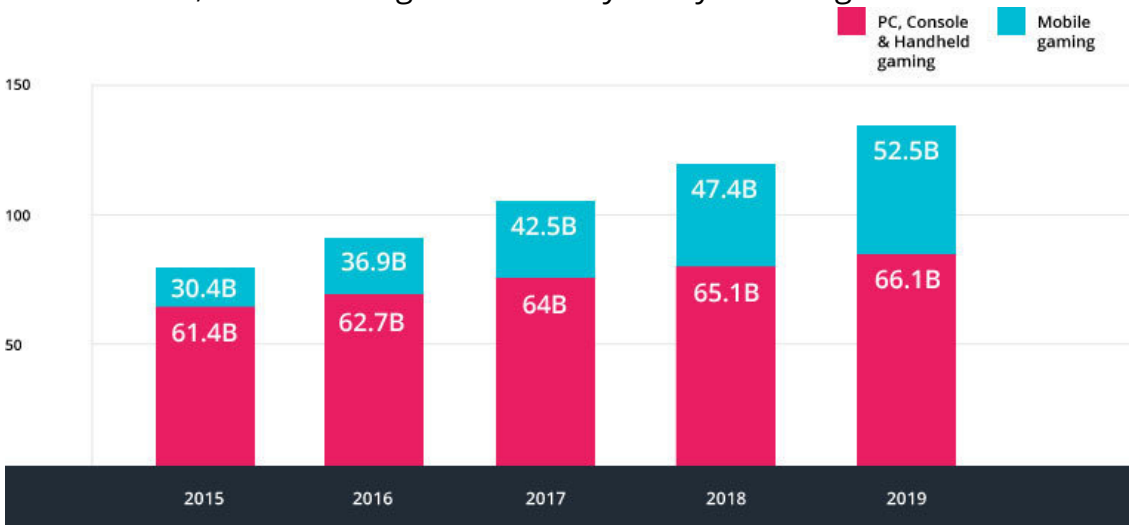
***Exercise:*** Read ***Game Making Tools Features and Comparisons***[123] to learn the 180 degree shift in game industry development.

Well, as difficult as this may sound, Phaser v2.x.x may not currently support your game's *"ultimate dream features".* It *might* in the future, and there are some pretty impressive features already in Phaser v2.x.x ***and v3.24+!*** **But, unless you ask** for those features or better still discuss them in the forums, you may have to look elsewhere to avoid barriers to your development ***— keep that thought in mind!***

Before we leave the topic about *"What technology …"*, I'd like to bring your attention to something I've discovered recently. Phaser Gaming Framework needs your voice

[119] https://dictionary.cambridge.org/us/dictionary/english/one-upmanship
[120] https://dictionary.cambridge.org/us/dictionary/english/welcome-wagon
[121] https://www.techopedia.com/definition/28231/performant
[122] https://dev.to/aspittel/navigating-the-spooky-world-of-javascript-3h45
[123] https://instabug.com/blog/game-making-tools/

in the gaming industry. It's an alarming fact, that can't go **"UN-noticed",** with the coming technology trends. The following chart from ***Instabug on Mobile Game Development***[124] shows a significant growth in mobile gaming. Phaser v2.x.x Gaming Framework alone may not fulfill everything required to enter the mobile/WebXR market. Furthermore, Phaser III might not be fully ready for a migration into ***WebXR***[125].




**Video game market revenue worldwide by Instabug Blog**

***Exercise:*** Read this report from ***Instabug on Mobile Game Development***[126].

Let me guide your attention to the "News Press Releases" so far this year (**as of 20180901 and 20200924! - Google's 1st page listing** for the search term "**The Best Game Development Tools 2018**[127]" and "**The Best Game Development Tools 2020**[128]" ).

- ***16 Best JavaScript Game Engine***[129] May 14, 2018 (5 years after formal Phaser v1 released; 18 months from the last official v2.6.2. and 3 months after Phaser III.)
- Best Dame Development Tools By James Konik[130] — Last Updated: 18 Jun'18 (18 months after formal Phaser v2.6.2 released and 4 months after Phaser III.)
- The Best 15 Mobile Game Development Platforms & Tools in 2018 By Cristina Stefanova[131] — April 25, 2018 (16 months after formal Phaser v2.6.2 release.)

[124] https://instabug.com/blog/mobile-game-development-tools/
[125] https://www.w3.org/TR/webxr/
[126] https://instabug.com/blog/mobile-game-development-tools/
[127] https://thetool.io/2018/mobile-game-development-platforms
[128] https://thetool.io/2018/mobile-game-development-platforms
[129] https://www.dunebook.com/16-best-javascript-game-engine/
[130] https://www.cloudwards.net/best-game-development-tools/
[131] https://thetool.io/2018/mobile-game-development-platforms

- Game Development Software by Capterra[132] The Smart Way to Find Business Software
- Top 5 Mobile Game Development Tools 2018[133] 17th July, 2018 (19 months after formal Phaser v2.6.2 release.)
- 7 Best Game Development Tools Of 2018 That Will Revolutionize The IT Industry by Henry Kundariya[134] March 18, 2018 (16 month after formal Phaser v2.6.2 release.)
- Best Game Development Software by G2 Crowd[135] (publication data unavailable)
- Mobile game development in 2018: best tools and advice[136] (publication data unavailable)
- The Most Recommended Game Development Tools and Engine of 2018 For Game Dev[137] (publication data unavailable)
- In-Depth Comparison of the Top Game Making Tools of 2018[138] (publication data unavailable)

***NOTE:*** Google first page is an indication of current trends in SEO. The listing are dynamic and will dramatically change over time.

***Exercise: History Lesson —*** Read the development ***history of Phaser from KiwiJS By Rich Davey***[139] Rich Davey quoted, "... On April 5th 2013 I started work on the very first version of Phaser. A couple of weeks later, on April 12th, version 0.5 was pushed up to GitHub and the rest, as they say, ***is history."***[140]
***Exercise:*** Do your own "Google Search" — with the latest up-to-date information! — for this search term ***"The Best Game Development Tools (insert current year)";*** or use my researched list above (dated: 20180901). Inside each article, try and find the word "Phaser" or "Phaser JavaScript Gaming Framework". Count how many time it appears. **Then answer this question: What technology will you use (or supplement) your mobile/WebXR game development?**

[132] https://www.capterra.com/game-development-software/
[133]
[134] https://www.linkedin.com/pulse/7-best-game-development-tools-2018-revolutionize-henry-kundariya/
[135] https://www.g2crowd.com/categories/game-development
[136] https://thinkmobiles.com/blog/mobile-game-development-tools/
[137] https://blog.sagipl.com/game-development-tools/
[138] https://instabug.com/blog/game-making-tools/
[139] http://www.html5gamedevs.com/topic/4281-kiwijs-vs-phaser/
[140] http://phaser.io/news/2016/04/phaser-is-3-years-old

## *"Loose lips sink ships" ...* and revenues!

So let us help Phaser Gaming Framework by discussing what we're planning for our future games and those features we hope to use. Discussing your ***innovative ideas*** in a public forum?!? ***Mmm,*** let's stop and think this through to its logical conclusion and consequences; and then, ***don't complain when your "idea" was "stolen" by someone. Simply stated: IDEAS***[141] ***are not Copyright-able!*** You should read what the US Copyrights Office says.

***Exercise:*** Read ***Works Unprotected by Copyright Law***[142]

> Quoted from: *http://www.copyright.gov/fls/fl108.pdf*
>
> ***Copyright does not protect the idea for a game, its name or title, or the method or methods for playing it. Nor does copyright protect any idea, system, method, device, or trademark material involved in developing, merchandising, or playing a game. Once a game has been made public, nothing in the copyright law prevents others from developing another game based on similar principles.***
>
> ***Copyright protects only the particular manner of an author's expression in literary, artistic, or musical form***

***Exercise:*** Study what items are "unprotected" in the Copyright Act. Read about ***Ideas, Web Blogs concerning Useful articles, and other such "WORKS UNPROTECTED BY COPYRIGHT LAW"***[143]
***Exercise:*** Research if ***"copy-left"*** is a valid form of EULA or implied license according to ***US Government Copyrights Office.***

> QUOTE: ***US Government Copyrights Office.***[a]
> ***An implied copyright license is a license created by law in the absence of an actual agreement between the parties.*** Implied licenses arise when the conduct of the parties indicates that some license is to be extended between the copyright owner and the licensee, but the parties themselves did not bother to create a license. **This**

[141] https://www.bitlaw.com/copyright/unprotected.html#ideas
[142] https://www.bitlaw.com/copyright/unprotected.html#ideas
[143] https://www.bitlaw.com/copyright/unprotected.html

> **differs from an express license in that the parties never actually agree on the specific terms of the license.** The purpose of an implied license is to allow the licensee (the party who licenses the work from the copyright owner) some right to use the copyrighted work, ***but only to the extent that the copyright owner would have allowed had the parties negotiated an agreement. (ed.: OMG, copy-left is wrong and can't enforce their claims?!!!)*** Generally, the custom and practice of the community are used to determine the scope of the implied license... .
>
> A commonly discussed scenario where implied licenses are destined to play a major role **is on the World Wide Web.** When a Web page is viewed in a Web browser, the page is downloaded through the Internet and placed on the user's screen. **It is clear that a copy of the Web page is being made by the user. It is also clear that the Web page is protected against unauthorized copying by copyright law.** (ed.: our modern laws need to be updated to society's current behavior.) But it would not make sense to allow the author of a Web page to sue a user who viewed her page, ***since the author intended that the page be viewed by others when she placed it on the World Wide Web.*** (ed.: author's original intent is marketing their content.) Rather, attorneys argue, courts should find that the Web page author has ***given end users an implied license*** to download and view the Web page. The extent of this implied license is unclear, and may someday be defined by the courts.
>
> [a] https://www.bitlaw.com/copyright/license.html

And hence, the reasons to write down our ideas in a ***"tangible form"***[144]; and furthermore, affix a properly labeled notice — ***using a legal ©***[145] prior to forum discussions (i.e., none of this stuff: ***"(c)", "pen names", "pseudonym" defecation*** or missing publication dates). That "tangible" form should be a game description. Let's take for example, a simple "Breakout" game. You might write your game description similar to this ***"elevator speech"***[146]. Naturally, you'll create a description about your own game; but for now, this should give us enough of **an idea** to continue our planning process with my game's description:

> **Breakout:** a game in which a player uses a sliding paddle along the bottom of the screen. They control the paddle's movements to collide with an animated ball causing it to bounce upwards or at various angles toward a grid of blocks. The game's objective is to hit all those blocks, while at the same time not letting the ball pass-by the paddle and fall off the bottom of the screen too many times.
>
> Use proper copyright notices:

[144] https://www.bitlaw.com/copyright/formalities.html
[145] https://en.wikipedia.org/wiki/Copyright_notice#Form_of_notice
[146] https://en.wikipedia.org/wiki/Elevator_pitch

***Exercise:*** Take a moment and jot down your game's description.

***Hint:*** I strongly encourage you to purchase ***"How to Copyright Software"*** by M. J. Salone[147] (a lawyer!) who shows information ***"over-looked"***[148] by the ***copy-left movement*** and ***open-source software licenses.***

## What features are included?

This is the planning stage ***where dreams are turned into real tangible items,***[149] and where it gets fun, in my opinion. In this step, our goal is to figure out what we're actually making — in other words, what will the game look like, what features it includes, and what features it won't include or that won't appear initially.

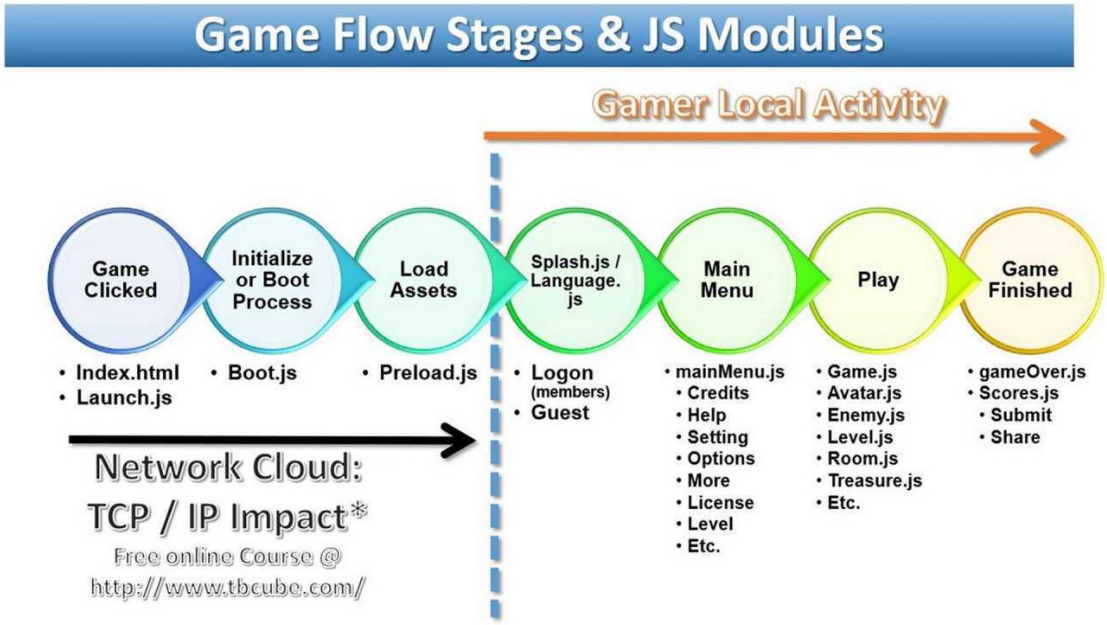
The first thing we can do is make a paper "mock-up" — sketches that look like the thing you're making, but without any details like coloring or exact sizing. You can make mock-ups on paper, or an online program if you prefer.

To give you an idea of what a mock-up might look like, I've included my mock-ups below for the "Breakout" game example. This becomes our "road map". Next, I'll sketch each "game phase" separately and have lines connecting those "visual displays" to show how one "menu" leads a player into another "visual section". Those lines help me understand what code I need in my game program to move between the various "game stages".

[147] http://amzn.to/2bmlAcH
[148] https://en.wikipedia.org/wiki/First-sale_doctrine
[149] https://www.youtube.com/watch?v=ZXsQAXx_ao0


**Simple Game Phase Flow for an RPG game**

Here's a more thorough illustration on my various "game phases" with a brake-down of JavaScript recommendations.



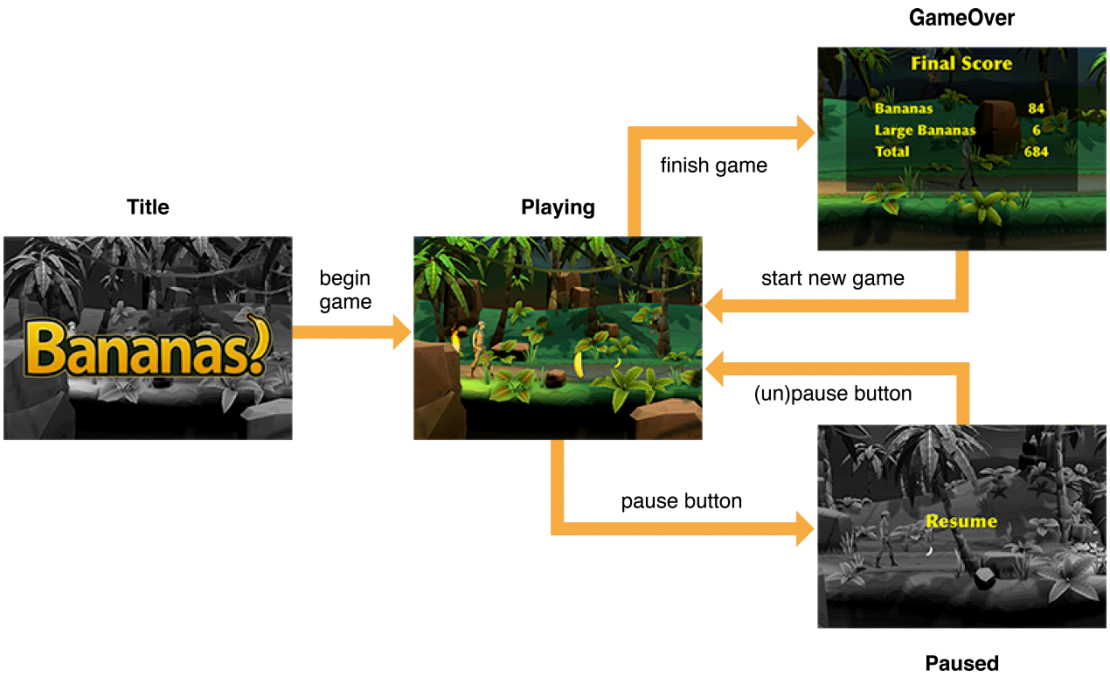

**Typical Game Phases from Phaser Game Design Workbook**

Below is an example from the ***Apple's Game Kit***[150] using a ***Finite State Machine (FSM)***[151] to manage a game's navigation and "user interfaces" (UI). I'm recommending

[150] https://developer.apple.com/documentation/gamekit

[151] https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867&ref=PBMCube

"Apple" since they are following more what I'm suggesting in the ***Game Design Systems***™**.** (more on FSM in later chapters)




**A 'Finite State Machine' for Game UI navigation**

Now we can use these drafts to help create a our game's features list. This is the step where we think of every possible feature we can imagine in our game. And take those ideas and put them into a list. Don't limit yourself just write everything down — ***BRAIN STORM!!*** — we sort these ideas later.

## Deconstruction

Quote Phaser Game Design Workbook, page 69, "From a game programming perspective, basic **Game-Play** can be deconstructed — revealing tactical components inside a game's overall mechanics and rules. For example, a fighting game deconstructs into various tactics such as attacks (or punches, throws, and kicks), defensive moves, and dodges. These tactics are assigned to ***game prototypes and mechanisms***[a] — those input keys, mouse clicks, and mobile screen interactions. These maneuvers are further enhanced into strong or weak punch/kick from the various input controls. Therefore, **game control mechanisms (e.g., buttons, mouse, touch-screen)** are more of an engineering programming concept while **Game-Play** is more of a design heuristic concept that we'll study later.

[a] http://gameprogrammingpatterns.com/command.html#configuring-input


***Game Design System*™ creating new Games from 3 Components!**

***Exercise: Component-based architecture*** and development is different from ***MVC***[152]. Discover those ***differences here …***[153] "An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data)." (***Wikipedia***[154] )

By combining all of our game mechanisms, game mechanics and rules systems — as ***non-invasive aspects in our gaming product***[155] — along with an ***artwork theme***[156], we're able to create multiple game products quickly. It simply becomes a matter of exchanging any of those ***"3 cross-cut" components***[157] into a new innovative-mixture for a ***"tasty"*** new game product. ***This is the secret in concocting a new game***[158] ***every month or even every week!*** For example, swopping a "Guitar Hero" artwork theme with a garden-variety "Plants & Zombies" artwork. The new game would use the same "Game Mechanics" and "Game Display Mechanisms" but with a "fresh organic-garden" look and feel!

> Quote: ***"If we're pasting the same code changing a few variables or arguments each time to fit the current context. That code is a prime candidate for AOP." Read entire***

[152] https://www.tutorialspoint.com/mvc_framework/mvc_framework_quick_guide.htm
[153] https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238
[154] https://en.wikipedia.org/wiki/Component-based_software_engineering
[155] http://know.cujojs.com/blog/oop-is-not-my-hammer
[156] https://craftpix.net/categorys/2d-game-kits/?affiliate=112238
[157] http://know.cujojs.com/tutorials/aop/intro-to-aspect-oriented-programming
[158] https://www.vocabulary.com/dictionary/concoct

***article***[a]

[a] https://hackernoon.com/aspect-oriented-programming-in-javascript-es5-typescript-d751dda576d0

***Exercise:*** Make a list of features you'd like in your game. Go wild and write down any and every idea!

Returning to my Breakout example, this could be a our potential game feature list,

**Broken down by Game Scenes:**

```
Game Play scene has following Game Mechanisms
    - User-controlled paddle
    - Multiple colored bricks
    - Angled ball movement
    - Collision detection
    - Life display
    - Score display
    - Sound effects
Main Scene
    - "Play" (button)
    - "Help" (button)
    - "Credits" (button)
Help Scene
    - Headline: "Game Instructions"
    - Text: "(explain how to play)"
    - "Return" to Main Menu (button)
Credits Scene
    - Headline: "Credits"
    - Text: "(about me & partners)"
    - "Return" to Main Menu (button)
Win Scene
    - Headline "You're Awesome!"
    - Text "(List of Scores)"
    - [background Fireworks] (animation)
    - "Return" to Main Menu (button)
Lose Scene
    - Headline: "So Sorry!"
    - Text: "Boo Hoo! Play Again?"
```

```
29          - "Restart" (button)
30          - "Return" to Main Menu (button)
```

## What features are mandatory?

If we had unlimited time to make all the game programs we could ever dream up, then they'd include every feature from our lists. But, unfortunately, none of us have that much dedicated free time! ***(If you do, please let me know! I could use some extra hands on deck!***[159]***)*** So in this next step, we must decide which features are the most important, and which features we'll postpone until we have more time to include in later game-release updates. This step further helps us figure out our features' priorities — that is, where to begin by writing our "most important features" down to our "least important".

Let's ask ourselves these questions to help sort the importance of each feature:

- If I shared this with a business sponsor, which features should be working? In other words, what is my ***vertical slice?***[160]
- Which features am I the most excited about building? Passion is an ***important ingredient*** in our ***Game Recipes™!***
- Which features are the most unique in my game? These will differentiate our final product from our competitors as unique entertainment and novel — new or unusual in an interesting way.
- Which features will I learn the most from implementing? Knowledge generation is a valuable asset that many game experts seek depending upon game product deadlines.
- Are there any features that seem too far beyond my current skills or capability level? You can always include them in a following release or game update. This is what ***Richard Davey***[161] is doing — ***hiring additional staff,***[162] paying bounty on bug fixes and raffling off prizes for completing his Phaser v3.x.x technical documentation. Or consider hiring ***game development contractors who have the abilities.***[163]

Now, let's go through our feature list, and begin sorting. I like to use an excel spreadsheet and add a ranking column along side to each feature name. Doing so, helps me sort faster.

[159] https://www.youtube.com/watch?v=kxUdFQ6N_OI
[160] https://en.wikipedia.org/wiki/Vertical_slice
[161] https://blog.github.com/2016-04-12-meet-richard-davey-creator-of-phaser/
[162] https://www.patreon.com/photonstorm
[163] https://www.indeed.com/jobs?q=Game+Design+Contractor&l=

For the Breakout example, I've used a priority column (next to the features column) with "[1]" to show my top priority, "[2]" for my middle priority, and "[3]" for lowest priority. I've decided to set those unique game mechanisms' priorities higher than those simple general game features such as scenes, because I've learned that those general game features are typically `game prototypes` I've already created in other games:

**Visual elements are the Game Mechanisms**

```
    [1] Game Play scene has following  "visuals" (Game Mechanisms)
        [1] User-controlled paddle (game object)
        [1] animated ball (game object)
        [1] Multiple colored bricks (game object)
        [1] Angled ball movement (coded)
        [1] Collision detection  (coded)
        [2] Life display (text)
        [2] Score display (text)
        [2] Sound effects (coded)
    [2] Main Menu Scene (game phase/menu)
        [2] "Play" (button)
        [3] "Help" (button)
        [3] "Credits" (button)
    [3] Help Scene (game phase/menu)
        [3] Headline: "Game Instructions" (text)
        [3] Text: "(explain how to play)" (text)
        [3] "Return" to Main Menu (button)
    [3] Credits Scene (game phase/menu)
        [3] Headline: "Credits" (text)
        [3] Text: "(about me & partners)" (text)
        [3] "Return" to Main Menu (button)
    [2] Win Scene (game phase/menu)
        [2] Headline: "You're Awesome!" (text)
        [3] Text: "(List of Scores)" (text)
        [3] [background Fireworks] (animation)
        [3] "Return" to Main Menu (button)
    [2] Lose Scene (game phase/menu)
        [2] Headline: "So Sorry!" (text)
        [3] Text: "Boo Hoo! Play Again?" (text)
        [3] "Return" to Main Menu (button)
```

Notice that I've added a brief description in parenthesis after each item. It is a naming "Category classification" I use in my game prototypes.

You can see my Excel spread-sheet can easily sort the priorities into game project

tasks, so you can easily see what you need to implement in each ***SCRUM Sprint***[164], and you can always stop after a particular iteration and just be happy with what you've made.

**Sample Sprint Backlog by priority:**

```
    Game has the following Game Mechanisms
    [1] User-controlled paddle (game object)
    [1] animated ball (game object)
    [1] Multiple colored bricks (game object)
    [1] Angled ball movement
    [1] Collision detection
        [2] Life display (text)
        [2] Score display (text)
        [2] Sound effects
        [2] Main Menu Scene (game phase/menu)
        [2] Play (button)
        [2] Win Scene (game phase/menu)
        [2] Headline: "You're Awesome!" (text)
        [2] Lose Scene (game phase/menu)
        [2] Headline: "So Sorry!" (text)
            [3] Help (button)
            [3] Credits (button)
            [3] Help Scene (game phase/menu)
            [3] Headline: "Game Instructions" (text)
            [3] Text: "(explain how to play)" (text)
            [3] Return to Main Menu (button)
            [3] Credits Scene (game phase/menu)
            [3] Headline: "Credits" (text)
            [3] Text: "(about me & partners)" (text)
            [3] Text: "(List of Scores)" (text)
            [3] [background Fireworks] (animation)
            [3] Text: "Boo Hoo! Play Again?" (text)
            [3] Restart (button)
```

Refer: Sprint Backlog by ***priority here***[165]

This next chart is **my preferred method.** It helps me identify how many items I need to create. This **Second attempt** is the ***chart sorted alphabetically by Game Mechanisms:***

[164] https://www.scrum.org/resources/what-is-a-sprint-in-scrum
[165] https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/sprint-backlog

```
// Second Chart: (Categories then original content)
(animation)          [background Fireworks]
(button)             Play
(button)             Help
(button)             Credits
(button)             Return to Main Menu
(button)             Restart
(coded)              Angled ball movement
(coded)              Collision detection
(coded)              Sound effects
(game object)        User-controlled paddle
(game object)        animated ball
(game object)        Multiple colored bricks
(game phase/menu)    Main Menu Scene
(game phase/menu)    Win Scene
(game phase/menu)    Lose Scene
(game phase/menu)    Help Scene
(game phase/menu)    Credits Scene
(text)               Life display
(text)               Score display
(text)               Headline: "You're Awesome!"
(text)               Headline: "So Sorry!"
(text)               Headline: "Game Instructions"
(text)               Headline: "Credits"
(text)               Text: "(explain how to play)"
(text)               Text: "(about me & partners)"
(text)               Text: "(List of Scores)"
(text)               Text: "Boo Hoo! Play Again?"
```

***Hint:*** The ***"400 Project Rule List"*** contains more than 100 game design rules. It is an ongoing formal study of gaming rules, together with attribution, scope, and trumping information that all may help create "game prototype categories. ***Download the list from here.***[166]

If you scan through the first listing, you'll discover many items begin to cluster together. For example `(game object)` are at the top of the list while `(button)` are mostly a lower priority. Alternately, I could have created a separate spreadsheet column for just those *Game Mechanisms* items and sort just those categories. (Refer to the Second Chart above) Doing so would tell me "common" items in my game, and let me write that code once and reuse it for similar items in other games as a component

[166] https://web.archive.org/web/20190417062220/http://www.finitearts.com/pages/400page.html

prototype ***— this is the secret sauce in our Game Recipes™! Keep your "featured ingredients" D.R.Y and use it everywhere in your game development!*** After a few game development cycles, I can refer back to all those game prototypes that were previously created. As you can quickly see — illustrated in the second listing above — there's only four (4) ***visual Game Mechanisms*** items to create (buttons, objects, menus and text), an animation effect (visual manipulation), and three (3) undefined items which appear to be some sort of "functions or process".

### How will you encode it?

Now that you have an idea on what features you'll encode first, What variable or functions names should you use in your program? How will we design our game software? What is our game architecture?

***Exercise:*** Sneak ahead to ***Chapter 2: "4-Step Method"***

## 1.5 Game Design Architecture

### "Oh! Oh!"

The answer comes from a phrase senior software engineers call **"high level architecture" design.**[167] Using ***Object-Oriented Analysis Design (OOAD)***[168] in your game development process involves breaking your game's idea into parts (i.e., data structures); and, then describing how those individual parts interact with each other. For example, dissecting your game description into categories like "things" (aka "objects"), rules and metrics (aka, "logic"), ***"human computer interaction"***[169] (HCI), "user data" information, and "camera viewports" (i.e., what the player see during the game progress) — then think about how you might write those items as JavaScript code, such as object types, functions, or variables. Here's another example:

- Game Menus and Scenes (plural noun)
- Music Tiles (plural noun)
- Music sound files (plural noun)

[167] https://en.wikipedia.org/wiki/Architectural_pattern
[168] https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design
[169] https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction

- Tool-tips text (a noun)
- User Interface button(s) for navigation (plural noun)
- Splash Screen (a noun)
- Background theme music (a noun)
- Heads Up Display (a noun)
  - Scores display (a noun)
  - timer display (a noun)

In a very primitive way, we have just created a game using the ***Object-Oriented Analysis Design (OOAD)***[170] method. From a different "Elevator Speech" and game description I wrote, I collected all the **nouns** from the game's description. This is not yet executable JavaScript; it's called "pseudo-code" and we have a lot more to do. "OOAD" should include some **adjectives** (properties of those **noun-thingies**[171] in our game), **adverbs** and **action verbs** (how, when, and what those things do respectively). For example,

> When (an "adverb") I click (a "verb") a game tile (a "noun"), it should play (a "verb") a music file (a "noun").

It becomes a trivial process to create games using just simple grammar.

Another side-benefit of using "OOAD" is an opportunity to test the game's "Enjoyable Factor" (aka, is the game fun?). Collect all of the player's **action verbs** (such as shoot, command, run, purchase, build, and look) and envision how a player might perform each one. Then, for each of those "verbs", ask yourself if that game action is fun. Again ask yourself, if the target market — identifying our target audience is coming later — would find it fun. Be objective! If those player "actions" are not enjoyable or fun, substitute another action for the player to do that would be fun; otherwise, drop the action-verb entirely.

***Exercise:*** Download my interpretation of ***"Comparison Chart of Fun to Human Emotions"***[172] and a white-paper entitled, ***"MDA: A Formal Approach to Game Design and Game Research"***[173]. This next reference was developed from a small (60 total) population sample while attending a game convention (a skewed population sampling). Regardless, it is still an interesting thesis about the ***"4 Keys to Fun"***[174]

[170] https://nunoalexandre.com/2017/02/12/this-is-not-object-oriented
[171] https://www.urbandictionary.com/define.php?term=thingie
[172] https://makingbrowsergames.com/design/_p3-16HumanMotivations.pdf
[173] https://makingbrowsergames.com/design/MDA.pdf
[174] http://www.xeodesign.com/the-4-keys-to-fun/

Sample Code derived from the “OOAD” Breakout Game Description Example:

- **Objects:**
  - var Brick
  - var Paddle
  - var Balls as new Array()
- **Scenes:**
  - Splash/Start
  - Play Game
  - Ending (Win and/or Lose)
  - Credits
- **Logic:**
  - Brick (.isHit())
  - Paddle (.move())
  - Ball (.move(); .droppedOut())
  - Ball-brick collision (function, use bounding box)
  - Build Brick Grid (function container)
  - Paddle-ball collision (function, use bounding box)
  - Paddle-ball angling (function, invert angle)
  - Reset Game (function)
- **User Interaction & Heads-Up Display:**
  - Keyboard-paddle movement (keyPressed)
  - Buttons for scene changes (mouseClicked)
  - Text boxes (Score, Remaining Attempts)
- **Game data**
  - Ball Dropped Out (Remaining Attempts -1)
  - Ball Hits Bricks (Score + 1)

***Note:*** I could have used this Chart “sorted by Items” instead of this listing above.

## “Top-down”

***Top-down design*** (aka Step-wise refinement) is **another technique**[175] — **among many; (click here to see the 10 most commonly used)**[176] — that professional programmers use when they have to go beyond simply identifying items — as we

[175] https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns
[176] https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

did in the examples above. ***Top-down design*** helps define tasks inside of tasks. Small-scale problems are usually composed of tasks all at the same level. This means that there are few opportunities for a task to contain several other tasks (i.e., inheritance) of which turn into JavaScript objects with method functions.

In a ***Top-down design,*** the solution to the problem is found by breaking down the problem into solvable sub-problems. However, these sub-problems are not smaller versions of the large problem. These sub-problems have these following characteristics:

- Each sub-problem must be solvable by a process or set of rules to be followed in calculations or other problem-solving operations.
- Each sub-problem should be independent of any other sub-problems.
- Solving a sub-problem should be significantly less complex than the main parent problem.
- Solving the sub-problems should lead to solving the overall main problem by jointly composing the solutions for all the associated sub-problems.
- Performing step-wise refinement will lead to software functions and "classes" nested in related "modules" when we begin writing our game's source code. (**NOTE:** more on JavaScript Modules in the Coding Appendix.)

## "Bottom-up"

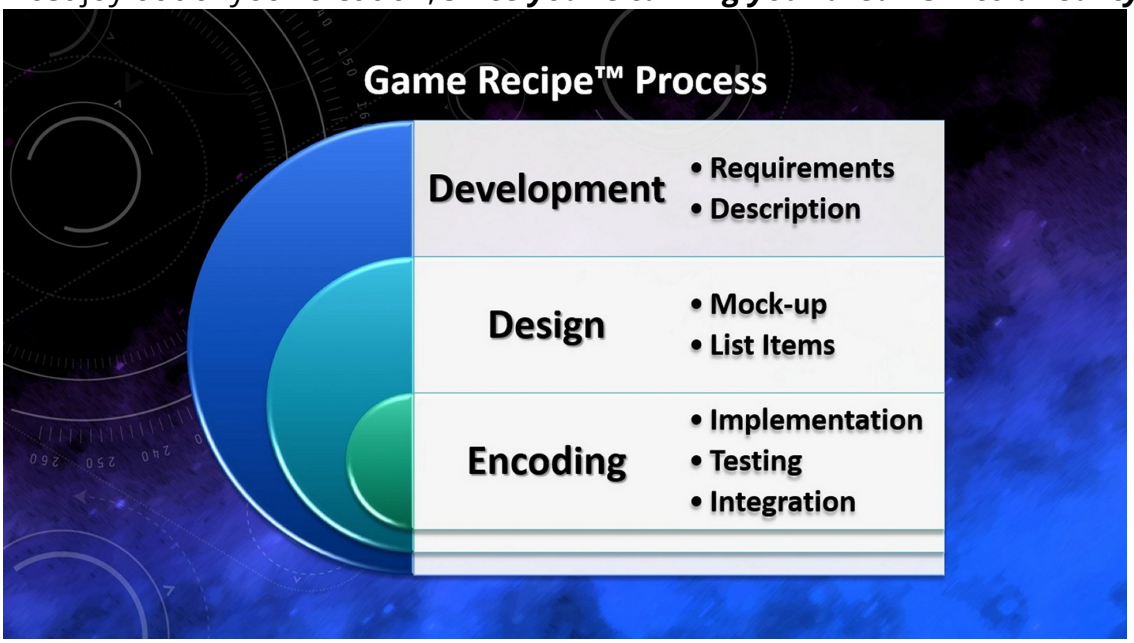
***Bottom-up design*** occurs when you determine what programming routines are **available to you already from the Phaser JavaScript Framework,** and you'll selectively use them to "build up" your program instead of creating that code yourself. Since we are focusing on the **Phaser JavaScript Framework**, you'll discover 90% of the work has already been done for you in your game's construction. *All we need to do is **simply find** those various pieces of code that our game requires from the Phaser API.*

## "Oh! Oh!" vs. Top-Down vs. Bottom-Up vs. OLOO

You might be wondering which design philosophy is the best one for you to adopt and use: "OOAD", top-down, bottom-up or "Objects Linking to Other Objects". In reality, neither are better than the others. These processes are complementary. When you have to design software from scratch or add to existing software, you are likely to use any of those processes to help you achieve your best design.

## 1.6 Game Project Summarized:

By now, you hopefully understand the ***Game Design System™*** process for creating a game project ***recipe.*** The most important thing is to ***"ACTUALLY START MAKING THE DAMN GAME",*** because that is where you'll learn the most, and also where you'll get the most joy out of your creation, ***since you're turning your dreams into a reality.***




**Game Recipe™ Process**

### Concept Development:

1. Copy your a fresh/new 'file structure' into a new/separate project directory/folder. A basic `index.html` file should be there already; just update the ***"<head>"*** metadata for search engine optimization (SEO).
2. Describe what you're making in an **elevator speech** (aka, "Game Description" text file).

### Design:

3. Draft a ***"mock-up" sketch*** of the game phases and content for each game phase.
4. ***List the items,*** their priorities, and **"Catalog their classification".** (with an Excel spread sheet?)
5. Sort the items by either: 1) priorities, or **2) "Catalog classifications"** — this is my favorite method and easiest for me to encode content.

## Production Encoding:

***Note:*** following the ***Principles of Software Engineering***[177]

6. Use ( aka ***"implement!"*** ) ***"software architecture design"***[178] to break down (i.e., ***Deconstruct!*** ) the various mechanisms and components into their logical elements. Use a combination of "OOAD", top-down, OLOO, and/or "bottom-up" design methods.
7. Find those ***previously tested*** prototype items you've already created in other game products ***— such as your "index.html" page —*** stored in the ***Game Recipes™ tool***[179]. Use those game prototypes. If they don't exist then go to step #8.
8. Create and ***integrate*** those missing game prototypes — using the ***4-step method (found in Chapter 2)*** Classify them with other similar components and include them into your automation tools. It's worth mentioning again ***— this is the secret sauce in our Game Recipes™! Keep your "featured ingredients" D.R.Y and use it everywhere in your game development!***

The core "Game Mechanics" (GM) are written in "pure" JavaScript functions (or ***ES6 "classes"***[180]) which support the game's logic, data, and rules. These components will "chat" among themselves ***with whatever "JS Gaming Framework"*** you've selected. This provides flexibility to exchange entirely different "Gaming Frameworks Mechanisms" (GFM) — the "front-end visual elements" — without changing the artwork theme nor game's mechanics. The "mechanisms component" come from your selected "JS Gaming Framework" and will handle the gamers' displays and User Interfaces (UI). Much of the code written in the "front-end" Phaser Gaming Framework is ***event-based.*** You'll define some behaviors for the gamer's input, and then attach those to "a triggering event" such as a button click or a press key from a keyboard. The "artwork component" will supply the graphics and multimedia for and into the "mechanism prototypes". The "game mechanics and rules" I include in the "main.js" (aka "game.js") to merge and minimize the number of files downloaded.

Download this FREE 400+ page ebook: ***"Game Development for Human Beings" from GameDev Academy.***[181]

[177] https://makingbrowsergames.com/design/_PrinciplesofSoftwareEngineering.pdf
[178] https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/
[179] https://makingbrowsergames.com/gameDesigner/
[180] http://know.cujojs.com/blog/oop-is-not-my-hammer
[181] https://gamedevacademy.org/free-ebook-game-development-for-human-beings/?a=47

## 1.7 Summary

Let's review and take inventory of what we've covered so far. In Part IV, we will revisit these steps again as we walk through the creation of several different games.

- We have read pages of supplement content from 52 external sources.
- Downloaded half-a-dozen additional Bonus Content files from reference links.
- We have set-up a workstation environment.
- We discovered helpful debug sites and data sources.
- Chose and set-up an operational web-server.
- Created a file structure to become a consistent foundation for all on-going projects.
- We learned what ***is*** and ***is NOT*** game development.
- Installed several development and supporting tools.
- **Learned where to find $1,000+ of free software for game development.**
- Reviewed tools to migrate the `HTML5` game onto various mobile platforms.
- Constructed a game "front-door" with SEO.
- Understand the use of JavaScript Modules.
- Learn the ***Game Design System™*** and how to make a game "recipe".
- Clarified what Copyright means from the US Copyright Office.
- Read several software architecture design concepts.
- Migrated all current game prototype mechanisms into a separate namespace.

## 1.8 Chapter References:

- ***Programming like a Pro***[182] Chapter 8, by Charles R. Hardnett
- ***Google Analytics***[183]
- ***Getting Started Making Video Games***[184] by John Horton.
- Tools for Web Developers — Setting Up Your Dev Environment[185]
- How to use browserify[186] to build modular applications. Free handbook.
- Google Search for Text Editor for Source Code Development[187]
- "Using ECMAScript 6 today"[188] gives an overview of ECMAScript 6 and explains how to compile it to ECMAScript 5. If you are interested in the latter, start reading in Sect. 2. One intriguing minimal solution is the ES6 Module Transpiler which only adds ES6 module syntax to ES5 and compiles it to either AMD or CommonJS.
- Embedding ES6 modules in HTML: The code inside `<script>` elements does not support module syntax, because the element's synchronous nature is incompatible with the asynchronicity of modules. Instead, you need to use the new <module> element. The blog post "ECMAScript 6 modules in future browsers"[189] explains how `<module>` works. It has several significant advantages over `<script>` and can be poly-filled in its alternative version `<script type="module">`.
- CommonJS vs. ES6: "JavaScript Modules" by Yehuda Katz[190] is a quick intro to ECMAScript 6 modules available on Github[191]. Especially interesting is another page[192] where CommonJS modules are shown side by side with their ECMAScript 6 versions.
- Understanding ES6 Modules[193]
- How the Web works[194] Mozilla Developer's Network (MDN) provides this Learning Area designed to answer common questions that come up.

[182] http://amzn.to/2b8gvUr
[183] https://developers.google.com/analytics/devguides/collection/
[184] https://makingbrowsergames.com/design/_p3-GettingStartedMakingVideoGames.pdf
[185] https://developers.google.com/web/tools/setup/
[186] https://github.com/substack/browserify-handbook
[187] https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=text%20editor%20for%20source%20code
[188] http://2ality.com/2014/08/es6-today.html
[189] http://2ality.com/2013/11/es6-modules-browsers.html
[190] http://jsmodules.io/
[191] https://github.com/wycats/jsmodules
[192] http://jsmodules.io/cjs.html
[193] https://www.sitepoint.com/understanding-es6-modules/
[194] https://developer.mozilla.org/en-US/docs/Learn/Common_questions#How_the_Web_works

# 2. Building a Game Prototype

***"What's a game prototype", you say?***

Game classification is diverse. So, I'd like to agree on some standard definitions as we "cook up" our game using the ***Game Design system™*** and its ***Game Recipes™ tools.***

- ***game prototypes —*** Quoted from ***"Phaser III Game Design Workbook",*** (page 69), "By combining all of our game mechanisms, with a set of game mechanics and its rules systems — as ***non-invasive aspects in our gaming product***[1] — along with an ***artwork theme***[2], we're able to create multiple game products quickly. It simply becomes a matter of exchanging any of those ***"3 cross-cut" components***[3] into a new innovative mixture for a new game product. ***This is the secret in concocting a new game***[4] ***every month or even every week!*** For example, swopping a "Guitar Hero" artwork theme with a "Plants & Zombies". The new game uses the same "Game Mechanics" and "Game Display Mechanisms" but with a "fresh organic-garden" look and feel! From Page 117, "... let me write that code once and reuse it for similar items in other games as a component prototype — this is the secret sauce in our Game Recipes™! Keep your "featured ingredients" D.R.Y and use it everywhere in your game development!"
- ***mechanics —*** Quoted from ***"Phaser III Game Design Workbook",*** (page 68), "From a game programming perspective, basic Game-Play can be deconstructed — revealing tactical components inside a game's overall mechanics and rules. For example, a fighting game deconstructs into various tactics such as attacks (or punches, throws, and kicks), defensive moves, and dodges. These tactics are assigned to game prototypes and mechanisms — input keys, mouse clicks, and mobile screen interactions ...." and
- ***mechanisms —*** Quoted from ***"Phaser III Game Design Workbook",*** (page 85), "The gameboard grid defines the ***Game Mechanics (GM)*** movement rules; ***how the grid is drawn*** is the ***Game Framework Mechanism (GFM).*** Players will send their decisions from their device's inputs — keyboard, mouse, game-pad, etc. — using their browser and the widget mechanisms we designate as drop-down menus, buttons, and "sliders". Mechanisms are the "displays elements" of the ***Game Framework Mechanism (GFM)."***

[1] http://know.cujojs.com/blog/oop-is-not-my-hammer
[2] https://craftpix.net/categorys/2d-game-kits/?affiliate=112238
[3] http://know.cujojs.com/tutorials/aop/intro-to-aspect-oriented-programming
[4] https://www.vocabulary.com/dictionary/concoct

***So then, "What's a game prototype", you asked?***
***Answer:*** It is an operational gaming foundation that can:

1. accept inputs (***Gaming Framework Mechanism (GFM));***
2. move game elements and components (***Gaming Framework Mechanism (GFM));***
3. the transition between game phases, and (***Game Mechanics*** and display elements as visual components from the ***Gaming Framework Mechanism (GFM).***)
4. reacts to internal game object collisions with visual feedback (both ***Gaming Framework Mechanism (GFM)).***

***What are the benefits of creating a game prototype first?***

***Answer:*** See the latest comments from various gaming experts ***here***[5] and ***here***[6]; and other software engineers' opinions about ***prototyping in general — here***[7]. In short, you're trying to find if your game is "fun"!

My game design concept is clearly echoed in both Apple's Game-Play Kit and ***Play Canvas***[8] as ***"Entities and Components".***[9] They plainly state, "The Entity-Component design pattern is an architecture that ***favors composition over inheritance.*** To illustrate the difference between ***inheritance-based*** and ***composition-based*** architectures, consider how you might design an example "tower defense" style game, with the following features …". It's a wonderful feeling to discover **after 20 years** that other *prominent game developers* are thinking along the same patterns of game prototype development.

[5] https://www.quora.com/What-is-the-benefit-from-creating-the-prototype-of-a-game-first#
[6] https://www.quora.com/Do-game-developers-create-prototypes-first-before-programming-the-actual-game
[7] https://www.sqa.org.uk/e-learning/IMAuthoring01CD/page_06.htm
[8] https://playcanvas.com/
[9] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html#//apple_ref/doc/uid/TP40015172-CH6-SW1

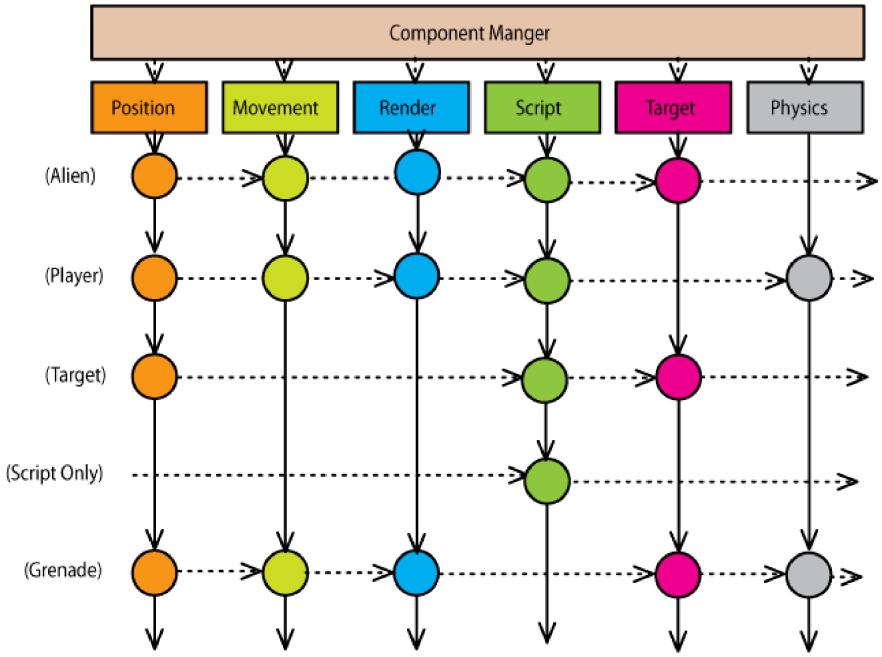



Figure 2 Object composition using components, viewed as a grid.

**Components and "Objects Linking to Other Objects" (OLOO)**

***Exercise:*** Read ***"Apple's Game-Play Kit: Designing with Entities and Components".***[10] and learn why Apple claims that ***"Inheritance-Based Architecture Hinders Game Design Evolution"*** and their illustrations on how ***"Composition-Based Architecture Makes Evolving Game Design Easy".***

Perhaps the most popular opinion — coming from one of my game development heroes is:

> ## How to Succeed at Making One Game a Month
>
> ***Reach the Finish Line More Often***
>
> ... "90% of game projects never see the light of day. My own personal experience confirms this. I've been making games for over twenty years, and of all the games I started - filled with enthusiasm, a detailed plan, and infinite brainstorms worth of ideas - only a small percentage were ever released. This caused me years of heartache. I was a good coder, I could produce ***acceptable artwork,***[a] I had enough good ideas to feel confident about my plans, and yet that wonderful state where the game is ready for the public was an elusive target. ...
> ...

[10] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html#//apple_ref/doc/uid/TP40015172-CH6-SW1

***#5. Make a No-Art Early Playable***
The next major handy tip for this challenge is to make a playable ***game in the first day.*** No title screen, only one level, and just the primary gameplay mechanic.

It won't be great, it won't be finished, and it certainly won't look that great or be all that fun. That said, this step is your best weapon. Challenge yourself ***to create a codebase that compiles and runs in the first few hours. Make it so that you can accept inputs, move around, animate something, and trigger some sounds. This prototype, lousy a game as it may be, is going to be your best friend.***

***The sooner you can have a working early playable prototype, the more likely you are to succeed.*** It will be your first "save point" - a resting plateau on the way to the top of the mountain that you can fall back on. ***It represents a vision of the working game.*** From here on you will be able to polish your game for as long as you like with the knowledge that you have something in hand that "works".

***No-art prototypes also have one other major advantage:*** in previous games, I would make beautiful mockups in Photoshop and gather hundreds of lovely looking sprites in preparation for the game. After development was complete, ***the vast majority of the art had to be replaced, resized, or thrown out. I've wasted thousands of hours making game-ready artwork[b] before coding; these days I know that the tech specs and evolving game-play mechanics[c] will mean that much of what you make at the start won't make it into the finished game.***"
Read more ***here.***[d]

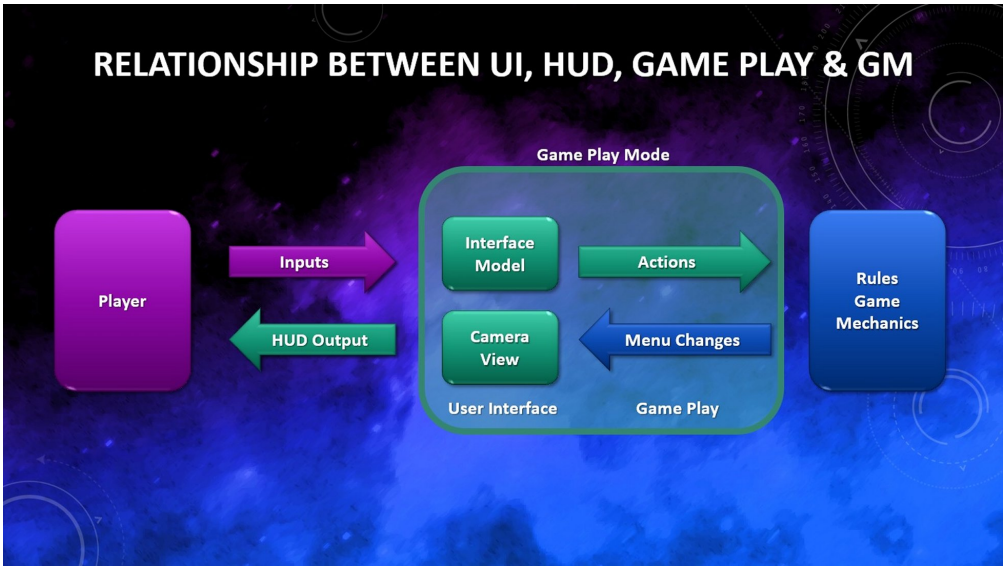
[a] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[b] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[c] http://www.lostgarden.com/2006/10/what-are-game-mechanics.html
[d] https://gamedevelopment.tutsplus.com/articles/1gam-how-to-succeed-at-making-one-game-a-month--gamedev-3695

***Hint: If you're tired of starting over, stop giving up***[11]

## 2.1 Creating Prototype Mechanisms — 4-Step method

The last step (i.e., #8) in our ***Game Recipes™*** was to create missing game components and prototypes. We'll follow these next 4-steps, from here on, whenever we need to generate ***new game prototypes and component mechanisms.*** It'll become our ***regimen***[12]:

[11] https://www.youtube.com/watch?v=yzeJ77RNcjs
[12] https://www.merriam-webster.com/dictionary/regimen




**Game Design System™ - Single Player**

***1st RULE: Always be consistent in placement, programming paradigm[13], JS coding style[14], and naming schemes[15]. It might be worth reviewing what others are doing[16] with their JS Styling[17] and how they program JavaScript.[18]***

***Exercise:*** Learn ***how to out-perform senior programming developers***[19] in 3 months time?! ... use ***game programming patterns***[20].

## Step 0) Preparation and Research

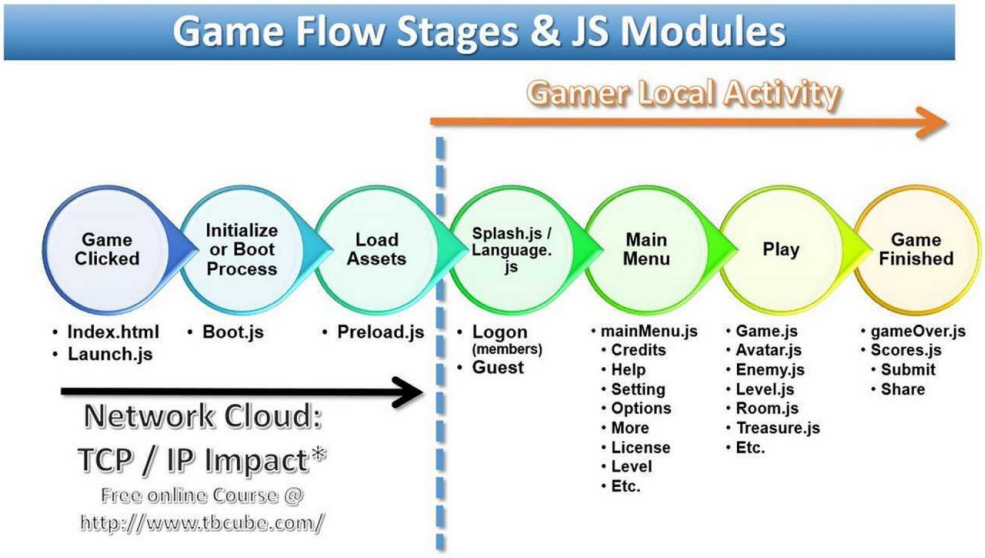
This is the fun part when developing a new game — so, enjoy! ***(Boy Scouts: Earn your Merit Badge![21])***

- Research and play a similar gaming genre, currently on the market, that match your ideas and appeal to similar target audiences. ***Record which sites host those games and investigate their submission policies.***

[13] https://github.com/getify/You-Dont-Know-JS/blob/1st-ed/this%20%26%20object%20prototypes/ch6.md
[14] https://codeburst.io/5-javascript-style-guides-including-airbnb-github-google-88cbc6b2b7aa
[15] https://en.wikipedia.org/wiki/Computer_network_naming_scheme
[16] https://standardjs.com/
[17] https://hackernoon.com/what-javascript-code-style-is-the-most-popular-5a3f5bec1f6f
[18] https://github.com/getify/You-Dont-Know-JS
[19] https://medium.com/better-programming/how-i-outperformed-more-experienced-developers-as-a-junior-developer-and-how-you-can-too-19bc6206fa68
[20] https://gameprogrammingpatterns.com/
[21] https://makingbrowsergames.com/book/Game_DesignMeritBadge.pdf

- Record their "Search Engine Optimizations" (SEO) — metadata descriptions, keywords, where & who the game is hosted, etc.
- ***Create your game project "front-end index.html" file(s).***
- Follow the ***Game Project Steps #1 through #8.*** or use the ***Game Recipe™ Tool***[22].

## Step 1) Generate Game Phases (as needed).



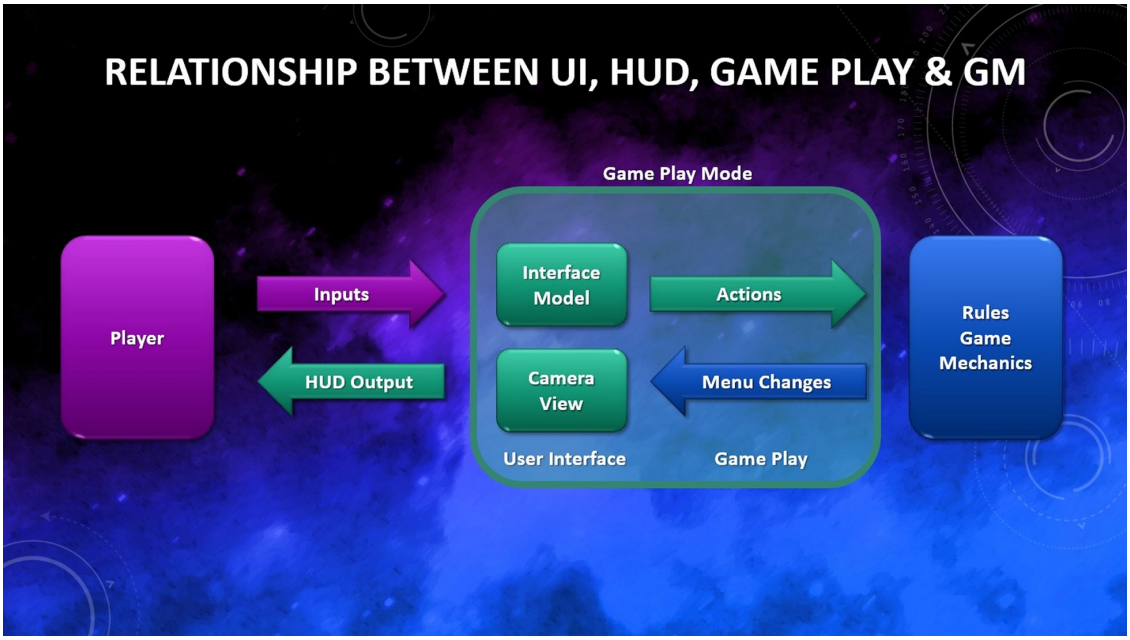

**Typical Game Phases from "Phaser III Game Design Workbook"**

Once these are created, ***they should be "relatively" D.R.Y. (Don't Repeat Yourself)***

## Step 2) Generate code for triggering events.

Much of the code written in the "front-end" Phaser Gaming Framework is ***event-based.*** You'll define some behaviors for the gamer's input, and then attach those to "a triggering event" such as a button click or a press key from a keyboard. These — listeners, observers, sensors, sentinels, web workers, etc. — will be placed in the "Play" Game Phase since most of these relate to the visual "Game Framework Mechanisms" (GFM) and display Components. See a ***flow chart***[23] when and what scenes update, and review the ***various scene event states***[24].

[22] https://makingbrowsergames.com/gameDesigner/
[23] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scene/#flow-chart
[24] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scene/#events




**Relationship between UI, HUD, Game Mechanisms, & Mechanics (Single Player)**

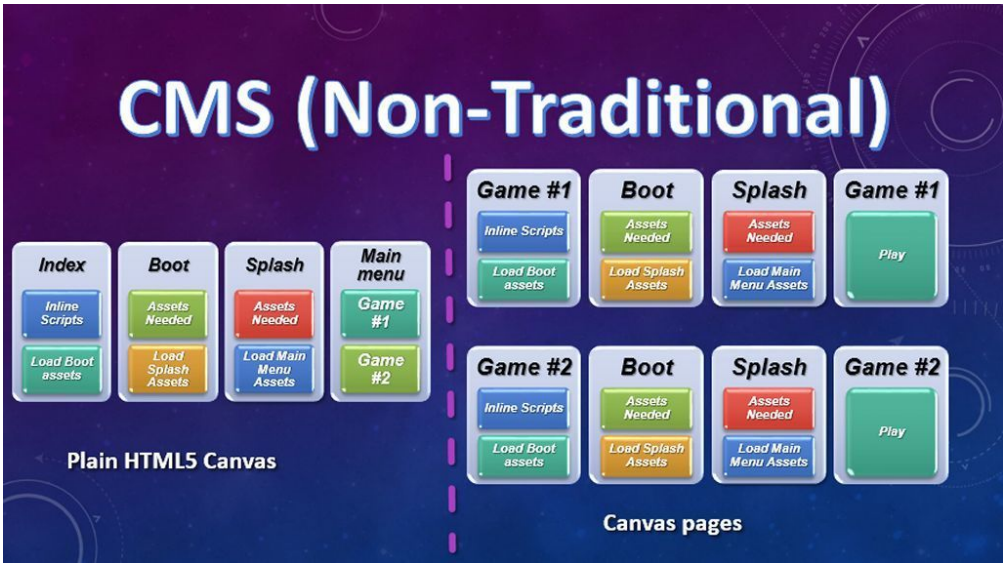
**Example of Phaser 3 triggering events**

```
    // an event listener as the 'handler' function.
    // The 'this' argument is the context.
    this.events.on('ohICU', handler, this);

    // Native Scene's own 'EventEmitter' dispatchs our events
    this.events.emit('ohICU');
```

## Step 3) Generate transition

This code is also placed inside the “Play” Game Phase. Its primary purpose helps transition “into and out from” the new game phase and their internal Phaser III Scenes. Once these are created, ***they should be “relatively” D.R.Y. (Don’t Repeat Yourself)***




**Transitions using separate pages instead of a single Phaser canvas**

***Example*** of a ***"Non-traditional Phaser III transitions"***[25] using individual web pages. WHY? It's due to the "Phaser.cache" and artwork created for this game.

## Step 4) Create your Game's Core & auxiliary functions


**Cloud-based Game Design - "Just In Time" Management**

Building your core display mechanisms into components; this means that you request "what" your game needs from "Infrastructure-as-a-Service", "Platforms-as-a-Service", "Content-as-a-Service", and "Data-as-a-Service". It's the same concept as using "npm and inserting dependencies" on steroids!

[25] https://makingbrowsergames.com/starterkits/dressup/v3.x.x/index-cms.html

***Exercise:*** Download this ***"template"***[26] as a new project reference. Open and watch the Developer's Console while running this template or simply click and watch for these ***4-Step numbers*** annotated inside the source code.:

***https://makingbrowsergames.com/p3gp-book/standalone/***

***Warning: Avoid "Anti-patterns"***[27] when developing your game source code when integrating others' supporting functions, document any ***encroaching***[28] "Anti-patterns" you find, and share your findings in the Phaser forums. Bookmark the following ***FREE*** online book: ***Essential JS Design Patterns by Addy Osmani***[29]. **It shows what to look for and how to resolve "Anti-patterns" sneaking into the Phaser Libraries.**

Once we have completed these steps for our new game phase, we must ***bring them alive.***[30] To do this, we load the JavaScript module either through an inline ***"<script>"*** tag in our ***"index.html"*** file; or by importing it into our ES6 ***"index.js"*** file (if you are using an ES6 structure, we'll learn how to automate this process later).

As we build our Game Prototypes it is helpful to use the browser's console and developer tools. The browser console in the "Developer Tools", tells us a lot about our game's performance. If you've never used the "Dev Tools" in the console, you might take a side trip to ***"Mastering The Developer Tools Console"***[31] But first, we need a web page to hold our game ...

### Deeper Dive: Writing D.R.Y. JS code

***Wait!*** How do I write **D.R.Y.** JavaScript source code? Ok, let's take a side-trip to ***Clean Code concepts adapted for JavaScript***[a] — (quote) "Software engineering principles, from Robert C. Martin's book ***Clean Code (Amazon),***[b] adapted for JavaScript. This is not a style guide. It's a guide to producing ***readable, reusable, and refactor-able***[c] software in JavaScript."

[a] https://github.com/ryanmcdermott/clean-code-javascript
[b] https://amzn.to/2WK2pAT
[c] https://github.com/ryanmcdermott/3rs-of-software-architecture

[26] https://makingbrowsergames.com/p3gp-book/_v3.x.x-p3gp-book.zip
[27] https://addyosmani.com/resources/essentialjsdesignpatterns/book/#antipatterns
[28] https://www.dictionary.com/browse/encroaching
[29] https://addyosmani.com/resources/essentialjsdesignpatterns/book/
[30] https://idioms.thefreedictionary.com/bring+it+alive
[31] https://blog.teamtreehouse.com/mastering-developer-tools-console

## 2.2 Using "Box" Graphics

Since we are making a working "game prototype", let's keep our artwork as generic as possible, and save the efforts of art selection and consistent styling until a later step. We will save ourselves those ***"thousands of hours";*** and, for now, just simply set up "block-style graphics" and assign basic colors to represent our gaming components. Phaser v3.13.x offered a feature that simplifies building these "block-style graphics". The reason we are doing such simple "placeholders" is to ***learn if our game idea is viable — if it's fun?!***

We will begin with a simple top-down (aka "Bird's Eye" view) game with an avatar character, several walls, text narrative, "heads-up display" (aka HUD), and several opponents. Doing so, we have a functional game prototype to use for other game ideas. ***By swopping out these simple blocks for a variety of artwork themes, this allows us to create 100s of games from similar game mechanics[32] using different theme settings.***

We'll explore different game perspectives, mechanics, themes, and modes later in this book and adjust these prototype mechanisms accordingly. For now, review how to use ***"Isometric View in Phaser 3"***[33]

***Exercise:*** Study the v3.13.0 "shapes features" by ***reading this DevLog 128***[34]

[32] http://www.lostgarden.com/2006/10/what-are-game-mechanics.html
[33] https://medium.com/@Tnodes/creating-an-isometric-view-in-phaser-3-fada95927835
[34] https://phaser.io/phaser3/devlog/128

**Sample 2.1: Prototyping Graphics**

```
    // available since Phaser v3.13+.
    // 2D: this.add.rectangle(x, y, width, height, color)
    // AND even ...
    // 2.5 & 3D as the new Phaser III "ISOBOX"
    // new IsoBox(scene, x, y, size, height, fillTop, fillLeft, fillRight])

    // direct method using available internal Phaser geometry.
    var shape = this.add.rectangle(400, 250, 32, 32, 0x00FF00);

    //
    // ====================================
    //OR use our prototype method ...
    var player1 = this.add.sprite(
            100, 400,  // display x and y coordinates.

            box(
               {who: this,
                whereX: 100,
                whereY: 350,
                length:100,
                width:100,
                color: 0xFF0000,
                border: 0xFFFFFF}
            ) // call out to factory function

        );  //End sprite
    ); //new shiny graphics-box avatar!

    //
    // ====================================
    //create a "box" on the HTML5 canvas.
    function box(opt) {
        //syntax: new Rectangle( [x], [y], [width], [height])
        //var bxImg = new Phaser.Geom.Rectangle(
        //      opt.whereX,
        //      opt.whereY,
        //      opt.width,
        //      opt.length);

        // OR use rectangle:
        var bxImg = opt.who.add.rectangle(
```

```
                opt.whereX,
                opt.whereY,
                opt.width,
                opt.length);
        // decorate our shiny new "box"
        var bxColor = opt.who.add.graphics(
                {fillStyle: {color: opt.color},
                 lineStyle: {color: opt.border} });

        bxColor.fillRectShape(bxImg); //fill box with color
        bxColor.strokeRectShape(bxImg); //draws a border around it.
        return bxImg;
    };
```

***Exercise:*** Download the example above:

**https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js**

This new ***"shape"***[35] feature from v3.13.0 takes on the characteristics of **a normal "game Object"** without having to "bake" a texture as you would have to do with a graphics object.

***What time did all that take? a couple of seconds?*** This way — using generic boxes or the ***new Phaser v3.14+ "rectangle shape"***[36] — we can deploy them to represent any game elements as well as player character(s), boundaries, walls, doors, treasures, and opponent(s) entities.

By swopping out these simple blocks for ***a variety of artwork themes***[37] settings, it gives us the opportunity to create 100s of games along similar ***game mechanics***[38].

***Hint:*** We'll explore different game perspectives, mechanics, themes, and modes in later chapters and adjust these prototype mechanisms accordingly. Here's an example of a ***side-view gaming prototype by another famous Phaser v2.x.x author — Thomas Palef.***[39]

[35] https://github.com/photonstorm/phaser/blob/v3.14.0/src/gameobjects/shape/Shape.js
[36] http://labs.phaser.io/edit.html?src=src/game%20objects/shapes/rectangle%20with%20arcade%20physics.js
[37] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[38] http://www.lostgarden.com/2006/10/what-are-game-mechanics.html
[39] http://www.lessmilk.com/tutorial/2d-platformer-phaser

## 2.3 Game Practicum: Box Prototyping

Let's apply the knowledge we're learning is a simple 2D maze.

### Phaser III Code Review

***Play Phaser III demonstration here***[40].

***Exercise:*** Download and use the following files with all the notes in this Code Review. Open the console and watch the internal operations.

- ***p3_2DRooms-mainJS.pdf***[41] — 4 pages.
- ***p3_2DRoomsDemoJS.pdf***[42] — 8 pages.

#### MainJS - p3_2DRooms-mainJS.pdf

- Lines 1 to 39 are general administration and acknowledgments.
- Lines 40 to 49 create the 2D array data structure with hexadecimal flags, a graphics object, and game namespace.
- Lines 57 to 154 is the Game Mechanics (GM) Component. I placed the function that handles character movement in the GM component. It could just as easily appeared in the ***"Game Framework Mechanisms" (GFM)*** Component for better "Separation of Concerns". See the illustration below.
- Lines 170 to 201 is the Phaser3 configuration object.
- Lines 203 to 207 launches the game using browser document instead of ***"`window.onload`"***.

#### DemoJS - p3_2DRoomsDemoJS.pdf

- Lines 1 to 53 are general administration, data structures, and acknowledgments.
- Lines 54 to 59 create "Game Framework Mechanism" variables, and a graphics objects.

[40] https://makingbrowsergames.com/starterkits/adventure/_p3-2DRooms/
[41] https://makingbrowsergames.com/starterkits/adventure/p3_2DRooms-mainJS.pdf
[42] https://makingbrowsergames.com/starterkits/adventure/p3_2DRoomsDemoJS.pdf

#### Lines 61 to 292

Lines 61 to 292 is the Game Mechanisms (GM) and Phaser JS Component. I don't use any artwork nor images. I create the room's wall using a "pseudo" graphics method in my game prototype.

**Canvas Graphics - Box function**

```
279     //
280     //=====================================================
281     //create a box Image (pseudo graphics) for the HTML5 canvas.
282     box: function (options) {
283         //var bxImg = this.add.bitmapData(options.length,options.width);
284         var bxImg = game.add.rectangle(options.length, options.width);
285         bxImg.ctx.beginPath();
286         bxImg.ctx.rect(0, 0, options.length, options.width);
287         bxImg.ctx.fillStyle = options.color;
288         bxImg.ctx.fill();
289         return bxImg;
290     }
```

#### Lines 293 to 298

- Lines 293 to 298 handles the avatar collision with walls.

**Avatar bumps into Walls**

```
293     //
294     // ================================================
295     function bumpWall() {
296         player.body.velocity.x = 0;
297         player.body.velocity.y = 0;
298     };
299     //
300     // ================================================
```

#### Lines 301 to 315

- Lines 301 to 315 handles the avatar collision with doorways.

**Avatar bumps into Doors**

```
299     //
300     // ===============================================
301     function bumpDoor(player,door) {
302         // if a player moves into a doorway.
303         player.body.velocity.x = 0;
304         player.body.velocity.y = 0;
305         console.log("LN 305: Bumped into Door "+door.name);
306         // which doorway? Is it visible???
307         // BUG: going through invisible doors? Why?
308         // FIX: if visible, then allow passage; otherwise, stop
309         if(door.visible){
310             newRoom(door);
311             resetRoom();
312         }
313         player.body.velocity.x = 0;
314         player.body.velocity.y = 0;
315     };
316     //
317     // ===============================================
```

**Lines 318 to 376**

- Lines 318 to 376 handles moving the avatar into the new room via a specific doorway.

**Determine new Room Entered**

```
316     // http://www.html5gamedevs.com/topic/5304-how-to-restartreload-a-state/
317     // ===============================================
318     //Main Door click handler
319     function newRoom(door) {
320         // 2 Options:
321         // - reset this phase with new room characteristics OR
322         // - have a "repaint" function to adjust the entered room.
323         // Option 1: this.scene.restart();
324         // Option 2: separation of concerns - new function
325         Rooms2D.LastRoom = Rooms2D.CrntRoom;
326         player.setPosition(64, 64);
327         var LastDoor = door.name;
```

```
328         console.log('Last Door Used: ' + door.name);
329 
330         switch (LastDoor) {
331         case "North":
332             //Rooms2D.CrntRoom -= 4; // or GRID_ROWS or MT.length
333             Rooms2D.CrntRoom = Rooms2D.LastRoom - 4;
334             Rooms2D.CrntRoomY -= 1;
335             //Leave via North; enter new room from South-side
336             Rooms2D.pPosX = config.width / 3;
337             Rooms2D.pPosY = 320;
338             break;
339         case "East":
340             Rooms2D.CrntRoom += 1;
341             Rooms2D.CrntRoomX += 1;
342             //Leave via East; Enters new room from the west-side
343             Rooms2D.pPosX = 50;
344             Rooms2D.pPosY = config.height / 2;
345 
346             break;
347         case "South":
348             Rooms2D.CrntRoom += 4; // or GRID_ROWS or MT.length
349             Rooms2D.CrntRoomY += 1;
350             //Leave via South; enter new room from North-side
351             Rooms2D.pPosX = config.width / 3;
352             Rooms2D.pPosY = 50;
353 
354             break;
355         case "West":
356             Rooms2D.CrntRoom -= 1;
357             Rooms2D.CrntRoomX -= 1;
358             //Leave via West; enters new room from east-side
359             Rooms2D.pPosX = 340;
360             Rooms2D.pPosY = config.height / 2;
361 
362             break;
363         }
364 
365         player.setPosition(Rooms2D.pPosX, Rooms2D.pPosY);
366         console.log("New Room #: " + Rooms2D.CrntRoom + "; Door Clicked: " + door.na\
367 me);
368         /**
369         //sfx camera fadein/out
370         this.cameras.main.once('camerafadeincomplete', function (camera) {
```

```
371         camera.fadeOut(1000);
372         });
373         this.cameras.main.fadeIn(1000);
374          */
375         resetRoom();
376     };
377     //
378     // ================================================
```

**Lines 379 to 431**

- Lines 379 to 431 handles the new room “set-up”.

**Reset Room properties**

```
377     //
378     // ================================================
379     function resetRoom() {
380         //Room Exceptions
381         //Hard-coded Error corrections for 2DRooms array:
382         if (Rooms2D.CrntRoomX < 0) {
383             Rooms2D.CrntRoomX = 0;
384         }
385         if (Rooms2D.CrntRoomX > 3) {
386             Rooms2D.CrntRoomX = 3;
387         }
388         if (Rooms2D.CrntRoomY < 0) {
389             Rooms2D.CrntRoomY = 0;
390         }
391         if (Rooms2D.CrntRoomY > 3) {
392             Rooms2D.CrntRoomY = 3;
393         }
394 
395         //redraw the new room; hard-code room door exceptions
396         if ((Rooms2D.CrntRoomY == 0) || (Rooms2D.CrntRoomX == 2)) {
397             doorN.visible = false;
398             doorN.setInteractive(false);
399         } else {
400             doorN.visible = true;
401             doorN.setInteractive(true);
402         }
```

```
403         if ((Rooms2D.CrntRoomX == 3)
404             || (Rooms2D.CrntRoomX == 1 && Rooms2D.CrntRoomY == 1)) {
405             doorE.visible = false;
406             doorE.setInteractive(false);
407         } else {
408             doorE.visible = true;
409             doorE.setInteractive(true);
410         }
411         if ((Rooms2D.CrntRoomY == 3) || (Rooms2D.CrntRoomX == 2)) {
412             doorS.visible = false;
413             doorS.setInteractive(false);
414         } else {
415             doorS.visible = true;
416             doorS.setInteractive(true);
417         }
418         if ((Rooms2D.CrntRoomX == 0)
419             || ((Rooms2D.CrntRoomX == 2) && (Rooms2D.CrntRoomY == 1))) {
420             //Left Column doesn't have Western doors
421             doorW.visible = false;
422             doorW.setInteractive(false);
423         } else {
424             doorW.visible = true;
425             doorW.setInteractive(true);
426         }
427         //update Room HUD information
428         newHUD = "Room  #" + Rooms2D.CrntRoom +
429             "\nUse arrow key to move or\nClick on doorway.\nGrid: [" +
430             Rooms2D.CrntRoomX + "][" + Rooms2D.CrntRoomY + "]  \n
431             Visible doorway: ";
432         //debug console.log("=====");
433         //debug console.log(newHUD);
434         //debug console.log("=====");
435     }
```

## Phaser v2.x.x Code Review

***Play Phaser v2.x.x demonstration here***[43].

**Phaser v2.x.x is not compatible with Phaser v3.24+.** The more "pure JavaScript" we use in our game projects the more "compatible" Phaser v2 becomes to Phaser

[43] https://makingbrowsergames.com/starterkits/adventure/_p2-2DRooms/

v3. Therefore, we should always create our Game Mechanics (i.e., the rules and logic) in "pure" JavaScript and any "Canvas Visual Elements" are placed in the "Game Mechanisms" file unique to the Phaser API implemented.

The Phaser v2.x.x code uses the same logic as found in Phaser v3.16+ above. The only difference is "how to say" those instructions with Phaser v2.x.x syntax.

***Exercise:*** Download and use the following files with all the notes in this Code Review.

- ***v2_phaser2DRoomsJS.pdf***[44]
- ***v2_phaser2DRoomsDemoJS.pdf***[45]

### MainJS - v2_phaser2DRoomsJS.pdf

- Lines 1 - 28 are general administration and acknowledgments.
- Lines 29 to 37 create the 2D array data structure with hexadecimal flags, a graphics object, and game name-space.
- Lines 38 to 127 is the ***Game Mechanics (GM) Component.*** I placed the function that handles character movement in the GM component. It could just as easily appear in the ***Game Framework Mechanisms (GFM) Component*** for better "Separation of Concerns".
- Lines 129 to 138 is graphic box function. Once again, this could move to the Game Mechanisms Component for better "Separation of Concerns".
- Lines 203 to 207 launches the game using browser document instead of ***"`window.onload`"***.

### DemoJS - v2_phaser2DRoomsDemoJS.pdf

- Lines 1 to 25 are general administration and acknowledgments.
- Lines 27 to 211 create game mechanism variables, and standard Phaser Essential Functions — "create" and "update".
- Lines 212 to 294 makes the Room Doors "clickable". This is an alternate method for the avatar to travel around the environment. Let's be ***HONEST!*** Did you really like all that ***"traveling" in Diablo***[46]??

[44] https://makingbrowsergames.com/starterkits/adventure/v2_phaser2DRoomsJS.pdf
[45] https://makingbrowsergames.com/starterkits/adventure/v2_phaser2DRoomsDemoJS.pdf
[46] https://en.wikipedia.org/wiki/Diablo_IV

#### Lines 212 to 294

The difference in Phaser v2.x.x from the Phaser III code above is the use of "camera.fade", delayed time events, unique syntax of **"`game.world.centerX`"** and **"`centerY`"** and restarting a "Phaser State". Otherwise, the avatar placement is written in "pure" JavaScript which draws both Phase APIs closer and cuts development time.

#### Lines 294 to 306

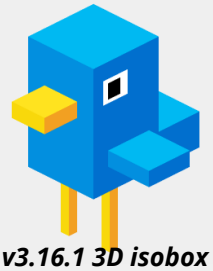
Creates the walls and doors, in a pure JavaScript canvas-drawing, which is similar to the Phaser III code.

#### Lines 305 to 350

These lines were consolidated into the ***"`newRoom`"*** function, but were retained for your study.

## 2.4 3D Prototypes

You can substitute ***"rectangle shapes"*** for images and sprites, and apply movement and physics reactions. It's even possible to build 2.5D and 3D games with just ***"`isobox`es".***



***v3.16.1 3D isobox***

Quote about the ***IsoBox Shape***[a] "... is a Game Object that can be added to a Scene, Group or Container — **WARNING: "some massive changes coming in v3.17+ about "containers.** It provides a quick and easy way for you to render this shape in your game without using a texture, while still taking advantage of being fully batched in WebGL. You can treat it like any other Game Object in your game, such as ***tween it, scale it, rotate it, alpha it, blend mode it, change its origin, give it a Camera scroll factor, put it inside a Container or Group, give it input abilities or even give it a physics body.*** It is ... a normal Game Object. The only difference is that when rendering it uses its own special bit of display code. ..."

This shape supports only fill colors and cannot be stroked.

An "IsoBox" is an 'isometric' rectangle. Each face of it has a different fill color. You can set the color of the top, left and right faces of the rectangle respectively. You can also choose which of the faces are rendered via the ***"`showTop`", "`showLeft`", and "`showRight`"*** properties. You cannot view an "IsoBox" from under-neath, however you can change the 'angle' by setting the projection property.

[a] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.IsoBox.html

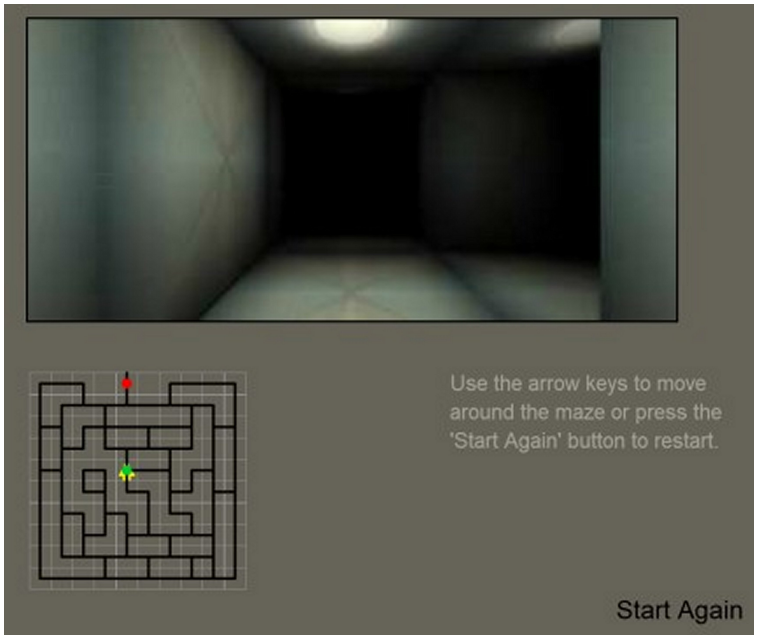
Exercise: Research the ***Phaser III.12 — Camera 3D Plugin***[47]

[47] https://github.com/photonstorm/phaser/tree/v3.22.0/plugins/camera3d

**Ensure the 3D Camera is loaded into a scene**

```
    this.load.scenePlugin({
      key: 'Camera3DPlugin',
      url: 'plugins/camera3d.min.js',
      sceneKey: 'cameras3d'
    });
```

```
// Deploy your 3D camera
var camera = this.cameras3d.add(85).setZ(500).setPixelScale(128);
```




**Phaser III ISO demonstration using 1st Person view in "Rescue of NCC Pandora"**™

See ***"Making 'Collapsing Blocks' Browser Games***[48] and its supporting ***website demonstrations***[49].

Quoted from ***newsletter #144,***[a]

"**Phaser3D** is a plugin that uses a new **Extern Game Object** and injects **Three.js** into it. It configures it properly for you, so that three.js can happily write to the current

[48] https://leanpub.com/mbg-collapse
[49] https://makingbrowsergames.com/starterkits/collapsingblocks/index.html

> context and then restores itself cleanly when it's finished. Because Externs sit on the display list like any other Game Object, you can layer your game content around a three.js scene. This means you could add 3D backdrops behind your 2D games, or 3D objects over the top of a 2D game, or any combination of. Of course, you can fully control three.js from Phaser too. This isn't some cut-down hobbled version of it, it's literally the entire library. Every single thing three.js can do, you can do via Phaser3D. There are loads and loads of helper methods to ease your workflow. These cover features like creating all supported forms of geometry, add spot, point or hemisphere lights, enabling shadows and fog, creating cameras, groups and all lots more. I've also included support for GLTF Models. You can, of course, load any other format, but as GLTF is the new standard, I included methods directly for it. Included in the bundle are no less than 34 examples, covering all kinds of different features, from geometry to cube maps to an example showing how to layer a normal 2D Phaser game over a 3D backdrop. There's even a little demo showing how to use Matter.js bodies for 3D objects. It's a powerful combination and I hope Phaser backers have fun playing with the demos and creating stuff.
>
> I will release Phaser 3D publically in a few months time, but for now backers get to play with this first, as they're the ones that enable me to work on Phaser full-time, so it's my way of giving back to them. It was also a really nice creative break for me. I had real fun putting the demos together and yet I only really scratched the surface of what could be done with it!"
>
> [a] https://madmimi.com/p/c1500e?pact=1015946-150303480-9209877774-f868549ae6f6d740beed7ce5868222c6f74a6aaf

## 2.5 "ToTo, … we're not in Kansas anymore" — Dorothy

***Welcome to OZ … er! uHMM! "PHAZ3R", Dorothy!***[50]

Phaser **pre-v3.16.x** was not for the "faint-heated". In those "early release months" (i.e., 20170201 to 20181025), due to the lack of hands-on tutorials and user documentation, it was difficult to architect any games using the "Bottom-Up" design method. Once documentation and a few great tutorials from ***William Clarkson (v3.9+)***[51] and ***Zenva Online Game Academy (v3.12+)***[52] began to appear, ***OOAD***[53] and ***"Bottom-up Design"***[54]

[50] https://www.youtube.com/watch?v=vQLNS3HWfCM

[51] https://www.udemy.com/course/making-html5-games-with-phaser-3/?ranMID=39197&ranEAID=pmlyJRiRsYE&ranSiteID=pmlyJRiRsYE-.Om65WbGQTSnaIiFty15zw&LSNPUBID=pmlyJRiRsYE&utm_source=aff-campaign&utm_medium=udemyads

[52] https://academy.zenva.com/?a=47&s=phaser&submit=Search&post_type=product&campaign=Phaser3GamePrototyping

[53] https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design

[54] https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design#:~:text=A%20bottom%2Dup%20approach%20is,environment%20to%20form%20a%20perception.

was then possible.

Phaser III is a sophisticated re-write that reminds me of its ***"grand-sire"***[55] ***KiwiJS***[56] — the grand-father of Phaser v2.x.x. Phaser III replaced ***"PIXI"*** with native code, and now supplies many new features that transcends any of its linage predecessors. In fact, Phaser III "scenes" remind me more of working with ***Flash MovieClips!***

> Quoting from the "DevLogs", "Fundamentally, v3.x.x **is completely different internally.** There's ***almost*** no code left over from v2.x.x. being used" ***(ed.: ... as I mentioned before, NOT EVEN PIXI! Yes, this is an incredibly bold business move, but they did it anyway. This created additional work on the new API and increased the restructuring required. In fact, Phaser v3 has undergone several massive rewrites since the formal conversion from "Lazer" into "Phaz3r"[a] in February 2017 until now (September 2018). The quote continues,)*** "However, we were very careful to keep the API as clean and friendly as possible." — quoted from FAQ #2 ***http://phaser.io/phaser3/faq***
>
> "Phaser 3 is the next generation of the Phaser Game Framework. Every last element has been rebuilt from scratch using a ***fully modular structure,***[b] ***(ed.: Please read "JavaScript Module Systems Showdown: CommonJS vs AMD vs ES2015"***[c]***.)*** combined with a data-orientated approach. It includes a brand-new custom WebGL renderer ***(ed.: PIXI is not used as stated earlier)*** designed specifically for the needs of modern 2D games." — quoted from R. Davey http://phaser.io/phaser3
>
> "Phaser 3 is now built entirely with ***webpack2.***[d] ***(ed.: research what webpack2 does***[e] ***to raw JavaScript code and how it works.***[f] ***)*** All of the code is being updated (or has been updated) to ***use CommonJS format modules.*** And webpack2 is managing the tree-shaking and package building of the whole thing. ***There are no grunt or gulp scripts to be seen anywhere, as we simply don't need them.*** On a side note I've also been using yarn for package management, and it's truly great! The speed is shockingly impressive." — quoted from R Davey https://phaser.io/phaser3/devlog/57
>
> [a] http://phaser.io/phaser3/history
> [b] https://webpack.js.org/concepts/modules/
> [c] https://auth0.com/blog/javascript-module-systems-showdown/
> [d] https://blog.madewithenvy.com/getting-started-with-webpack-2-ed2b86c68783#.fnuaum5tw
> [e] https://webpack.js.org/concepts/
> [f] http://kangax.github.io/compat-table/es5/

"Webpack2" outputs ***ES5 source code***[57]! Do we need to know this? Not just yet. Bookmark these sites for later use:

- ***Beginners guide to webpack—How to start a basic application with webpack***

[55] https://www.merriam-webster.com/dictionary/grandsire
[56] http://www.kiwijs.org/
[57] https://medium.com/@rajaraodv/webpack-the-confusing-parts-58712f8fcad9

***2***[58]

- ***A Beginner's Guide to Webpack 4 and Module Bundling***[59]

## 2.6 Starting *Your "Game Recipe"™*

There are so many cookbooks showing folks how to create delicious meals. It's time to have a "recipe" to build some "scrumptious" games!

### Step #0) the Front-Door

We need to load ***Phaser JavaScript Game Framework*** into a web page for it to work properly. Let's create two distinctive "front-door delivery systems" for our games. Once we have this setup, we can leave it alone; because, we will use D.R.Y (you remember! "Don't Repeat Yourself") in our file names. ***The only thing we'll need to adjust is the page's title and "`metadata`" inside each new project's "`index.html`".*** Both delivery methods are in the ***_v3.x.x-p3gp-book.zip***[60]

The first version is a standard ***"`index.html`"*** web page, and the second version is tailored for mobile devices as a ***"single web page application" (SWPA)***[61] or ***"Progressive Web App" (PWA)***[62]. I recommend using a ***"mobile-first, responsive design"***[63] for all "HTML" pages. There are two ways we can proceed:

1. the *"traditional method"* — see "Task #1-1" (below) or see the ***Appendix: Building HTML5 Web Page***[64] (3-pages)
2. the *"unorthodox method"* for mobile devices — see Task #1-2

For now, let's just follow the "traditional method" for a ***"Standalone" game.*** This traditional method needs two files; one file must be labeled ***"`index.html`";*** unfortunately, this is ***NOT*** a choice in our game development. As for the other file, name it whatever you like. I'll entitle mine as ***"`main.js`"*** and place it inside the subordinate directory/-folder labeled "js" — for JavaScript. The "main.js" holds my "Game Mechanics".

[58] https://medium.com/@ahsan.ayaz/beginners-guide-to-webpack-how-to-start-a-basic-application-with-webpack-2-ebed3172fa8c

[59] https://www.sitepoint.com/beginners-guide-webpack-module-bundling/

[60] https://makingbrowsergames.com/p3gp-book/_v3.x.x-p3gp-book.zip

[61] https://en.wikipedia.org/wiki/Single-page_application

[62] https://developers.google.com/web/progressive-web-apps/

[63] http://fredericgonzalo.com/en/2017/03/01/understanding-the-difference-between-mobile-first-adaptive-and-responsive-design/

[64] https://makingbrowsergames.com/book/Appendix-buildHTML5webPage.pdf

***Note:*** If you're curious as to why we must have an ***"index.html"? Answer:*** There are three (3) "default" pages used by most web server configurations. Those are "index.html", "index.htm", and on most Microsoft servers it is "default.htm". Here's how a webserver responds to a request. Whenever, a gamer makes a connection to a website, without specifying any URL file, most webservers will return the configured "default" page (aka "landing page"). For example, a gamer goes to ***"https://www.renown-games.com/"***[65], my server would find and return the URL ***"https://www.renown-games.com/index.html".*** Webmasters could configure their servers to use a different "landing file name", but only if they're bored or have time which is not the case in reality. Many will keep the defaults and spend their time on more pressing network matters.

## Task #1-1 Instructions:

1. Make a new copy of your ***Chapter 1 project directory*** for this new game project. (Refer ***Game Recipe™ Step #1*** above.)
2. Update the ***"`index.html`"*** header ***"`metadata`"*** and content with details about this project to improve Search Engine Optimization (SEO). (***Game Recipe™ Step #1***)
3. Either download this example ***6-page worksheet #1-1***[66] from here, refer to the ***Appendix: How to create an HTML5 web page***[67], or review the ***project starter index.html***[68] with ***"web socket" launched***. Use the developer console to watch the internal operations.

***Exercise:*** Observe a live "Bare-bones" ***"`Index`" Page***[69] here. This is an example of an ***"`index.html`"*** that is used for game prototyping only.

***Note:*** You might like to try the ***"15 seconds" HTML page creation tool.***[70] This responsive template comes with the ***"Golden Ratio"*** already pre-configured. You can read more about the ***"Golden Ratio" here***[71]. The Golden Ration is a weird mathematical proportion that our visual perception prefers. Learn even more about cutting-edge web design ***using the Golden Ratio here***[72] and in Phaser Game Design Workbook. Otherwise, if you've previously worked with "Bootstrap", you might like using their new ***Drop-n-drag Layout Builder***[73].

[65] https://renown-games.com./
[66] https://makingbrowsergames.com/book/ProjectIndex.pdf
[67] https://makingbrowsergames.com/book/Appendix-buildHTML5webPage.pdf
[68] https://makingbrowsergames.com/p3gp-book/standalone/index.html
[69] https://makingbrowsergames.com/book/demos/bareBonesIndex.html
[70] http://www.initializr.com/
[71] https://www.goldennumber.net/
[72] https://code.tutsplus.com/tutorials/the-golden-ratio-in-web-design--net-2272
[73] http://www.layoutit.com/build

You shouldn't have to change too much in this ***"index.html"*** file; you only need to modify the ***"<head>"*** metadata for each project. But look over my examples to ease your mind. This is, ***debatably (See Warning below)***[74], the absolute barest essentials for a properly formatted ***"index.html"*** page. In our ***Phaser III Design Guide workbook,***[75] we go into greater details concerning web pages and search engine optimization (SEO). You should find the complete ***"index.html"*** in the ***Source Code Appendix***[76].

***Warning: Google AMP***[77] ***"index.html"*** page **requires the head and body tags** in browser documents. ***Read more about it here***[78].

Inside your ***"index.html"*** you need to choose which Phaser version to use in your game. Notice that the Phaser scripts are minified and already come from the appropriate Content Delivery Networks (CDN). ***ALWAYS use the CDN versions for the fastest load times since Phaser v3.24.1 (minimized and zipped) is over 7MB and 43+ MB unzipped!*** "Why?", you ask? Because the minified CDN version of Phaser Framework are moved closer to your gamers and reduce their download time. It further increases the chance that a gamer may have Phaser III **already** in their browser cache which results in ***0 download time!*** If you develop your own unique version of Phaser, then you're gambling that someone from somewhere has played your game and migrated your "special pet files" to their local Internet Tier-3 Access Point. Read Yahoo's analysis on ***"empty cache" vs. "full cache"*** the ***"Surprising Results"***[79] (excerpt from Yahoo blog)

***Exercise:*** Study which CDNs are ***the fastest (click here)***[80]. This is a critical element in Massive Multi-Player online Games (MMoG).

There's more than what you see here! Download the following "Production" grade ***"index.html"*** pages and read their source code annotations:

- ***Production Optimized "index.html" Analysis***[81]
- ***AMP Mobile "index.html" Analysis***[82]
- **Neither of these** use ***"window.onload"!*** Refer to this article for ***more details on WHY!***[83]

[74] https://stackoverflow.com/questions/9797046/whats-a-valid-html5-document
[75] http://leanpub.com/phaser3gamedesignworkbook
[76] https://makingbrowsergames.com/p3gp-book/
[77] https://www.ampproject.org/learn/overview/
[78] https://www.ampproject.org/docs/getting_started/create/basic_markup
[79] https://yuiblog.com/blog/2007/01/04/performance-research-part-2/
[80] https://www.cdnperf.com/
[81] https://makingbrowsergames.com/book/ProjectIndex.pdf
[82] https://makingbrowsergames.com/book/ProjectIndex-Mobile.pdf
[83] https://javascript.info/onload-ondomcontentloaded

## Compare your code

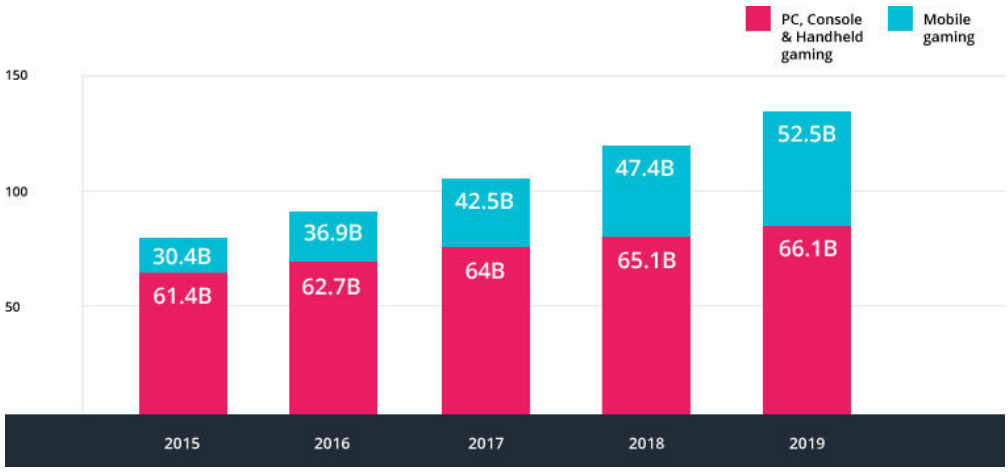
Here's the Chapter 1 **"Break Out" (Phaser III full source code)** as a bonus download:

- ***Phaser III full source code***[84]
- ***Phaser v2.x.x full source code***[85]

Here's what mine looks like, with all the ***'bells & whistles':***[86]

- My traditional "***index.html (v2.x.x)***[87]" for a ***Dating game*** — use the Developer's Console to watch some interesting dissection of the ***Phaser v3.16+***[88] version.
- My unorthodox method: "***index.html (v2.x.x)***[89]" for the same Dating game. — use the Developer's Console to watch some interesting dissection of Phaser v2.x.x.
- In my "***game.js,***[90]" I kept the game object inside this file for consistency.

## Mobile *"Single Web Page Applications"* (SWPA)




***Video game market revenue worldwide by Instabug Blog***

***Exercise:*** Read this report from ***Instabug on Mobile Game Development***[91].

[84] https://leanpub.com/c/p3gdc/c/Tx4iHQ6m64c5
[85] https://leanpub.com/c/phasergamedesignworkshop/c/3IWDBydPFVj1
[86] https://www.phrases.org.uk/meanings/bells-and-whistles.html
[87] https://makingbrowsergames.com/starterkits/quiz/game3/index.html
[88] https://makingbrowsergames.com/starterkits/quiz/p3game3/index.html
[89] https://makingbrowsergames.com/starterkits/quiz/game3/index-mobile-SWPA.html
[90] https://makingbrowsergames.com/starterkits/quiz/game3/index-mobile-SWPA.html
[91] https://instabug.com/blog/mobile-game-development-tools/

Let's look at the tailored "unorthodox" mobile device "index" page — Task #1-2. This construction is different than before; my goal is to load as much as possible into a single page without exceeding the "20 seconds" rule imposed by app stores. I have two different styles of mobile device pages. The example below creates a normal JavaScript link to the ***"main.js"*** (or ***"game.js"*).** I take a "less formal" approach in the mobile versions and try to "in-line" scripts inside the ***"index.html"*** * * tags. **The single web page application** is divided into ***" "*** sections. Each ***" "*** section represents a single game phase menu and the ***"game.js"*** is placed into the "play game" ***" ".*** If the game is small enough, I will simply insert the entire raw ***"game.js"*** contents directly into a ***"script"*** tag and thus avoid an additional file to download. Doing so ensures all the game's content is an embedded part of the ***"index.html"*** page.

**Single Web Page Application (SWPA mobile)**

```
<!doctype html>
<html lang="en">
<head> ... </head>
<body> ...

<!-- Mobile Dating game -->
<div  class="ui-content" data-theme="b" data-role="page" id="game">
<div data-role="header">
    <h1><b>(Your Game Title here)</b></h1>
<a href=" " class="ui-btn ui-mini ui-icon-home ui-btn-icon-left ">Home</a>
</div>

<div id="game-area" data-role="main" class="ui-content">
</div>


<!-- import external file or simply include its full contents -->
<script src="game.js"> </script>


<div class="ui-content center footer" data-role="footer">
<hr class="center" style="width: 60%" />
<nav class="menu"><a href=
'http://www.copyright.gov/fls/fl108.pdf' target='_blank'> Copyright </a>
    &copy; 1978-2016, <a class="w3-btn btn-footer w3-hover-deep-orange
w3-theme-d3 w3-round-xlarge w3-border w3-text-shadow "
href='http://www.stephen-gose.com/en/' target='_blank'> Stephen Gose LLC
</a>. <br /> All Rights Reserved. <br />
Questions or comments?
```

```
<a class="w3-btn btn-footer w3-hover-deep-orange w3-theme-d3
w3-round-xlarge w3-border w3-text-shadow "
href="http://www.stephen-gose.com/about/contact/"> Please Contact </a>
<br />
<hr class="center" style="width: 60%" />
</nav>
</div>
</div>
<!-- End Game Page -->
```

All that remains is a method to bind all these into a ***single web page application (SWPA)***[92]. Using a single monolithic file has advantages per Google's ***Accelerated Mobile Pages Project (AMP)***[93]. We'll do this through our game's ***"index.html"*** page. Many authors create yet another script file, but I prefer to use inline scripting for mobile devices.

***Exercise:*** Worksheet #1-2 Mobile ***"index.html".*** You can see this "live example" at:

***https://makingbrowsergames.com/starterkits/quiz/game3/index-mobile-SWPA.html***

## Cocoon.js - Cloud Alternatives

**20190228: Cocoon Termination of the Service.** Unable to access Cocoon. Customer service closed. They will not retain copies of any of your data from that date forward. Accordingly, you are encouraged to download and keep copies of your data if you wish to have access to it in the future.

In case you prefer not to use the **Apache Cordova CLI**[a] for building your apps, there are a few Cordova base cloud services that might serve the purpose as long as you update the config.xml to not use Canvas+ or Webview+.

[a] https://cordova.apache.org/

[92] https://www.seguetech.com/what-is-a-single-page-application/
[93] https://www.ampproject.org/

- **Phonegap Build**[94] — From the team behind **Apache Cordova,** the **Adobe® PhoneGap™** framework is an open-source distribution of Cordova — providing the advantage of technology created by a diverse team of pros along with a robust developer community — plus access to the PhoneGap toolset, so you can get to mobile faster. Write a PhoneGap app **once with HTML and JavaScript** and deploy it to any mobile device without losing features of a native app. **Adobe® PhoneGap™** is a standards-based, open-source development framework for building cross-platform mobile apps with HTML, CSS, and JavaScript for iOS, Android™, and Windows® Phone 8. Simply upload your HTML5, CSS, and JavaScript assets to the **Adobe® PhoneGap™ Build cloud service** and they do the compiling for you.
- ***Ionic Framework***[95] — Learn the difference between hybrid and native. They break down all the myths and misconceptions in their ***FREE ebook***[96].
- ***Monaca***[97] — makes HTML5 hybrid mobile app development with **Adobe® PhoneGap™**/Cordova simple and easy. Monaca is the most open hybrid app development platform available and ready to be immediately plugged into your existing work-flow and development environment. From Cloud IDE, CLI to debugger, and remote online build, everything you need for your hybrid app development is there.

***Exercise:*** Read about the ***differences between Cordova and PhoneGap***[98].
***Exercise:*** Read how to ***migrate from Cocoon.js to Cordova***[99].

## Task #2: Launching a Game

Phaser III, v2.6.2, and CE versions ***are all launched from within a web page*** as either an inline JavaScript script or from an external file using JS modules. What happens next differentiates each Phaser API family versions from its siblings. ***The official examples***[100] put the game launching code and all the ***"Phaser Essential Functions"*** into a single ***"`index.html`"*** file. I prefer using separate files while developing my game. Because it helps me focus on the task at hand and localizes software bugs to the file currently under development.

[94] http://docs.phonegap.com/phonegap-build/
[95] https://ionicframework.com/docs
[96] https://ionicframework.com/books/hybrid-vs-native
[97] https://monaca.io/
[98] https://ionicframework.com/resources/articles/what-is-apache-cordova
[99] https://docs.cocoon.io/article/from-cocoon-io-to-apache-cordova/
[100] http://labs.phaser.io

**Sample: Phaser v3.x.x *'Essential Functions'* as an Anti-Pattern in Official methods**

```
/**
 * Anti-Pattern Warning:
 *
 * Polluting the global namespace with global context variables
 *
 * init:     function init() {},    //initial game phase data
 * preload:  function preload() {}, //queue & download game assets
 * create:   function create() {},  //make cached assets available
 * update:   function update() {},  //begin the game loop
 * render:   function render() {},  //render current display
 * shutdown: function shutdown() {} //close and garbage collection?
 *
 */
```

Phaser.io examples are "just that" ***— examples!*** If you attempt to follow these examples ***"verbatim" as "holy writ",*** you will soon discover that you're ***"painted into a corner"***[101] when using your Phaser III Scenes as if they were Phaser v2.x.x "States".

It's time to return to our ***"`game.js`"*** (or **Create it** now with whatever name you'd like). In this file, let's fill it with the following downloaded content available in the ***online Source code Appendix.***[102]

**Example: 2.2 Launching a Game - two methods.**

```
58  //
59  // ====================================
60
61  // ============
62  // Example: 2.2a Launching as a name-space.
63  // ============
64
65  // window.GAMEAPP.main(); //name space activation
66  // console.log("Game obj: === Ext? "+Object.isExtensible( GAMEAPP ));
67  // console.log(Object.values(GAMEAPP));
68  // console.log(Object.getPrototypeOf(GAMEAPP));
69  // console.log(Object.getOwnPropertyDescriptors(GAMEAPP));
```

[101] https://idioms.thefreedictionary.com/paint+into+a+corner
[102] https://makingbrowsergames.com/p3gp-book/index12.html#12.3

```
70      // OR: global variable launched; similar to Phaser v2.6.2
71
72
73      // ============
74      // Example: 2.2b Launching as a Global variable.
75      // ============
76      // var gWidth  = 800;  //Using Golden Ration is important.
77      // var gHeight = 500;  //Using Golden Ration is important.
78      // Lessons learned from colleagues
79      // initial size determined
80      // creates a global variable called game
81      // ====================================
82      var game = {};
83      var gWidth, gHeight;
84      var isMobile=navigator.userAgent.indexOf("Mobile");
85
86      if (isMobile != -1) {
87          //-1 is desktop/anything other than mobile device
88          console.log("isMobile="+isMobile);
89          gWidth = window.innerWidth * window.devicePixelRatio;
90          gHeight = window.innerHeight * window.devicePixelRatio;
91          //resize();
92      }
```

I'm ***building a unique "name-space"***[103] for my game prototype in this second example. In ***Bob Nystrom's book, "Game Design Patterns",***[104] he warns about using object expressions as ***"`singletons`".*** This is "mandatory" reading for everyone with less than 15 years in software engineering — I have 37 years in networking; so, this includes me too! He states, ***"Despite noble intentions, the Singleton pattern described by the Gang of Four usually does more harm than good.*** "... Like any pattern, using Singleton where it doesn't belong is about as helpful as treating a bullet wound with a splint. Since it's so overused, most of this chapter will be about ***avoiding singletons,*** but first, let's go over the pattern itself. ..." (Nystrom)

This creates a new blank ***"<canvas>"*** as our game's stage; it has a black background that is 800 pixels width by 500 pixels tall — ***the "Golden Ratio".*** All of our game elements will be inside of this game "world" box. Time to double-check our work so far; save everything. Then double-click on your ***"`index.html`"*** file; your browser should open to show a large black rectangle. Right?

[103] https://javascriptweblog.wordpress.com/2010/12/07/namespacing-in-javascript/
[104] http://www.gameprogrammingpatterns.com/singleton.html

***Advanced Exercise:*** Compare your work ***to another example.***[105] Open the Developer's Console and study what I've done with the "Game Object" namespace and "Phaser.Game" object in the console's drop-down menus. ***This example also provides a timing test between the*** `window.onload` ***vs.*** `document.onload`***.***
***Exercise:*** Read about using the "Singleton pattern" in game design and development from ***Bob Nystrom's book, "Game Design Patterns"***[106].
***Exercise:*** Download this 3-page ***"main.js" example file***[107] I use as my standard prototype foundation and Refer to lines 112 to 150 in the file (you just downloaded? Right?).

**Example: 2.2 continued.**

```
94      ;   //Closes any previous scripts
95      //
96      // ====================================
97      // creates our Phaser Game configurations.
98      //    dozens of configurations parameters;
99      var config = {
100         width:  gWidth || 800,      //Using Golden Ration is important.
101         height: gHeight || 500,     //Using Golden Ration is important.
102         type:   Phaser.AUTO,
103
104         //Game Title
105         title: 'Phaser3 Game Prototyping Starter Kit',
106
107         //Game URL
108         url: 'https://makingbrowsergames.com/p3gp-book/',
109
110         //https://semver.org/ + DATE
111         version: '0.0.1.2016 semver ',
112
113         //Custom RGB color or "#369"
114         backgroundColor: 0x336699,
115         input: {
116             keyboard: true,
117             mouse: true,
118             touch: true,
119             gamepad: false
120         },
121         physics: {
```

[105] https://makingbrowsergames.com/p3gp-book/standalone/
[106] http://www.gameprogrammingpatterns.com/singleton.html
[107] https://makingbrowsergames.com/p3gp-book/_mainp3.pdf

```
122             default: 'arcade',
123             arcade: {
124                 // Debug turned on for arcade physics
125                 debug: true
126             }
127         },
128         scene:        {
129             main: main,
130             combat: combat,
131             gameOver: gameOver
132         },
133         pixelArt: false,        //set TRUE for retro styling
134         antialias: true
135         //parent:        document.body
136         };
137         console.log("Configure Obj: Ext? "+Object.isExtensible(config));
138         //console.log(Object.values(config));
139         //console.log(Object.getPrototypeOf(config));
140         console.log(Object.getOwnPropertyDescriptors(config));
141 
142     //
143     // ====================================
```

- Line 94: I used a ***" ; "***. Why? Well, there are a lot of folks telling everyone you don't need to use the "semi-colons" because JavaScript automatically inserts the semi-colons for you; so, let's just forego typing them. Well, ***hold onto you hats***[108] cowboy! Read what ***Javascript Gardens*** has ***to say about using semi-colons***[109]. JavaScript gets a lot of "back-wash" about how terrible the language is, when in reality, it's the lazy programmers who incite JavaScript to "read their minds". Oh! and while I'm on the topic of poor programming, you must always use the curly braces ( ***"{ }"*** ) too. Here's JavaScript Garden's take on those topics.

> Quote: "It is highly recommended to ***never omit semicolons.*** It is also recommended that braces be kept on the same line as their corresponding statements and ***never omit them for single-line if / else statements.*** These measures will not only improve the consistency of the code, but they will also ***prevent the JavaScript parser from changing code behavior."***

[108] https://youtu.be/EpgP2Gtx8QY
[109] https://bonsaiden.github.io/JavaScript-Garden/#core.semicolon

***Exercise:*** Download this example from: ***_p3-demos/game.js***[110]
***Advanced Exercise:*** Review advanced game setup using ***"_index.html"***[111]
***Advanced Exercise:*** Review advanced game setup using ***"_index-mobilep3.html"***[112] in the browser's console

## Deeper Dive: Launching a Phaser III Game.

When Phaser v3.x.x boots, it creates an instance of a ***"`Phaser.Game`". It could load an optional `Game Configuration object` (ed.: which is now mandated in Phaser v3.x.x),*** which is passed into the ***Config handler (see source code)***[q], and all the various things it needs are extracted from the ***"`config object`".***

[q] https://github.com/photonstorm/phaser/blob/dd39f9ab08d57fa1bacd1287ccadb03fb3151267/src/core/Game.js#L388

***http://phaser.io/tutorials/getting-started-phaser3 OR***
Run in the Cloud: ***http://phaser.io/tutorials/getting-started-phaser3/part3***

## Game "`Config`"

Game ***"`config`"*** has been around since before ***v2.4.2.***[113] It's a JavaScript ***"`object`"*** that holds all the initial game configurations.

Open ***"`main.pdf`"*** (3-pages you downloaded from above) and read lines 160 to 163. This waits for the browser to finish the Document Object Module (DOM) and then calls the ***"`window.GAMEAPP.main()`"*** which begins on lines 118 to 140. Line 120 creates an internal variable ***"`this.game`"*** that holds the ***"`new Phaser.Game`"*** object.

[110] https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js
[111] https://makingbrowsergames.com/p3gp-book/_indexp3.pdf
[112] https://makingbrowsergames.com/p3gp-book/_index-mobilep3.pdf
[113] https://labs.phaser.io/index.html?dir=game%20config/&q=

**Sample: Bare-bones v3.x.x `Config` object & `Phaser.Game`**

```
    <script>
    var config = {
        width: window.GAMEAPP.viewportWidth,   //x width using main.js
        height: window.GAMEAPP.viewportHeight, //y height using main.js
        type: Phaser.AUTO,                     //.WEBGL or .Canvas
        parent: gameCanvas                     //canvas container
        scaleMode: Phaser.ScaleManager.EXACT_FIT //NOT available in v3.x.x
    };

    //Deeper Dive with Analysis
    console.log("Configure Obj: Ext? "+Object.isExtensible( config ));
    console.log(Object.values(config));
    console.log(Object.getPrototypeOf(config));
    console.log(Object.getOwnPropertyDescriptors(config));

    /** Phaser III config as a function?
    // config experiment as a function.
    // best placed in the index.html file since functions are hoisted.
    function config() {
        var width = window.GAMEAPP.viewportWidth, //x width using main.js
        height = window.GAMEAPP.viewportHeight,   //y height using main.js
        type = Phaser.AUTO,                       //.WEBGL or .Canvas
        parent = gameCanvas;                      //canvas container
    };
    */

    /** Phaser III config as a lambda? (as of 20181223)

        See: http://labs.phaser.io/edit.html?
            src=src/scenes/change%20scene%20from%20objects.js
        // ... .
        class SceneC extends Phaser.Scene {
            constructor() {
                super('SceneC');
            }
            create() {
                console.info('SceneC started.');
                this.add.image(160, 120, 'aqua_ball')
            }
        }
```

```
    //NOTICE: the config object is simply embedded!
    //  experiment with this.
    var game = new Phaser.Game({
        type: Phaser.AUTO,
        parent: 'phaser-example',
        scene: [SceneA, SceneB, SceneC]
    });

*/

/**
 *  DEPRECATED METHOD - NEVER EVER USE THIS AGAIN!
 *  See "Phaser Game Design Workbook" for complete explanation
 *  http://leanpub.com/phaser3gamedesignworkbook
 *
 *  window.onload = function () {
 *     // local scope used????
 *     let game =  ... ...
 *  };
*/

//Global name-space used
var game = {};

//preferred game launch method.
document.addEventListener('DOMContentLoaded', function(){
   //standard Phaser III launch method
   game = new Phaser.Game(config);
/**
//Strangely, old v2.6.2 also works in Phaser III!!!
//  try it out and learn what happens!
game = new Phaser.Game(
        gWidth, gHeight,    //width and height of canvas
        Phaser.AUTO,        // how to render canvas
        "gContent");        // place canvas inside div
*/
console.log("Game obj: Ext? "+
            Object.isExtensible( game ));
//console.log(Object.values(game));
//console.log(Object.getPrototypeOf(game));
console.log(Object.getOwnPropertyDescriptors(game));

console.log("Phaser.Game prototype: Ext? "+
```

```
            Object.isExtensible( Phaser ));
    //console.log(Object.values(Phaser));
    //console.log(Object.getPrototypeOf(Phaser));
    console.log(Object.getOwnPropertyDescriptors(Phaser));
    }, false);
    // Example: 2.2 ends
    // ============

    </script>
```

Do you need to do something as extensive as I've provided in my examples ***"`index.html`"*** and ***"`main.js`"*** files? No, not really.

***Exercise:*** Compare what we're doing with ***my examples***[114] to the ***"Official Phaser v3.x.x. tutorial"***[115]

## 2.7 Deeper Dive: To Infinity and Beyond!

Notice how Phaser v3.x.x uses its ***configuration object.***[116] ***Let's take the next step! This "`config`" object could easily become a "`JSON`" data object passed into a Phaser v3.x.x game. This permits dynamic game set-ups based on who plays, what permissions they are granted, and how they access various game phases. Furthermore, we can create a separate "`config`" for each game scene.*** We could go so far as to define "a different config" for every level inside our game; or better yet, ***display separate game editions for those who have "FREE" access from those who have membership "PAID" access.*** Let your imagination run wild! Truly, Phaser v3.x.x opens up more game management and access possibilities than the former v2.x.x.

***Exercise:*** Review the default parameters for the ***Phaser v3.x.x. config***[117]

***Warning:*** There is a limit of ***255 arguments passed into a JavaScript function per MDN.***[118]

[114] https://makingbrowsergames.com/p3gp-book/_p3-demos/
[115] http://labs.phaser.io/
[116] https://github.com/photonstorm/phaser/blob/dd39f9ab08d57fa1bacd1287ccadb03fb3151267/src/core/Game.js#L25
[117] https://github.com/photonstorm/phaser/blob/dd39f9ab08d57fa1bacd1287ccadb03fb3151267/src/core/Game.js#L25
[118] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions

## 2.8 Summary

Whew! Chapter 2 down! Here's what we've covered.

Examples: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/***

- Understand what a game prototype contains.
- Learned the benefits of building a game prototype.
- Discovered OOP is **NOT** the best approach for game design as stated by Apple Game Developers.
- Read insightful tips from various developers about how to rapidly build games.
- Saved 1,000s of hours in development time.
- Practiced the ***4-steps*** of creating new game mechanisms and prototypes.
- Built our game's "front door" in various delivery styles.
- Downloaded helpful resources in game development.
- Differentiated between various Phaser v3.x.x. formats for production & development.
- Studied how a ***Content Delivery Network*** impacts a client's enjoyment.
- Discovered which ***CDN*** has the best performance.
- Researched Google's AMP.
- Learned about encroaching "Anti-patterns" slipping into Phaser.
- Understand how to filter clients using `config`.
- Read about alternate methods for re-size our game.

## 2.9 Chapter References:

- ***Apple Game Developers GameKit***[119]
- ***Intel: Resources for Game Developers***[120]
- ***MDN - Implementing game control mechanisms***[121]
- ***Aspect-oriented programming***[122]
- ***MV* frameworks***[123]
- ***Rewriting A WebApp With ECMAScript 6***[124]
- ***How to create Phaser v3.16.x Graphics***[125]

[119] https://developer.apple.com/documentation/gamekit
[120] https://software.intel.com/content/www/us/en/develop/topics/gamedev.html
[121] https://developer.mozilla.org/en-US/docs/Games/Techniques/Control_mechanisms
[122] https://en.wikipedia.org/wiki/Aspect-oriented_programming
[123] http://todomvc.com/
[124] https://medium.com/tastejs-blog/rewriting-a-webapp-with-ecmascript-6-39417b642cb2
[125] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Graphics.html

- ***Turning static graphics into Sprites***[126]
- ***Sample Stacker Game using "shapes"***[127]

[126] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Graphics.html#generateTexture
[127] http://labs.phaser.io/view.html?src=src%5Cgame%20objects%5Cshapes%5Cstacker%20es6.js&v=128

# 3. Game Phases, Scenes & Roses.

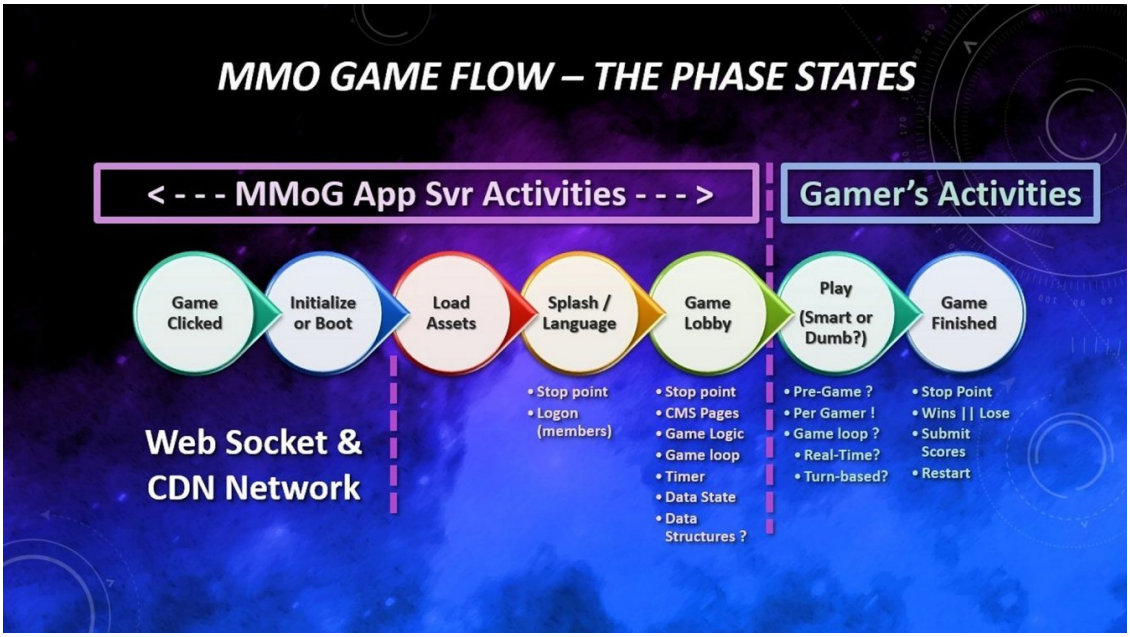
***"A rose is a rose ... by any other name", paraphrased from ...***
*"Rose is a rose is a rose is a rose." — **Gertrude Stein***[1]; and
*"What's in a name? That which we call a rose, By any other name, would smell as sweet." — **William Shakespeare***[2].

The "Game's logical flow" is the path our gamers follow ***despite*** which Phaser version we've deployed ***— or any JavaScript Gaming Framework for that matter.*** When a gamer launches our game from its "index.html" page, we lead them through a series of **stages that I call "game phases"**. Some developers call these "game states" from a reference to ***Finite State Machine (FMS)***[3]. Eventually, our gamer will arrive at a "play" button somewhere on the *"main menu"* to start the ***"Gaming Play Loop " (aka "the event loop").***

## 3.1 Bare-Bones Prototypes

The illustration below is common to any game found on the Internet; it is ***NOT unique*** to Phaser JS Gaming Frameworks. It is a design concept and progression a player takes through a game. Notice that there are two aspects.

[1] https://en.wikipedia.org/wiki/Gertrude_Stein
[2] https://en.wikipedia.org/wiki/William_Shakespeare
[3] https://en.wikipedia.org/wiki/Finite-state_machine




**Game Phases as JS Modules in a Massive Multi-Player online Game (MMoG)**

- Delivering the game across the "Cloud" (Internet, WAN, telcos, the name keeps changing based on marketing services.)
- Content is placed on the local gamer's device.

What I'm calling a **"game phase"** is a place during our **game's flow.** For example, many games provide a "splash screen" — showing sponsorship, advertisements, logos, a "downloading progress bar" and such — while booting-up the initial game settings and downloading most of the game's assets. Another example is the game's "Main Menu" from which a player can choose various options as pictured above. Many game developers simply refer to these "game phases" as ***"`states`"*** — from a programming technique known as a "finite state machine" (FMS). The "FMS" helps us "bookmark" where a player is inside our game (i.e., their progression) and helps us determine what to show them. Obviously, the gamer can switch between game phases and return to former game phases, but there is ever ***only one active game phase presented at any time.*** For example, moving into the "Game Finished phase" from a `Win` or `Lose` event and then returning to the game's "Main Menu" phase. For this very reason, many Phaser games are just simple "single staged" game-plays — they ***DO NOT USE a game shell.***

All of our games follow this similar game-flow pattern ***— despite which Phaser version used or whether we're using another Gaming Framework altogether!*** As the gamers migrate through our game(s), they have options; it doesn't matter ***what we call*** these sections — ***"roses"***[4], phases, states, menus, scenes, screens, thingies, ***dumaflache,***[5]

[4] https://www.thefreedictionary.com/rose-like
[5] https://www.urbandictionary.com/define.php?term=dumaflache

or ***"Aardvarks!"***[6] It's just a matter of focusing on ***what the gamer is allowed to do inside each "part" of our game.*** In the Phaser community, there is a lot of confusion over these ***"roses"*** — resulting from vague descriptions and inarticulate definitions about what they are. ***In the new Phaser III,*** I'm seeing the same confusion beginning all over again. As a review, think of a "game phase section" as if they were ***"`menus`" — in other words, simply individual "JavaScript modules" —*** those various JavaScript files you create ***to concoct***[7] your game-flow migration. Basically, if you took your game and separated it into various "phase sections", such as a splash screen, main menu screen, the game-play itself, and so on ***(ed.: sounds like the drill we did in Chapter 1? Right?),*** each of these "chunks" would match a phase in the gamer's progress through a game — each phase has ***its own separate and internal collection of "Phaser Essential Functions".*** In short, the new ***"`Phaser.Scene`"*** in v3.24+ reminds me **more of an Adobe Flash MovieClips** on its main timeline.

## 3.2 Using a Phaser Scene as a "Game Phase"

Inside a "scene", Phaser uses 9 internal ***"Essential Functions".*** Many Phaser game developers, at this point, will create a new separate JavaScript file (i.e., a module) for each "game phase" to act as a stand-alone "Phaser III Scene" with its "9 essential functions". But for now, we will keep this simple; later in this chapter, we'll begin to separate our game phase prototypes into separate ***JavaScript modular files.*** This will provide the maximum flexibility in our software development when we begin to mix-and-match and re-use our "D.R.Y. code".

Each of these "game phases" (aka movieClips, screens, scenes, states, or ***"Roses"),*** as we discussed above, has **"its own internal essential functions".** These functions give us a way to organize our code inside each separate "game phase" module and ensure that only the minimal game assets (for the current phase) are supplied at just the proper time. These **"essential functions"** help us isolate distinct "game flow events" from each other. For example, booting the game, loading assets, main menu, playing levels, winning, losing, **all have their unique individual "`initiate`", "`preload`", "`create`", and then "`update`" and "`render`" essential functions.**

The goal we achieve, by using this "Finite State Machine (FSM)" (aka "game phases") structure, makes our game development simpler and less painful to support.

Now that we understand this, let's talk about ***Phaser III "Scenes".*** You can have **multiple Phaser III Scenes inside a single Game Phase.** In the following game phase, I have 5 "Phaser III Scenes" all running at the same time in the "Play" phase. I

[6] https://en.wikipedia.org/wiki/Aardvark
[7] https://www.vocabulary.com/dictionary/concoct

use Phaser III Scenes as I did with "movieClips" in Adobe Flash. I have the main timeline (aka "Play phase") with multiple movieClips (i.e., "Phaser III Scenes") for various display sections.


**5 "Phaser III Scenes" inside the "Play Game" Phase**

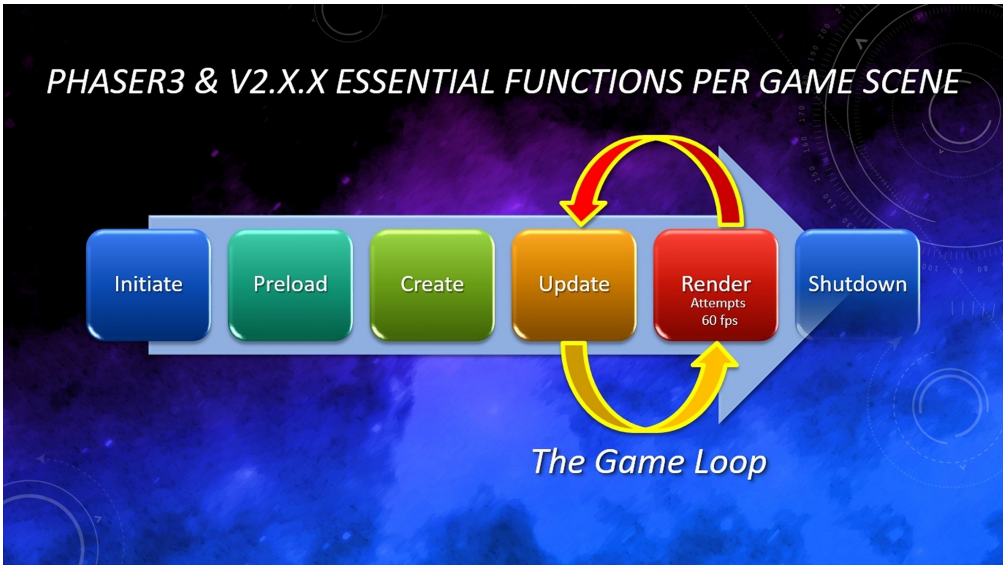
Play the ***demonstration here***[8] from ***"Making RPG Browser Games"***[9].

***Exercise:*** Read DevLog #119 **https://phaser.io/phaser3/devlog/119**

## 3.3 *9 Essential Functions* of a Phaser *"Scene"*

***Note:*** *"... Essential Functions ..."* is not a term I invented! ***It comes directly from Official Phaser tutorials!***[10] So folks, ***let's call a spade a spade, and not a gardening tool***[11] shall we?!?!

[8] https://makingbrowsergames.com/starterkits/rpg/_arrp-phaser/p3/
[9] https://leanpub.com/mbg-rpg
[10] https://phaser.io/tutorials/making-your-first-phaser-2-game
[11] https://en.wikipedia.org/wiki/Call_a_spade_a_spade



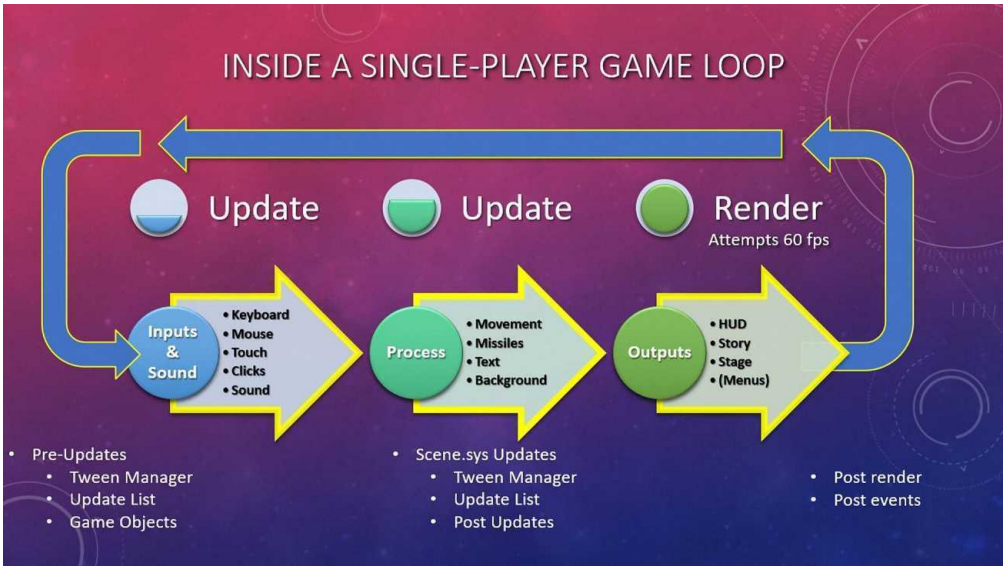

**"Phaser Essential Function" found inside every Scene**

Phaser JavaScript Game Framework ***— any version! —*** uses several encoded "functions" to process itself. Quoted from ***"Making Your First Phaser Game"***[12] — "Phaser supports a ***full "State" system*** allowing you to break your code into much cleaner ***JavaScript single-objects.*** But for a simple Getting Started guide such as this we'll use this approach as it allows for **faster prototyping … "; and, in this book, so will we!** The two most important ***"Essential Functions"*** are the ***"`create()`"*** and the ***"`update()`"*** functions ***within a single game scene "life-cycle".*** The ***"`create()`"*** function places all the game's visual elements inside an HTML5 canvas; the ***"`update()`"*** function attempts to refresh the display 60-times per second.

***Phaser III Scene Constants:***[13] Each Game Phase also has these additional functions ***(sorted alphabetically with its sequence of execution):***

| Name | Order |
|---|---|
| CREATING: | 4 |
| DESTROYED: | 9 |
| INIT: | 1 |
| LOADING: | 3 |
| PAUSED: | 6 |
| PENDING: | 0 |
| RUNNING: | 5 |
| SHUTDOWN: | 8 |
| SLEEPING: | 7 |
| START: | 2 |

[12] http://phaser.io/tutorials/making-your-first-phaser-3-game
[13] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/const.js




**Phaser III Game Updates and Render Displayed**

Flow chart of the ***Phaser III Game Loop.***[14] here.

***During these updates,*** Phaser checks for:

- any player's inputs from the keyboard, touch and/or mouse clicks;
- calculates any collisions between game objects;
- and further processes anything else we want our game to do.

Then Phaser "paints" these changes to the HTML5 ***"`canvas`"*** in the output "render" phase. After "rendering", Phaser returns, once again, into the ***"`update`"*** function checks for new "inputs to process" to begin the next "Circle of Life" for the game loop `event`.[15]

We can associate all these *essential functions* ***inside each "Scene".*** When a Phaser III ***"`this.scene.add`"*** is created, it automatically has the following systems set inside it. Examples of the Scene Systems are:

- The Game Object Factory
- The Loader
- The Main Loop
- The Update Manager
- A Camera

[14] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/mainloop/#flow-chart
[15] https://www.youtube.com/watch?v=GibiNy4d4gc

- Event Dispatcher

**https://phaser.io/phaser3/devlog/121** describes the new capabilities that a ***"`Phaser.Scene`"*** offers. In my mind and coming from an Adobe Flash background (as Davey has too!), ***the new Phaser III Scenes behavior reminds me of multiple Flash `MovieClips` along its timeline.*** Other commonly used "Essential Functions" inside of a ***"`Phaser.Scene`"*** are ***charted here***[16]:

- ***initialize***[17] — A method called when any Scene starts. It is passed as an argument variable to facilitate data-sharing among the different Scenes. ***It must be called initialize or you risk adopting the default Phaser III object.***
- ***preload***[18] — A method called whenever any Scene begins. It is used for loading your game's resources and assets before their use. Normally, in Phaser III, you'd load your game assets just for the current Scene. ***Each scene in Phaser III now has its own Load manager.*** If you call any ***"`this.load`"*** method from outside of the ***"`Scene.preload`"*** function, you need to start the Loader yourself by calling ***"`Loader.start()`"*** — it's only automatically started during the ***"`Scene preload`".*** Game assets, loaded by a ***"`scene`"*** Loader, are placed into global game-level caches. You shouldn't create or make any objects during the preloading. Since the load scene may take up to 20 seconds, this a good time to use a "Splash Game Phase". A Splash phase shows a game's title, sponsors, and legal copyright notices. You should build a gamer's anticipation for the upcoming gameplay. A progress bar aids in that anticipation; it is your "launch count down" to "fun". Another dynamic element, in addition to the progress bar, is a "spinner" — it tells the gamer that your game code is still running and nothing's crashed.
- ***create***[19] — A method called automatically after preload finished; it is used for generating game objects. If you didn't actually load any game assets at all or didn't use the preload function, then create is the first function executed by the Phaser III engine. Use create to set-up the bulk of your visual elements from the downloaded game assets.
- ***update***[20] — The "work-horse" of the Phaser JavaScript Framework. The update method is left empty for your own use. It is called during the core game loop AFTER debug, physics, plugins and the Stage have had their "pre-Update" methods called. It is used for user input polling and game object collisions and detection. This method is often used to capture game events (such as key presses, button clicks, mouse movement, etc.), and then update those variables as a result. It is called on every frame. The engine attempts to execute, at best efforts, 60

[16] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scene/#flow-chart
[17] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/Systems.js#L35
[18] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/SceneManager.js#L478
[19] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/SceneManager.js#L639
[20] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/SceneManager.js#L628

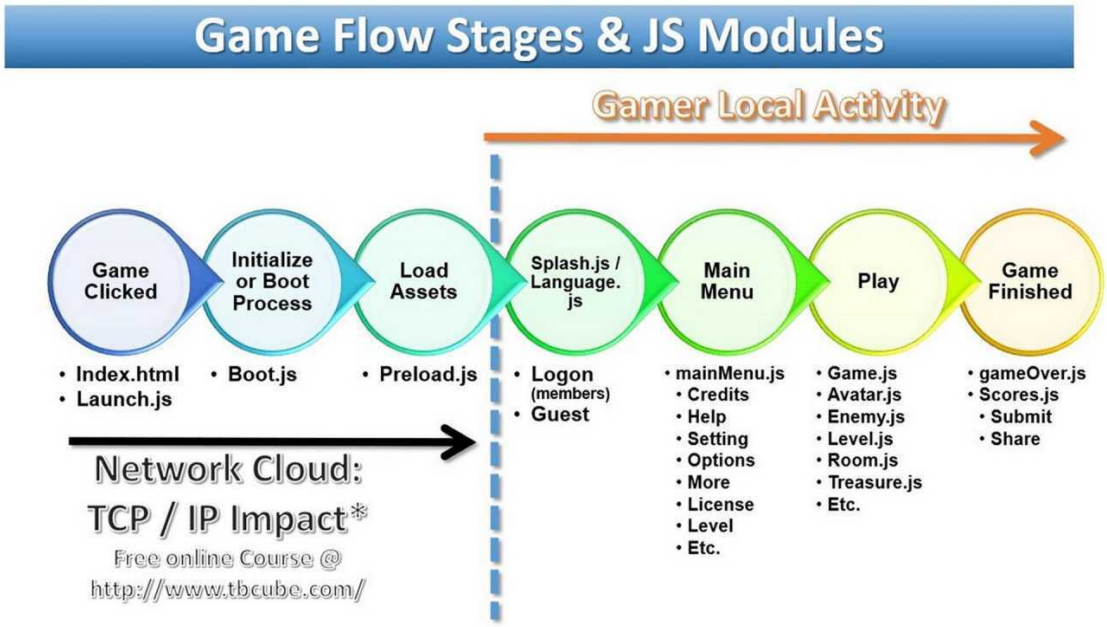
times per second, but that is not guaranteed. It is called BEFORE Stage, Tweens (see ***Deeper Dive 3.19+ Tweens***), Sounds, Input, Physics, Particles, and Plugins have had their "post-Update" methods called.

- ***render***[21] typically follows "update"; it flushes the information to the display. Nearly all display objects in Phaser render automatically, you don't need to tell them to render. Richard Davey warns that "The render function is called AFTER the WebGL/canvas and plugins render has taken place, so consider it the place to apply post-render effects or extra debug overlays." You're able to do any final post-processing style effects here. Note that this happens before plugins "post-Render" takes place.
- ***shutdown***[22] — A method called when a scene shuts down (i.e. you switch to another scene from the current one). You could consider this a "garbage collection" routine that cleans up orphaned game objects.

***Warning:*** Remember JavaScript is a prototype-based language. If you try to store these functions as objects or as arrays, mutating any member of the object or array will mutate the member for every instance that shares the prototype. In order to preserve instance safety, you need to make a copy of the scene.[23]

[21] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/SceneManager.js#L571
[22] https://github.com/photonstorm/phaser/blob/v3.24.1/src/scene/SceneManager.js#L1203
[23] https://medium.com/javascript-scene/3-different-kinds-of-prototypal-inheritance-es6-edition-32d777fa16c9#.iy7efb917

## 3.4 Game Phases as Modules




**Game Flow Phases with Defined JS Modules**

We'll follow this chart in the order of appearance. The only phases we need to revisit, ***tweak***[24] and re-validate might be the "main menu" with new business drivers or the "language menu" when new languages are added. Here's a ***game programming flow chart***[25] located in your bonus download content it demonstrates the same concepts as pictured above. ***Notice it has nothing to do with any Phaser version we're using!! This applies to ANY JS Gaming Framework.***

***https://makingbrowsergames.com/starterkits/_GameFlowChart.pdf;***

***NOTE:*** Some Game Distribution Channels will reject your game if you ***use any text.*** They require only universally understood symbols.

### "Phaser.Game" — One File to Rule them all …

***QUOTE:***[26] "Ash nazg durbatulûk, ash nazg gimbatul, ash nazg thrakatulûk, agh burzum-ishi krimpatul"
(***Note:*** Do NOT utter these words aloud … you've been warned!)

[24] https://www.urbandictionary.com/define.php?term=tweak
[25] https://makingbrowsergames.com/starterkits/_GameFlowChart.pdf
[26] https://www.youtube.com/watch?v=lMSLM33PQDM

### Yes, Gandalf got it wrong!

The actual literal translation from the ***Grimoire — "Lore of Phaser v3.x.x"***[a] is:
*One **File** to rule them all **— (`Phaser.Game` the God-class!**[b]**),***
*One **File** to find them — **(`Phaser.Boot`)**[c],*
*One **File** to bring them all — **(`Phaser.Load`)**[d],*
*and in the darkness bind them! — **(`Phaser.Scenes`)**[e]*

[a] https://en.wikipedia.org/wiki/Grimoire
[b] https://photonstorm.github.io/phaser3-docs/Phaser.Game.html
[c] https://photonstorm.github.io/phaser3-docs/Phaser.Core.Events.html#event:BOOT
[d] https://photonstorm.github.io/phaser3-docs/Phaser.Loader.html
[e] https://photonstorm.github.io/phaser3-docs/Phaser.Scenes.SceneManager.html

## Main.js (aka "launch" or index.js)

Let's review each JavaScript file in the skeleton header. The ***main.js (aka "launch.js" or index.js):***[27] file "IS" the game's genre's foundation. It contains all the particular data, rules, and logical configurations for our game(s). If we build a similar genre, this file should be relatively ***D.R.Y*** It is the first external JS game file loaded, immediately after the Phaser III framework, in the "index.html". I take a "less formal" approach than before inside the mobile version. As explained earlier for mobile single web page applications (SWPA), I insert the entire raw `main.js` script ***as an inlined `script` tag.***

***Exercise:*** Review the source code; it is thoroughly annotated and documented to reduce the price of this book.

***https://makingbrowsergames.com/p3gp-book/_p3-demos/js/mainp3.pdf***

***Hint:*** Review the two demonstration games at https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/ and open the browser console to watch the Phaser Game Framework in the Developer Console. If you've never used the Developer's Console, please read ***How to Run JavaScript Code***[28].

[27] https://makingbrowsergames.com/p3gp-book/index3.html#9.1
[28] https://fireship.io/courses/javascript/beginner-js-survival-guide/

## Boot.js

Our ***boot.js: (aka "initialize")***[29] phase was launched from the game's "index.html" page. This game phase has the responsibility of configuring and setting-up the ***HTML5 <canvas>,*** and game physics. As its name suggests, its purpose prepares the web browser and sets the game dimensions — loading various game assets and storing them in the **Phaser "`cache`",** having them readily available when needed throughout the game. Once the ***<canvas>*** is prepared, it will typically hand-off control toward the next phase called the "load" phase. You can "control" its behavior from within the ***"`config`"*** object — which we placed inside either the ***"`index.html`"*** or ***"`main.js`"*** file.

Modification to this file should be minimal as long as you follow ***a standard naming convention across all your games.*** Loading the standard game phase menus, images, and buttons are already listed. ***There should not be anything in this file you need to modify nor change.*** "Why's that?", you ask! Because if you maintained the same consistent naming conventions for your new artwork and graphics as presented in the `boot.js` file, everything just overwrites the former artwork. Do you remember reading earlier:

> ***"If we create new graphics files, but call them by the same names we have in our game shell. We are simply replacing (i.e., over-writing!) the game art with new art (with the same file names) and VOILA! NEW GAME ... same mechanics, same source code, yet with different "look & feel" — this is the secret sauce for cranking out a game per week."***

### *Typical Boot internal functions:*

- `initialize` function — prepares critical variables for game usage
- `preload` function — manages downloaded game assets.
- `create` function — manages the game re-size (min and max), alignment, and input.
- `enterIncorrectOrientation` function — notify gamer
- `leaveIncorrectOrientation` function — adjust game

***Note:*** You can download this "11-page example file" from ***https://makingbrowsergames.com/p3gp-book/_p3-demos/js/state/boot.js***

[29] https://makingbrowsergames.com/p3gp-book/index3.html#9.2

***Exercise:*** Review the source code above

***Hint:*** To further preserve this file's integrity **and keep it D.R.Y**, you might consider having the "boot.js" simply upload a JSON data file ( aka ***"Asset Pack File"***[30] and is a built-in feature of the ***Phaser Editor 2D***[31] ) with all the game resource unique to this project. This is the way Phaser Editor 2D performs.

## Preload.js

This Game Phase manages our listed game assets into a queue for parallel downloading. It will place the game assets into the ***Phaser "`cache`"*** for immediate use across all game phases. You should optimize this process with the fewest possible downloads that are immediately required by your game. In a normal CMS game (discussed later), I "inline" the normal ***"`boot.js`"*** into the ***"`index.html`"*** and consolidated everything else into a ***"`play.js`";*** by doing so, I have deferred several potential downloaded files with this single combined file.

Many developers use ***Browserify***[32] to the same effect. The formal and separate ***"`preload.js`"*** now becomes a simple JavaScript object in a "single web page application" (SWPA)

***Note:*** You can download an example "1-page file" from ***https://makingbrowsergames.com/p3gp-book/_p3-demos/js/state/load.js***

## Deeper Dive: Artwork & Resources Security

The most exciting thing is the game assets protection now available by using the ***"set Base URL" feature in Phaser III.*** You should see inside the ***"`preload`" essential function*** a statement such as

[30] https://photonstorm.github.io/phaser3-docs/Phaser.Loader.LoaderPlugin.html#pack__anchor
[31] https://help.phasereditor2d.com/v3/asset-pack-editor/asset-pack-file.html
[32] http://browserify.org/

```
this.load.setBaseURL('your domain')
```

When you run this statement all the assets are loaded directly from your domain's designated directories. This is asset security at its ***BEST!*** You maintain control over all your gaming artwork while your games are in the "wild". Simply modifying or updating your game assets dynamically on your website, updates all your clients world-wide. Furthermore, it permits updates to your artwork and the client gamers will get those updates directly from your centralized source, or through your CDN. Naturally, you must have ***Cross-Origin Resource Sharing (CORS)***[33] enabled.

## Deeper Dive: Phaser *Cache*

As soon as the game boots, a global game-wide ***"cache"*** is created. This cache is the gatekeeper to the various subordinate caches created for each game asset and resource. For example, you could access any text by simply using ***"cache.text".*** Here's an example of the resource caches created after booting.

**Phaser Cache for various resources.**

```
1     this.binary       = new BaseCache();
2     this.bitmapFont   = new BaseCache();
3     this.json         = new BaseCache();
4     this.physics      = new BaseCache();
5     this.shader       = new BaseCache();
6     this.sound        = new BaseCache();
7     this.text         = new BaseCache();
8     this.tilemap      = new BaseCache();
9     this.video        = new BaseCache();
10    this.xml          = new BaseCache();
```

You can manage your cached content using common access methods such as ***".add", ".has", ".get", or even ".remove";*** you will use string-based keys with these methods to designate which resource.

[33] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

### Working with the Phaser Cache:

Paraphrased from
http://www.html5gamedevs.com/topic/5683-add-bitmapdata-to-cache-as-image/

Phaser has one single cache in which it stores all assets.

The cache is split up into storage compartment (aka sections; such as images, sounds, video, JSON, etc). All assets are stored using a unique string-based name (e.g.: its index `key`) as their unique identifier and path locations. Assets, stored in different areas of the cache, could use the same key indexing names. For example, `playerWalking` could be used as a key for both a sprite sheet and an audio file, since they are unique and different data files, stored in separate sections areas.

The cache is automatically populated by the `Phaser.Loader` state. When you use this loader to pull external assets — such as images, they are automatically placed into their respective cache sections.

You can access the Phaser cache from inside any State using `this.cache`. You can pull any public method from the cache.

Normally, the cache will return a `reference` handle to items stored. This means, that when you retrieve an item and then modify it, the original item in the cache is modified too. The stored item is passed by its handle reference.

By default, when you change States, the cache is ***not cleared.*** However, there is an option to clear the cache if you require it. In a typical game, during the `boot` and `pre-load` states use the cache as storage.

***Note: Tiled***[34] is a ***free software package*** designed specifically for creating "tiled maps". Another ***licensed application*** is ***Texture Packer***[35] that will also help you create sprite sheets and their atlas. Texture Packer can also create "tile maps". ***Shoebox***[36] is similar to Texture Packer except that it is ***FREE.***

[34] http://www.mapeditor.org
[35] https://www.codeandweb.com/texturepacker
[36] http://renderhjs.net/shoebox/

## Deeper Dive: Loader Examples

Each "Phaser III Scene" is responsible for loading its own resources and gaming assets when it starts. Scene loading runs in parallel; meaning that a scene will load its resources even if another scene is currently loading.

The ***"`BaseLoader`"*** class governs this loading process. It is responsible for the "queue management", "dispatching events", and "load management". The ***"`BaseLoader`"*** class handles the following file types using its ***"`addfile`"*** method:

- Animation JSON File
- Atlas JSON File
- Binary File
- Bitmap Font File
- GLSL File
- HTML File
- Image File
- JSON File
- SpriteSheets
- SVG File
- Text File
- XML File

***Hint:*** Each Scene can further use ***"`this.load.image`", "`this.load.json`", and "`this.load.atlas`".*** You can also pass configuration objects to these methods.

***Warning:*** ***"`load.path`" and "`load.baseURL`"*** are acknowledged when "relative" paths are used. Absolute URL — those starting with ***"`http`" or "`\\`"*** (ed.: not recommended) — are ignored.

**Samples from** `phaser.io/phaser/api/loader`

```
    // Original image loader signature:
    this.load.image('bunny', 'assets/sprites/bunny.png');

    // Object based
    this.load.image({ key: 'bunny', texture: 'assets/sprites/bunny.png' });

    // Allows for arrays of objects
    this.load.image([
        { key: 'bunny', texture: 'assets/sprites/bunny.png' },
        { key: 'atari', texture: 'assets/sprites/atari400.png' },
        { key:  'logo', texture: 'assets/sprites/phaser2.png' }
    ]);

    // Object based including XHR Settings
    this.load.image({
        key: 'bunny',
        texture: 'assets/sprites/bunny.png',
        //ext: 'jpg',       // png is the default
        xhr: {
            user: 'root',
            password: 'th3G1bs0n',
            timeout: 30,
            header: 'Content-Type',
            headerValue: 'text/xml'
        }
    });

    // Auto-filename based on key:
    //  Will load bunny.png from the defined path,
    //  because '.png' is the default extension.
    this.load.image({ key: 'bunny' });

    // Will load bunny.jpg from the defined path,
    //  because of the 'ext' property.
    this.load.image({ key: 'bunny', ext: 'jpg' });

    //  ----------------------
    //  Texture Atlas Examples
    //  ----------------------

    //  Original atlas loader signature:
```

```
//  this.load.atlas(
            key,
            textureURL,
            atlasURL,
            textureXhrSettings,
            atlasXhrSettings)

this.load.atlas('level1', 'assets/level1/items.png',
    'assets/level1/items.json');

// Object based
this.load.atlas({
        key:        'level1',
        texture:    'assets/level1/items.png',
        data:       'assets/level1/items.json' });
```

**Preload JSON Samples**

```
function preload() {
    this.load.json('jsonData', 'assets/atlas/megaset-0.json');
}

function create() {
    console.log(this.cache.json.get('jsonData'));
}
```

## Splash.js or Language.js?

We arrive at our ***splash.js or language.js***[37] phase; it will adapt text information variables to the gamer's chosen language. It is one of the first "stopping points" after the "network cloud access" (hopefully within 20 seconds?). If our game takes longer than 20 seconds to activate, it stands rejection from most "app stores". Here is an excellent place to inform our gamers about our sponsorship, provide advertisements(?? See Phaser Game Design chapter 1!), offer language selections, and present your own logo or "White Label" branding. ***I prefer to offer my gamers a "language screen" instead of an initial "splash screen".*** While the gamer pauses to select their language, it allows our game more time to download more ***appropriately targeted*** game resources ***or (better yet!) launch a web socket connection.*** This phase further allows me to set the mood/theme music, provide a background narrative in their native language. The

[37] https://makingbrowsergames.com/p3gp-book/index3.html#9.4

more standardized these screens are … the better! — resulting in a savings of our development time.

What I do, when presenting a "language menu" to my gamers, is to let them select — ***dynamically on-demand!!*** — their native language for continued game-play and interaction. I set ***a global variable***[38] to their "language index" and attach their language JSON file to all the game's "text variable". I'll not download *every language lexicon known to mankind* at this point — just their language lexicon; and, of course, ***we should steer clear of "Enochian"***[39] … we'll have none of that here; they can go "somewhere else and play".

The natural choice for "language selection" is a visual button mechanism designed around the gamer's national flag — ***"iconic symbols" ARE the universal (international) language.*** When our gamer glides over any nation's flag, the "tool-tip text" changes into that nation's predominant language. Mesmerized by the sudden display of various languages and spellings, our gamers **— doing what they do best (i.e.: "play") —** might spend, perhaps, a whole 3-seconds goofing around, thus providing us more time for downloading game resources ***through perhaps a newly activated web socket (click here for a demonstration from your "Bonus Content")***[40]***.*** Naturally, there must be a different method to handle mobile touch input. Clever as our gamers are, they will "select-click" the flag button representing their native language as their visual clue. On that ***"click-event",*** the internal game functions will send a request to download that specific ***"JSON"*** language file and dynamically populate (e.g.: substitute) all text variables inside our game to their language content. Read some interesting facts about the Internet and ***who your "real" target audience is becoming!***[41]

***Note:*** More about international targeting in the Phaser Game Design Workbook.

[38] https://www.w3docs.com/learn-javascript/variable-scope.html
[39] https://en.wikipedia.org/wiki/Enochian
[40] https://makingbrowsergames.com/p3gp-book/standalone/index.html
[41] http://www.internetworldstats.com/stats.htm


**Adventurers of Renown: The Ruins of Able-Wyvern™**

Live language demonstrations: ***Ruins of Able-Wyvern™ (as pictured above)***[42]

For our Mobile "SWPA" or "PWA", we'll use another **"`<div>`"** tag inside the "index.html" file. Review the mobile "index.html" source code, and you find that the "splash scene" and "language menu" are mere **"`<div>`"** tags using Bootstrap CSS!

Modifications for the ***splash.js or language.js***[43] should be minimal as long as you use a consistent information and ***menuing system***[44] across all your games. Loading standard game phase menus, images, buttons are already inside your ***Bonus Content/standalone***[45]. There should not be anything in these files you would need to modify nor change … unless we have a new sponsorship or perhaps adding a new targeted language.

***Exercise:*** Review the source code from ***language.js.***[46]

[42] https://makingbrowsergames.com/p3gp-book/_p3-arrav15/index.html
[43] https://makingbrowsergames.com/p3gp-book/index3.html#9.4
[44] https://en.wikipedia.org/wiki/Menu_(computing)
[45] https://makingbrowsergames.com/p3gp-book/standalone.zip
[46] https://makingbrowsergames.com/p3gp-book/_p3-demos/js/state/language.js

## Main Menu.js

Our next phase is the game's "main menu" — the central hub of all the game's activity. The more standardized this screen is … the better. (***NOTE:*** Again, this is another *"stopping point"*) It is during this game phase our loaded "language text" kicks-in for all tool-tips, feedback, and menus.

| Reference | |
|---|---|
| 1 | ***`preload`*** function — not used; everything was downloaded in the ***`boot.js`*** |
| 2 | ***`create`*** function — links downloaded assets for use during the game. |
| 3 | ***`beginGame`*** function — manages theme music |
| 4 | ***`gameCredits`*** function — manages theme music and game author information |
| 5 | ***`MoreGame`*** function - manages theme music, and provides access to more games from the author |

**NOTE: You can download an example file from**

***https://makingbrowsergames.com/p3gp-book/_p3-demos/js/state/menu.js***

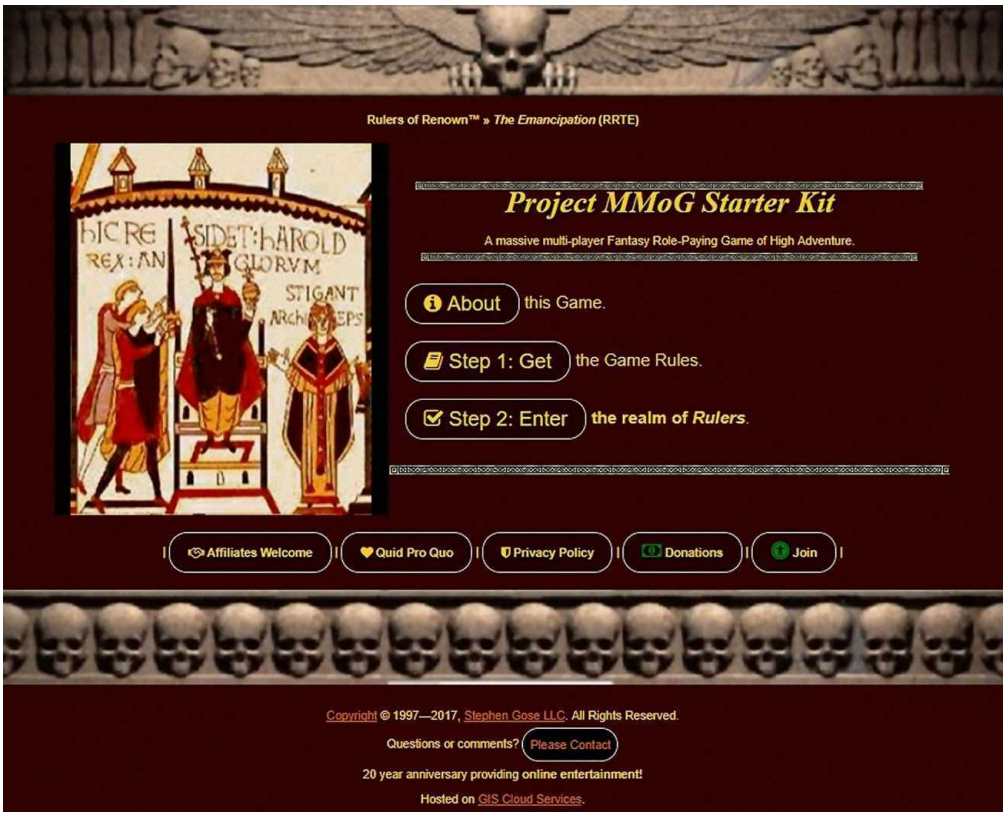
***Warning:*** Some Game Distribution Channels restrict the number of "out-going links".

On the main menu, you should offer your gamers several options before starting their game's play in earnest. The following scripts are not included and would be handled better as separate HTML web pages ***from a Content Management System (CMS)*** rather than trying to stuff everything inside a single game ***"`canvas`" tag.*** That would merely bloat our final gaming file. Remember the HTML5 ***"`canvas`" tag*** is merely a graphical display; in essence, it replaces the former Adobe Flash plugin. (***NOTE:*** End of life for Flash plugins is 20201231) Visit **other games on this book's website** for examples of a **Content Management System (CMS).**

- ***Rulers of Renown™: The Emancipation***[47]
- ***Adventurers of Renown™: The Ruins of Able-Wyvern***[48]
- ***Adventurers of Renown™: The Blood Pit***[49]

[47] https://www.adventurers-of-renown.com/quests/rrte.php
[48] https://www.adventurers-of-renown.com/quests/arra.php
[49] https://www.adventurers-of-renown.com/quests/arbp.php

- ***Adventurers of Renown™: The Rogue Prince***[50]
- ***Adventurers of Renown™: The Rescue of NCC Pandora***[51]




**Simple CMS or PWA "game-shell"**

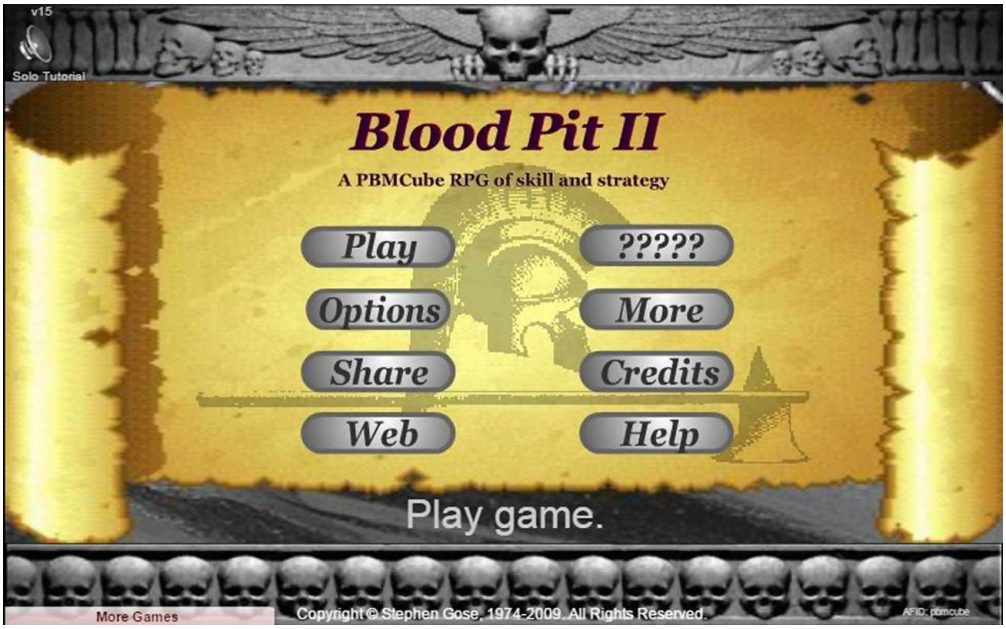
Typical pages within a Content Management System (CMS) are ***not directly related to the actual game mechanics nor gameplay.*** These pages enhance or support several business aspects surrounding our game and develop customer loyalty and a sense of community. So, why should we bloat ***— and clutter —*** our Phaser game with ***superfluous information.***[52] Example of such candidate pages placed ***outside the Phase Game "`canvas`"*** are:

- ***"About.js"*** — a page biography to enhance your portfolio and resume. In our mobile demonstration, the "about page" is used to enhance SEO and page content since it is simply another ***"`<div>`"*** tag.
- ***"Credits.js"*** — a page giving attributions.
- ***"Donations.js"*** — a crowd-funding page requesting financial support.
- ***"Instructions.js"*** or ***"help.js"*** — a page offering helpful hints, walk-throughs, achievement, awards, entitlements, or game rules. If the game instructions are

[50] https://www.adventurers-of-renown.com/quests/arrp.php
[51] https://www.adventurers-of-renown.com/quests/arnp.php
[52] https://www.merriam-webster.com/dictionary/superfluous

minimal in content, it could be combined with another page as typically seen in various “splash pages”.

- ***“Language.js”*** — a page offering gameplay in their native language. It downloads a specific language “dictionary” and populates all text displays and HUD with their native language content.
- ***“More-Games.js”*** — a redirection page to your whole collection of games; used to build a loyal fan base. ***This is the #1 Marketing Tip from various successful game indies.***
- ***“Options.js”*** — a configuration page used to set keyboard, input, and the like. A live example at ***https://www.adventurers-of-renown.com/quests/arra.php/welcome/lobby.html***
- ***“Scores.js”*** — pulls from a master database (back-end) of recorded scores. It’s also possible to simply use the browser ***“`localStorage`”.***
- ***“Share.js”*** — a page to enhance the viral distribution of your game or announcements within the game. See the **twitter enhancement here**[53].
- ***“Submit-Scores.js”*** — collects and transmits the current game session for permanent storage either locally and/or remotely.
  - ***“wins.js”*** — records information into the gamer’s registered account.
  - ***“loose.js”*** — records information into the gamer’s registered account.
- ***“Webmasters.js”*** — a page offering license and distribution information.




**Adventurers of Renown™: “The Blood Pit” (Main Menu)**

[53] https://makingbrowsergames.com/p3gp-book/_p3twitterEnhancementJS.pdf

## Play.js

Finally, we arrive at what this book is all about ***— the "Play" game phase*** (aka "game event loop" or "life cycle")! This phase dives straight into creating our gameboard **entities and components as a browser display.** It is here that our "Gaming Framework Mechanisms" appear — the User Interface controls, head-up-display (HUD), and supporting function about "how" the game is displayed. We should be able to exchange this file with a different "Gaming Framework Mechanisms" file, of the same genre, and have a visually new game.

**NOTE: You can refer to the "Skeleton Game Phase". Download from**

***https://makingbrowsergames.com/p3gp-book/_p3-demos/js/state/play.js***

## Deeper Dive: JS Modules

As you have seen, separating our code into these various files is a very good practice during the first developmental stages of our game. It also gives us more focus on those immediate game actions and logic driving our game while we prototype. Most importantly, we can re-use our source code ***— mix, match, and arrange our "ROSES"!***

Nearly every programming language has the concept of *modules* — a way to include code written in one file and insert it inside another file. Senior programmers have used external *coded libraries* for inclusion into their projects for over half a century! JavaScript did not originally include *modules* until ECMAScript 6 at the end of July 2014. Until that time, the JavaScript developers' community invented clever work-rounds. Perhaps, you may have heard of:

- ***CommonJS Modules:*** — The dominant use of this standard is found in Node.js (***NOTE: Node.js modules***[54] have a few features that go beyond CommonJS). CommonJS has several characteristics such as:
  - Compact syntax
  - Designed for synchronous loading
  - Primarily used on the server-side
- ***Asynchronous Module Definition (AMD):*** — The most popular use of this standard is `RequireJS`. AMD has these characteristics:
  - Slightly more complicated syntax, enabling AMD to work without `eval()` (or a compilation step).
  - Designed for asynchronous loading
  - Primarily used on the client-side browser

[54] https://www.openmymind.net/2012/2/3/Node-Require-and-Exports/

***Note:*** These generalized features are simply an overview. You can dive deeper into these formatted modules from ***"Writing Modular JavaScript With AMD, CommonJS & ES Harmony"***[55] by Addy Osmani.

There are many reasons to use "JavaScript modular files" while coding your game project. Since ES6 now includes *modules*, you can ***go beyond the CommonJS and AMD capabilities.*** If designed properly, these "JavaScript modular files" help in the game's portability, and its "reusable chunks of code" (i.e., game prototypes and components).

**Browser JavaScript modules versus inline scripts:**

| Elements | Modules | Scripts |
|---|---|---|
| Default mode | strict | non-strict |
| Execute sync/asyn imports | yes | no |
| File extension | .js | .js |
| HTML5 tag | <script type="module"> | <script> |
| Programmatic (Promise API) | yes | yes |
| Top-Level Value of "this" | undefined | window |
| Top-Level Variables are | ***local to module***[56] | global |

***Note:*** If you're not familiar with "JavaScript modular files", read this chapter about ***"Modules"***[57] from the ***FREE online book "Eloquent JavaScript" by Marijn Haverbeke***[58] or you might review this superior article ***by Preethi Kasireddy's.***[59]

***QUOTE from Phaser III Game Design Workbook***[a] "Development source code is what you read and write, and "check-in" to your source control system such as ***GitHub.***[b] It should be highly modular (i.e., split across many files), extensively commented, and should make liberal use of white-space to indicate formatting structure. On the other hand, Machine code is what gets served up to a browser. It should consist of a small number of merged files and should be stripped of any developer's comments and unnecessary white-space. Your "build" process — refer to ***Google Developer: Setup Your Build Tools***[c] — is a step in which you apply these transformations; many developers use the automated ***"Grunt".***[d]
Finally, your web server should deliver the machine code with `gzip` compression for extra speediness." Read more tips ***here.***[e]

[a] https://leanpub.com/phaser3gamedesignworkbook

[55] http://addyosmani.com/writing-modular-js/
[56] https://www.openmymind.net/2012/2/3/Node-Require-and-Exports/
[57] https://eloquentjavascript.net/10_modules.html
[58] https://eloquentjavascript.net/index.html
[59] https://medium.freecodecamp.org/javascript-modules-a-beginner-s-guide-783f7d7a5fcc

[b] https://github.com/PBMCube
[c] https://developers.google.com/web/tools/setup/setup-buildtools
[d] https://24ways.org/2013/grunt-is-not-weird-and-hard/
[e] https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/

## 3.5 Step #1 of 4: Generate Game Phases

Now that our game's "index" page is in place and loads our ***Phaser Gaming Framework — or any JavaScript Gaming Framework for that matter —*** we'll turn our efforts toward our core gaming code and then the ***"Gaming Loop's `event logic`".***

We also have a couple of choices in this construction. We can build either a ***single web-page game*** or a full-blown ***"Content Management System (CMS)" game shell.*** Examples of ***single-page games*** drop the gamer directly into the "play phase" with little warnings. This is typical of most games you find. Examples are:

- Our ***Breakout sample game***[60] we started in Chapter 1 — and will continue referring to it throughout this book.
- ***All the Official Phaser III Games examples***[61]

**Example 3.1: Creating Game Phase (traditional `object literal` method)**

```
123     // ============
124     // Examples 3.1 to 3.19: Creating Game Phase (traditional method)
125     //   This is an Anti-pattern: polluting the global namespace.
126     // ============
127     // Step #3) new game state additions:
128     // -------------
129     // Notice: This could be placed into a separate module file.
130
131     var main = {
132        // Essential Functions found inside this state.
133        // Phaser v2.x.x called this "init"
134        initialize: function(){
135             //stuff to generate this function
136             // debug header information
```

[60] https://makingbrowsergames.com/p3devcourse/standard/lesson15.html
[61] https://labs.phaser.io/index.html?dir=games/&q=

```
137         },
138
139         create: function() {
140              // ============
141              // Example 2.6: Additional Phaser Properties begins
142              // ============
143              console.log("mainState Ready!");
144                 //stuff to generate for this scene.
145         }, //the comma is very important.
146
147         update: function() {
148              //frame refresh and display updates
149         }
150     }; //the semi-colon is very important.
```

***Exercise:*** The example above refers to: **https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js**

## Dynamically Including Game Phases

**Example 3.1a: Creating Game Phases from Dynamically loaded files**

```
198     // ====================================
199     // Step #1) Let's tell Phaser about our new phase
200     // ============
201     //Phaser uses our code and gives it a name of 'main'.
202     main: function main(){
203
204         this.game = new Phaser.Game(config);
205
206         // This is the SECRET SAUCE!!
207         //    add all game phases into Phaser III scenes.
208         for(var stateName in window.GAMEAPP.state){
209             console.log("Creating Crnt State: "+stateName);
210             this.game.scene.add(
211                     stateName,
212                     window.GAMEAPP.state[stateName]);
213
```

```
214         } //End For Loop
215
216         //using v3? use this manual start method below.
217         console.log("Leaving GAMEAPP.main -> boot"); //debug
218         //tells Phase to start using it.
219         this.game.scene.start('boot');
220         // Example 3.7: ends
221         // ============
222     } //End Main,
223     // ===================================
```

***Note:*** Review complete example in the ***Source code Appendix.***[62]

**Example 3.1b: Creating Skeleton Game Phase - per Phaser Labs**

```
    // dozens of ways to launch Phaser III Scenes?
    // pick one and be consistent.
    // Refer to: http://labs.phaser.io/index.html?dir=scenes/&q=
    var <GamePhaseName> = new Phaser.Class({

        Extends: Phaser.Scene,

        initialize: function <GamePhaseName> (config) {
            Phaser.Scene.call(this, { key: '<GamePhaseName>', config });
        },

        init: function (data) {},

        preload: function () {
            this.load.image('<GamePhaseNameBackGround>',
                            'assets/images/<GamePhaseName>.png');
        },

        create: function (data) {
            this.add.image(0, 0, '<GamePhaseNameBackGround>').setOrigin(0);
            this.input.once('pointerdown', function () {
                console.log('From <GamePhaseName> to <NextPhaseName>');
                this.scene.start('<NextPhaseName>');
            }, this);
```

[62] https://makingbrowsergames.com/book/index12.html#12.3

```
        },

        update: function (time, delta) {}

    });
```

## Deeper Dive: D.R.Y. Stand-alone

**Sample: Game Launch in `game.js` with Name Space - D.R.Y. method**

```
203     // ====================================
204     // ----------------------------------------
205     // Main game Handler methods
206     // ----------------------------------------
207     //**TODO**
208     //  re-factor and adjust for your game deployment
209     //  remove console debug information BEFORE public deployment
210     // ====================================
211     // Step #1) Let's tell Phaser about our new phase
212     // ============
213     //Phaser uses our code and gives it a name of 'main'.
214     main: function main(){
215
216         this.game = new Phaser.Game(config);
217
218         // add all game phases into Phaser v3.x.x scenes.
219         for(var stateName in window.GAMEAPP.state){
220             console.log("Creating Crnt State: "+stateName);
221             this.game.scene.add(
222                     stateName,
223                     window.GAMEAPP.state[stateName]
224             );
225         }
226
227         //v3  manual start method below.
228         console.log("Leaving GAMEAPP.main -> boot");    //debug
229         //tells Phase to start using it.
230         game.scene.start('boot');
231         // Example 3.7: ends
232         // ============
233
```

```
234     },
235     // ====================================
236
237     /**
238     // main function - using Object.create experiment!
239     main: function(){
240        this.game = Object.create(Phaser).Game(
241             this.viewportWidth,
242             this.viewportHeight,
243             Phaser.AUTO,
244             document.body,
245             window.GAMEAPP.state.boot);
246     },
247     */
248
249     // here we will store all game phase/states
250     //  state object filled as js files load.
251     state: {},
252     // ====================================
```

You'll recall that I said earlier, "I do not place my game scenes inside the ***"`config`"*** object. You can see how I inform Phaser about my game's phases in ***"main.pdf" lines 119 to 126***[63]. I add the game scenes directly into my game instance. Then on Line 125, I tell Phaser to move to my initial `boot` scene. "

I follow this method so that, whatever game phases I'm using, they will be ***automatically identified and loaded. I can now pick and choose which game phase files to load from one place — its "`index.html`" — and those phases will appear in my game without touching any code.***

***Exercise:*** The example above refers to Lines 203 to 242: **https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js**

## Step #3 of 4: Game Phase Transitions

***Exercises:*** Review DevLog #120 https://phaser.io/phaser3/devlog/120

[63] https://makingbrowsergames.com/p3gp-book/mainp3.pdf

**Sample from v3.5.0**

```
    this.scene.transition({
        //allowInput: false, // set true to enable input system
                             // of current scene and target scene
        data: {x: x, y: y},  // an object containing any data you wish
                             //passed into target scene init. methods.
        duration: 1000,      // in milliseconds
        //moveAbove: true,   // move the target Scene above this current
                             // scene before the transition starts
        //moveBelow: true,   // move the target Scene below this current
                             // scene before the transition starts
        //onUpdate: null,
        //onUpdateScope: scene,
        //sleep: false,      // set true to sleep this scene,
                             // set false to stop this scene
        target: 'nextScene'  //, the scene key name to transition into
    });
```

## Deeper Dive: The CMS "Game Shell"

A **Content Management System (CMS)** is "game shell" that surrounds the "Play Phase" and is merely a simple method toward building a ***Progressive Web App (PWA)***[64]. It reliably and instantly loads the gameplay content, and is similar to what you would see in native mobile applications.

The "game shell" has the minimal required technologies of HTML, CSS, and JS to display the game's browser interface. When it is cached offline, it ensures instant, reliably good performance to gamers in their returning gaming sessions. The network provides the newest or updated gaming content and assets.

For games — "a single web-page application with heavy JavaScript architectures" — using a "game shell" is ***"THE go-to approach".***[65] The "game shell" approach relies on aggressively caching the "shell" content using a ***web-service worker***[66] to get the game up and running. Next, the dynamic game content and artwork loads for each ***game phase.*** The secret sauce that a "game shell" provides is: ***Getting the initial HTML into a user's device and display it without any help from the network connection!***

In other words, the "game shell" is similar to the code you'd publish to any "app store" when building a native mobile app. The "game shell" is a ***skeleton (aka Bare Bones***

[64] https://developers.google.com/web/progressive-web-apps/
[65] https://hbr.org/2016/06/the-go-to-market-approach-startups-need-to-adopt
[66] https://developers.google.com/web/fundamentals/primers/service-worker/

***Prototypes)*** of your game's user interface ("UI") and all those prototype components necessary to launch your game from the ground up ... but doesn't include the game's "data logic."

***Note:*** Try the ***First Progressive Web App (PWA)***[67] to learn how to architect and implement your first generic mobile application shell. The ***"Instant Loading with the App Shell model"***[68] video also walks you through this design pattern.

## Deeper Dive: When to use a game shell

Building a PWA does not mean starting from scratch. If you are building a modern single-page app, then you are probably using something similar to a "game shell" already whether you call it that or not. The details might vary a bit depending upon which gaming libraries or frameworks you are using, but ***the concept itself is framework agnostic!***

The **"game shell"** architecture makes the most sense for ***any game project*** with relatively unchanging navigation but changing internal content — ***the canvas?!*** Many JavaScript Gaming Frameworks and libraries already encourage splitting your game logic from its content (aka "Separation of Concerns"), making this "game shell" design appealing. For certain types of games — that only have static content — you can still follow the same idea but the game's ***"canvas"*** tag becomes 100% of the **"game shell"** — this is what you find in the majority of published **Phaser v2.x.x & v3.15+** games — single-page games described earlier that use a single static ***"canvas" tag*** with all various game ***"roses"*** one is accustom to expect ( aka menus, movieClips, parts, phases, sections, stages, states, scenes, screens, ***thingies, dumaflache,***[69] or ***"Aardvarks!"***[70] ).

***Exercise:*** Let's see how Google builds its typical mobile-app shells. Take a look at ***Building the Google I/O 2016 Progressive Web App.***[71] This real-world mobile app started with a SPA to create a PWA that pre-caches content using a web-service worker, dynamically loads new pages, gracefully transitions between views, and reuses content after the first load.

The benefits of using this "game shell" architecture and a ***web-service worker***[72] are:

[67] https://codelabs.developers.google.com/codelabs/your-first-pwapp/#0
[68] https://www.youtube.com/watch?v=QhUzmR8eZAo
[69] https://www.urbandictionary.com/define.php?term=dumaflache
[70] https://en.wikipedia.org/wiki/Aardvark
[71] https://developers.google.com/web/showcase/2016/iowa2016
[72] https://developers.google.com/web/fundamentals/primers/service-worker/

- Reliable performance that is consistently fast across all your game projects. Repeat visits are extremely quick. Static assets and the UI (e.g. HTML, JavaScript, images, and CSS) are cached on the first visit so that they load instantly on repeat visits. Content may be cached on the first visit, but is typically loaded as needed — "just in time!"
- Native-like interactions. By adopting the game shell model, you can create experiences with instant, native-application-like navigation and interactions, complete with offline support.
- Economical use of game assets. Design for minimal resource usage. Be judicious in what you immediately cache because loading non-essential files (i.e., large images that aren't shown on every page) will result in browsers downloading more game assets than is used immediately. Even though WAN bandwidth is available in western countries, this may not be the case in emerging game markets where connectivity is expensive and data is costly.

## 3.6 Encoding Phaser *Scenes* as a *"Game Phase"*


**Game Phases Reviewed**

Let's review those game phases which compose a "game shell". Since each phase performs a similar task, it is easy to keep them ***D.R.Y. So, once we've written a game phase for our "game shell", we're done! We never have to touch it again — unless you want to muck around with some tweaks***[73] — such as small unique modifications or features we might want to include inside the core "Play" phase. Each game phase could become ***siloed***[74] as a separate JS module file and prototype! We can ***mix, match, and arrange our "rose" bouquet***[75] ***any way we want!*** When we eventually create our final artwork — and assign them the same file names we have inside our "game shell" — we are simply replacing the original ***"block-style graphics"*** with new art (by using the same file names; we are ***intentionally overwriting the "block-style graphics*** ), and thus, **VOILA! NEW GAME** ... same game mechanics, same source code, yet with different "look & feel" **coming from the newly imported artwork**[76] — ***this is the secret sauce for cranking out a game per week!*** (Refer back to chapter 1)

### Vanilla, Chocolate, or Strawberry Creme-filled?

[73] https://www.collinsdictionary.com/us/dictionary/english/muck-around
[74] https://en.oxforddictionaries.com/definition/siloed
[75] https://www.merriam-webster.com/dictionary/bouquet
[76] https://www.gamedevmarket.net/?ally=GVgAVsoJ

**Sample: Step 3) Vanilla Phaser Scene as a function object**

```
    //Step 3) new game state additions as a function
    //  Notice: This could be inside a separate module file.
    //This is a namespace. Replace <Phase_Name>:
    window.GAMEAPP.state.<Phase_Name> = function(game){
        // Phaser v2.x.x called this "init"
        initialize: function(parameters) {
          // any required initialization for this phase?
          //This is the first function called when any Phase State begins
          // and launched prior to preload, create (or anything else).
        }, //comma is very important

        preload: function() {
          // load required resources for this phase
          //  for example:
          this.load.image("preloaderBar", "assets/images/preloader-bar.png");
          this.load.spritesheet("button",
                    "images/buttons/mmm-sprites.png",129,30);
        }, //comma is very important

        /** Creates the sets-up game environment.
        This is called once immediately after the preload function completes.
        If you do not have a preload method then
            create is the first method called after init.
        */
        create: function() {
           //OR create items to display on this game scene.
        }, //comma is very important

        update: function () {
          //used for verification that game assets are available.
        } //no comma here
    };
    // ... .

    var params = ['L1', 'L2'];
    var autoStart = true;
    var sceneConfig = { ... }
    this.scene.start('Phase_Name', sceneConfig, autoStart, params);
    //See notes from:
    //https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scenemanager/
```

## Overriding Essential Functions inside Phaser.Scene

```
    // NOTE: this style could be applied to ANYTHING inside Phaser!
    var demo = new Phaser.Scene('Demo');

    // Phaser v2.x.x called this "init"
    demo.initialize = function initialize (data){ ... };
    demo.preload    = function preload (){ ... };
    demo.create     = function create (data){ ... };
    demo.update     = function update (time, delta){ ... };
```

## Creating Scenes using ES5 Prototypes

```
    var MyGame = { ... };

    MyGame.Boot = function () {
        //stuff boot game phase performs.
    };

    MyGame.Boot.prototype.constructor = MyGame.Boot;

    MyGame.Boot.prototype = {
        // Phaser v2.x.x called this "init"
        initialize: function initialize (data) { ... },
        preload:    function preload () { ... },
        create:     function create (data) { ... },
        update:     function update (time, delta) { ... }
    };
```

***Exercise:*** Using the sample code above, create each of the following game phases for future use. These new files, you are creating, will become our "game container" or "game shell". Refer to the following in the Source Code appendix.[77]

***Exercise:*** Study the various combinations of creating JavaScript ***Object creation patterns***[78] and styles from ***Code reuse patterns***[79]

[77] https://makingbrowsergames.com/p3gp-book/index9.html
[78] https://www.jspatterns.com/category/patterns/object-creation/
[79] https://www.jspatterns.com/category/patterns/code-reuse/

**Sample: Step 3) Chocolate: ES6 Phaser Scene.**

```
    //Generated by Phaser v3 Typescript example
    /**
    * NOTE: the alternate acceptable form for ES6 and TypeScript
    * Classes are NOT hoisted.
    *
    * Alternate syntax per
    * "Professional JavaScript for Web Developers 3rd Edition" pg: 873
    * class <Phaser_State_Name> prototype Phaser.Scene {
    */

    class <Phaser_State_Name> extends Phaser.Scene {
        constructor() {
           //super(); if needed.
           //only properties are allowed here per MDN
        }

        initialize() {
        }

        preload () {
        }

        create() {
        }

        // state-methods-begin
        // user code here
        // state-methods-end

    }
    // end generated code
    // user code here
```

***Quote from Wikipedia:***[80] "TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict **syntactical superset of JavaScript and adds optional static typing to the language.** TypeScript is designed for the development of large applications and ***transcompiles to JavaScript (ed.: ES6!).***"[81] Per ***this website***[82] and after 6 years from the initial release, only 24% of web developers use Typescript.

## Creating Scenes using Phaser.Class

Using the ***`Phaser.Class`*** is an interesting option. If you study the Phaser source code[83] — starting from line 171, it is simply using the **ES5 "`.apply`" method.** The ***"`apply`"*** method is similar to the ***"`call`"*** method; the only difference is that ***"`apply`"*** takes arguments in an array. "Phaser.Class" creates a new class with the given descriptor; the normal OOP constructor is labeled by the name ***"`initialize`"*** and is an optional function. If unspecified (i.e., using "init" instead of "initialize" an anonymous function will be used which will call the parent class.

## Creating Scenes by extending Phaser.Class

```
1     var MyScene = new Phaser.Class({
2
3         Extends: Phaser.Scene,
4
5         initialize: function MyScene (config) {
6             Phaser.Scene.call(this, config)
7             // add more internal variables as needed.
8         },
9
10        // typical "Phaser Essential Functions" for this scene.
11        initialize: function initialize (data) { ... },
12        preload:    function preload () { ... },
13        create:     function create (data) { ... },
14        update:     function update (time, delta) { ... }
15    });
```

[80] https://en.wikipedia.org/wiki/TypeScript
[81] https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/
[82] https://www.jetbrains.com/research/devecosystem-2018/javascript/
[83] https://github.com/photonstorm/phaser/blob/v3.24.1/src/utils/Class.js#L171

## ES6 Considerations: "Strawberry"

**Sample: Step 3) ES6 Game**

```
import Boot from 'js/states/boot';
import Preload from 'js/states/preload';
import Main from 'js/states/main';
import GameOver from 'js/states/gameOver';

* Alternate syntax per
* "Professional JavaScript for Web Developers 3rd Edition" pg: 873
* class <Phaser_State_Name> prototype Phaser.Game {
*/
class GAMEAPP extends Phaser.Game {
    constructor() {
        super('100%', '100%', Phaser.AUTO, 'gContent');
        //scene.add (key, sceneConfig, autoStart, data)
        this.scene.add('boot', boot, false);
        this.scene.add('preload', preload, false);
        this.scene.add('main', main, false);
        this.scene.add('gameOver', gameOver, false);
        this.scene.start('boot');
    }
}
new Game();
```

***Hint to REMEMBER:*** ES6 classes in JavaScript are not blueprints as found in other Object-Oriented (OO) languages. They are simply "defined objects" that are modified "at-will" during run-time. Refer to ***A prototype-based language***[84]

Compare the "2016 ES6 format" to the TypeScript format. They are similar because they are transpilers. They translate down into acceptable ES5 JavaScript code for older browsers used today. I anticipate current browsers to uphold the ***ES6 to ES9 standards.***[85] This means that all your game prototypes should be compatible with the current ES5 JavaScript standards; you must "future proof" your efforts — make your code simple and correct; then make it fast and small, but only if necessary. Here's an ***example from this article.***[86]

[84] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes
[85] https://kangax.github.io/compat-table/es6/
[86] http://brianchang.info/2016/01/23/how-to-future-proof-your-code.html

***Hint:*** The current JavaScript standard is ***ES10***[87] (as of FEB 2019).

## Creating Scene Configuration files

Each scene in Phaser III can load its unique configurations ***— even from remote JSON files.*** Here's a standard sets of configuration options:

```
    var config = {
        key: '',
        // active: false,
        // visible: true,
        // pack: false,     //see JSON file pack below
        // cameras: null,
        // map: {},
        // physics: {},
        // loader: {},
        // plugins: false,
        // input: {}
    };

    // JSON file pack example
    // use: scene.load.pack(key, url, dataKey);

    {
        'dataKey': {
            // "prefix": "...",         // optional, extend key by prefix
            // "path": "...",           // optional, extend url by path
            // "defaultType": "image",  // optional, default file type
            'files': [
                {
                    'type': 'image',
                    'key': '...',
                    'url': '...'
                },
                {
                    'type': 'image',
                    'key': '...',
                    'url': '...'
```

[87] https://www.w3schools.io/javascript/es10-features/

```
                }
                // ...
            ]
        },

        'node0': {
            'node1': {
                'node2': {
                    'files': [
                        // ....
                    ]
                }
            }
        }
        // dataKey: 'node0.node1.node2'
    }
```

## Deeper Dive: Defining Other Scene Properties

We can provide other basic properties in our Phaser III game by describing more settings. For example, we might set a background color or we could set up a "physics reaction system" that will define how game objects interact with each other. In the former Phaser v2.x.x, Game Objects (such as Sprites) could only belong to 1 physics system at a time, but you can have multiple physics systems active within a single v2.x.x game canvas. ***"In v3.16+, all this has changed! A Scene can only have 1 physics system running at once, never more than this.*** The difference is that in v3.x.x you can have ***multiple Scenes and they each could have their own physics system (if required). (ed.: Scenes, in v3.x.x, are sub-sections of a single game phase all running in parallel. Their new behavior reminds me of Adobe Flash MovieClips on the timeline. Refer to the Phaser III Game Design Workbook[88] a sister companion to this book.), "*** (Quoted from Phaser newsletter no. 94 Examples **http://labs.phaser.io/** » physics » impact » multiple 20 scenes.)

Here is some examples that you might add inside the ***"`create()`"*** function:

[88] https://leanpub.com/phaser3gamedesignworkbook

**Example 3.4: Additional Phaser III Properties**

```
211     // ============
212     // Example 3.4: Additional Phaser Properties begins
213     // ============
214     // remote URL to game assets
215     //   Cross-origin resource sharing (CORS)
216     this.load.setCORS = 'anonymous';
217     this.load.setBaseURL( '<URL to>/images/' );
218     console.log("Additional Phaser Properties set in preload!");
219     //Example 3.4: ends
220     // ============
221
222     //Set a neutral background color
223     //notice we used the shorthand version; instead of #FF0000
224     //photonstorm/phaser/blob/v3.14.0/src/boot/Config.js#L532
225     //This sets up Phaser's Arcade physics engine,
226     // which are simple but effective for arcade-style games.
227     //this.physics.startSystem(Phaser.Physics.ARCADE);
228     //this.renderer.renderSession.roundPixels = true;
229
230     //this applies physics to every item.
231     //.enable(object, [bodyType array or group])
232     //.enableBody(object, DYNAMIC_BODY) or
233     //.enableBody(object, STATIC_BODY)
234     //this.physics.world.enableBody(key,CONST.DYNAMIC_BODY);
235
236     //OR ...
237     this.scene.physics.world.enable(this);
238
239     console.log("Additional Phaser Properties set in preload!");
240     //Example 3.4: ends
241     // ============
242     },
243
244     //Recommended scaling for Phaser III
245     //  managed by CSS.
246     function resize() {
247         var canvas = document.querySelector("gameCanvas");
248         var gWidth = window.innerWidth;
249         var gHeight = window.innerHeight;
250         var gRatio = gWidth / gHeight;
251         var gameRatio = config.width / config.height;
```

```
252
253         if (gRatio < gameRatio) {
254             canvas.style.width = gWidth + "px";
255             canvas.style.height = (gWidth / gameRatio) + "px";
256         } else {
257             canvas.style.width = (gHeight * gameRatio) + "px";
258             canvas.style.height = gHeight + "px";
259         }
260     };
```

***Note:*** The example above comes from:
**https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js**

The `this.scene.physics.world.enableBody` above applies physics to every item we create in our game. The `Physics Manager` is responsible for looking after all of the running physics systems in Phaser III. Phaser III currently (as of 20181212) supports three different physics systems:

- ***Arcade Physics (available in both V2.x.x and v3.x.x)***[89] — "The Arcade Physics Plugin belongs to a Scene and sets up and manages the Scene's physics simulation. It also holds some useful methods for moving and rotating Arcade Physics Bodies." You can access it from within a Scene using this.physics.,
- ***Impact JS***[90] — "... a compatible physics world, body and, solver, for those who are used to the Impact way of defining and controlling physics bodies. Also works with the new Loader support for Weltmeister map data.",
- ***Matter***[91] — "The `Matter.Body` module contains methods for creating and manipulating body models. A `Matter.Body` is a rigid body that can be simulated by a `Matter.Engine`. Factories for commonly used body configurations (such as rectangles, circles, and other polygons) can be found in the module `Matter.Bodies`."

[89] https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Arcade.html
[90] https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Impact.html
[91] https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.html

***Exercise:*** Review completed examples in the ***Phaser III Game Prototype Library on the book's website***[92].

***Exercise:*** Review the Phaser's Official examples[93].

***Exercise:*** Learn more about the various Phaser physics engines in the documentation[94].

***Exercise:*** Research these references about using multiple physics engines at the same time. Which Physics System To Chose?[95]

## Deeper Dive: ES9 Modules

***Why mentions this??***

Because in release v3.16.2 (newsletter #139 20190211), we have ***"`scenefiles`". "Scene Files"***[96] are down-loadable configurations to manage and active scenes **ON THE FLY!** This is THE feature I've been waiting for; it means that I can dynamically influence the gamer's sessions by loading membership entitlement, enticements, or rewards.

```
this.load.sceneFile('keyName', 'path')
// allow time to download and processing by the Scene Manager, then
this.scene.start('keyName');
```

The key given must be the class name of the newly downloaded Scene. Once the scene is downloaded by the Loader, it's added into the DOM with a script tag and then processed by the Scene Manager.

[92] https://makingbrowsergames.com/p3gp-book/
[93] http://labs.phaser.io/index.html?dir=physics/&q=
[94] https://photonstorm.github.io/phaser3-docs/Phaser.Physics.html
[95] http://www.html5gamedevs.com/topic/4503-which-physics-system-to-chose/
[96] https://labs.phaser.io/index.html?dir=scenes/&q=

**Enabling dynamically loaded parts of a JavaScript application at runtime**

```
    import(`./section-modules/${link.dataset.entryModule}.js`)
    .then(module => {
        module.loadPageInto(main);
    })
    .catch(err => {
        main.textContent = err.message;
    });
```

## 3.7 Summary

Examples:

- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html***
- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/index.html***

Chapter 3 done! Here's what we've covered.

- Distinguished between `game flow control` and internal "Phaser Essential Functions".
- Identified the various game phases and those names used to describe them.
- Learned the two aspects of game delivery.
- Understand the impact of writing D.R.Y code.
- Discovered the secret to cranking out a game per week.
- Studied a standard game "flow chart."
- Compared "Lord of the Rings" to a grimoire — "Lore of `Phaser v3.x.x`" — how `Phaser III`'s framework architecture works.
- Reviewed JavaScript modules formats.
- Matched various game phases to the JS modules used.
- ***Learned how to "arrange rose bouquets".***
- Identified the internal "Phaser Essential Functions".
- Dissected the "Game Loop `event logic`".
- Understand the new twist on multiple `scenes` in Phaser v3.16.x.
- Studied how multiple `scenes` are used.
- Reviewed the new "Game Loop cycle".
- Discovered the hottest target markets for games.
- Reasoned the correct presentation order of game phases.
- Learned the importance of coding consistency in styling and paradigm.
- Discover several ***FREE online resources.***
- Studied deeper implications on security and asset caching.
- Differentiated core game mechanics from supporting auxiliary support.

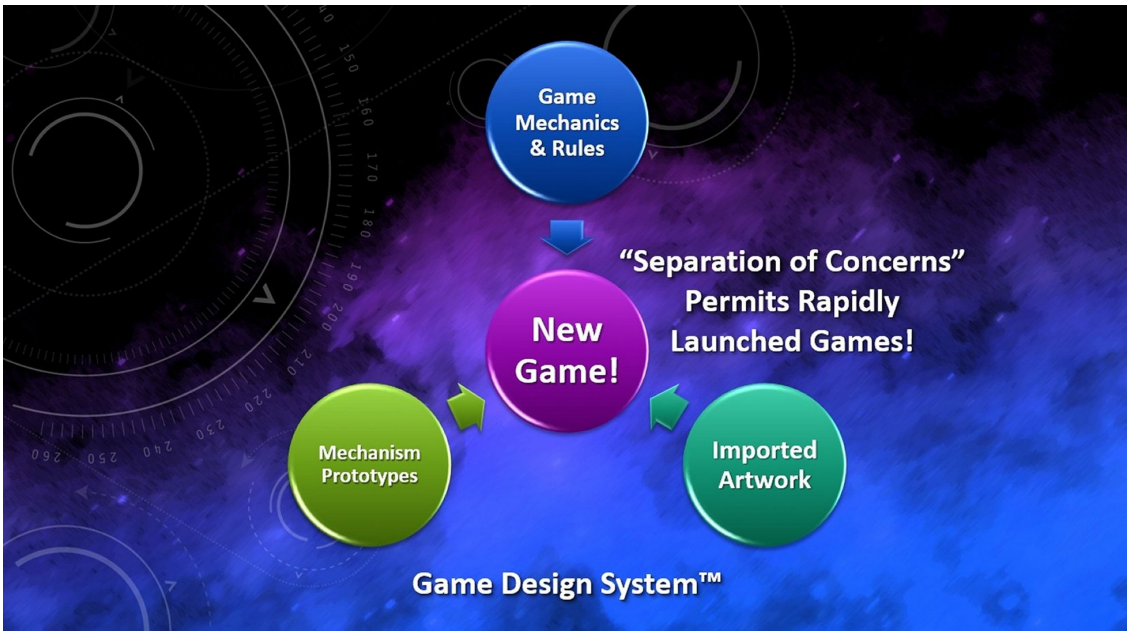
## 3.8 Chapter References:

***(See more references in front)***

- ***Plain English Guide to JavaScript Prototypes***[97]
- ***JavaScript Classes***[98]

[97] http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/
[98] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

# Part II: Mechanisms vs. Mechanics




***Game Design System*™ creating new Games from 3 Components!**

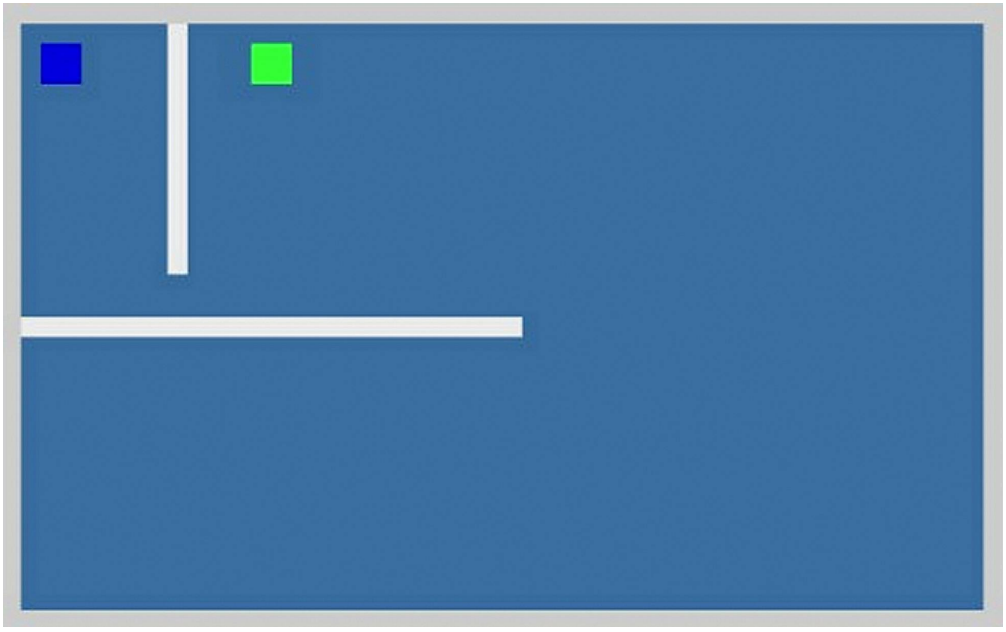
***Part II*** covers ***Game Recipes*™** from the ***Game Design System*™.** This is the "production phase" of my project management. We've created various common "visual components" in both Phaser III and v2.x.x, and prepared our ***Part I*** prototypes for use in this Part II's Game Mechanics (GM), rules, data, and logic.

# 4. Building Game Prototypes, Mechanisms & Tools

Our goal in this chapter is to have a fully functional Game Prototype. From that foundation, we can branch, combine various mechanisms and components into various Game Mechanics found in Part II or from the ***Phaser III Game Starter Kit Collection.***[1]

There are simple worksheets for each task we plan to do. By the end of this chapter we will have created everything a game uses and a tool that will automatically generate ***Game Recipes™.***:

- Interactions between game elements,
- Collecting players' input from the keyboard, mouse pointer, or mobile touch,
- Detecting collisions among various game units,
- Representing visual avatars and their associated data structures,
- Monitoring the gaming loop,
- Migrating to various menus, Scenes, and heads-up displays/changes.



**Chapter 4: Game Prototype Project**

[1] http://leanpub.com/p3gskc/

***Exercise:*** Download the resources from
***https://makingbrowsergames.com/p3gp-book/_p3-demos.zip***
or start a new project following Task #1 & #2 from previous chapters.

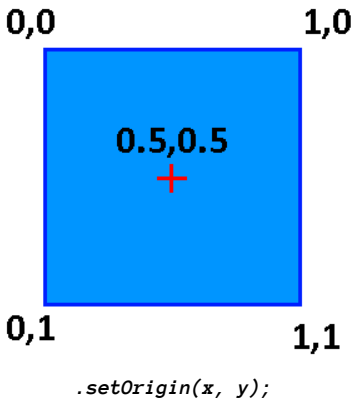
## 4.1 Task #3: Mini-Me

To represent a player's avatar in our game(s), we need to define two things:

1. a **visual image** that represents our gamer and their location in relation to things on the gaming stage; and
2. a separate **data object** describing that avatar's characteristics and skills.

***Note:*** We've already discussed the ***"Principles of Software Engineering"***[2] and the ***"Separation of Concerns"***[3] as effective software engineering principles from Chapter 1.

### Creating an Avatar - *"visual display"*




Let's deal with the "visual display" first. Of course, to do that, we must have some artwork to create our visual avatar. Fortunately, there are dozens of artwork references in the Appendix that are ***free and open source.*** We could use these image files; but, if we want to tailor them that means we either have to purchase, create our own with an image editing program, or perhaps we could hire a graphics artist. We could spend hours ***possibly thousands of hours?! — Remember?*** — in this search.

When placing any Phaser III Game Object (GO), you must remember that visual elements ***are "centered" by default — except for text(!??!) which for some reason, not revealed yet, are "center" by the "top-left".*** Personally, I feel this should be reversed. In Phaser v2.x.x, we used ***"`setting anchor(0.5,0.5)`"*** to have its "anchor" (i.e., its origin point) in the center. If you do not want all visual objects "centered", then use ...

[2] https://makingbrowsergames.com/design/_PrinciplesofSoftwareEngineering.pdf
[3] https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/

**Review Example 4.1: Prototyping a Visual Avatars**

```
56      // ============
57      // Example 4.1: Prototyping Graphics begins
58      // ============
59      // create a character avatar sprite
60      // player1 = this.add.sprite(-100, -100,
61      //   box({who: this, whereX: 150, whereY: 100,
62      //   length:32, width:32, color: 0x0000FF,
63      //   border: 0xFFFFFF}));
64      // ------------
65      // OR just assign the box prototype method
66      player = box(
67                  {who: this,
68                   whereX: 150,
69                   whereY: 100,
70                   length: 32,
71                   width:  32,
72                   color:  0x0000FF,
73                   border: 0xFFFFFF}
74      );
75      // add physics characteristics
76      this.physics.add.existing(player);
77      console.log("Blue avatar created as a 'player' variable.");
78      // ------------
79      // OR the direct method using either rectangle or graphics
80      //    and set movement velocities.
81      var avatar =  this.add.rectangle(100, 100, 32, 32)
82                      .setStrokeStyle(5, 0x3399CC);
83      var graphics = this.add.graphics({ fillStyle: { color: 0xFF0000 } });
84      graphics.lineStyle(10,0x6699CC,0);
85      graphics.strokeRect(100, 100, 32, 32);
86      graphics.fillRectShape(avatar);
87      this.physics.add.existing(avatar);
88      //non-controlled movement (usage AI bot; see Chapter 6)
89      avatar.body.velocity.x = 50;
90      avatar.body.velocity.y = 10;
91      //non-controlled movement (usage AI bot; see Chapter 6)
92      this.physics.add.existing(graphics);
93      graphics.body.velocity.x = 50;
94      graphics.body.velocity.y = 50;
95      console.log("Moving Red avatar variable.");
96      // ------------
```

```
97      // create an opponent - direct rectangle method
98      var monster = this.add.rectangle(180, 60, 32, 32, 0x00FF00);
99      console.log("Green monster avatar created as 'monster' variable.");
100     // Example 4.1: ends
101     // ============
```

***Exercise:*** Refer to these resource files: ***Phaser III Game Prototyping: Chapter 4***[4]
***Lesson 4: https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson04.html***

The ***main >> create >> this.player = this.add.sprite*** experiment — commented out — placed a box shape "off-screen" from the top-left. We drew this image using the ***"box"*** function using the parameters delivered to it:

- the starting coordinates ***(150,100, ...)*** and
- its dimensions ***"(... options.length, options.width)".***

The "Blue Avatar" used just the box function; the moving "Red avatar" was another method. The Phaser III game uses a coordinate system just like the ones used in CSS positioning; the upper-left corner of a game world is: (x=0, y=0). Finally, we filled the box image with our selected colors. Refresh your browser and you should see both the player avatar box and another box.

Review completed examples in this chapter's resource file:

***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/***

There are ***other clever v3.x.x methods at Phaser.io***[5] that we could have used; those are found in the ***Phaser.io v3 examples***[6]. We could reuse our new ***"box" function*** to define any colored rectangle as a game object prototype.

[4] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/
[5] http://labs.phaser.io/index.html?dir=game%20objects/&q=
[6] http://labs.phaser.io/index.html?dir=game%20objects/shapes/&q=


**Sample image from text-image.com. See! Base64 Retro is TOO SEXY!**

I use these two favorite resources to build sprites and ***"`spriteSheets`"*** for my game collection.

- ***Charas***[7] — the online resource generators and review their sample games at ***https://charas-project.itch.io.*** The "Charas Project" is a community founded back in 2003. It's a community for indie-game development with a focus on the "RPG maker" tools. But all forms of development are welcome. If you're not into game development but just want to hang out, you are of course, welcome too. On this ***itch.io page***[8], they share their community games and projects.
- ***Universal LPC Sprite Sheet***[9] — Create a character sprite sheet for your games using ***100% open art.*** I've used "LPC sprites" for my own games with their enhanced "universal spriteSheet". ***Our GitHub repository***[10] is modified for our game prototypes in this book.

## Deeper Dive: Display selected frames from a sprite-sheet.

### Displaying a particular sprite from sprite-sheet

***Paraphrased from:***[a] Phaser III has support for two types of sprite sheet: ***the "classic" ones,*** where every frame is the same size, and ***the "texture atlases"*** that are created with the help of a third-party app like Texture Packer, Shoebox or Flash CC and require

[7] http://www.charas-project.net/charas2/index.php
[8] https://charas-project.itch.io
[9] http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/
[10] https://github.com/MakingBrowserGames/Universal-LPC-spritesheet

***an associated JSON file.***

You could load the ***"classic"*** version with ***"`game.load.spritesheet`".** You must specify its width and height of the frames, and optionally the number of frames. For example ...

game.load.spritesheet('uniqueKey', 'cubee.png', 37, 45, 18);

To use a ***texture atlas*** you must use the ***"`game.load.atlas`".** You'll find several examples in the ***Phaser Examples***[b].

Once loaded, create your sprite:

var sprite = game.add.sprite(x, y, '<spriteSheetKey>');

This tells Phaser to use the image with the key ***"`spriteSheetKey`"*** as its texture. By default, it always jumps to the sprite sheet "frame 0", but you could change it to jump to any frame with the command ***"`sprite.frame`"*** within the spriteSheet.

If the sprite uses an "atlas", it's easier to change its frames based on the given "frame name". For example, ***"`sprite.frameName = 'card4'`"*** wherein the name is exactly specified in the ***texture atlas JSON file*** (open and reference its label!).

[a] http://labs.phaser.io/edit.html?src=src/loader/sprite%20sheet/load%20sprite%20sheet.js
[b] https://labs.phaser.io/index.html?dir=game%20objects/sprites/&q=

## Deeper Dive: Using Base64 Images

Another consideration is using base64 images in Phaser. Many image formats can be converted into ***"base64"***[11]. If you're unfamiliar with what "base64" is or why it exists ***take a look here***[12] and ***here.***[13]

[11] https://www.base64decode.org/
[12] https://en.wikipedia.org/wiki/Base64
[13] https://tools.ietf.org/html/rfc4648

**How to use Base64 as an image**

```
    function create () {
        this.textures.once('addtexture', function () {
            this.add.image(400, 300, 'brain');
        }, this);
        this.textures.addBase64('brain', imageData);
    }
```

See the entire `Base64` example at ***labs.phaser.io***[14].

## Creating an Avatar's metadata

Keeping the ***visual display*** separate from ***its data*** allows us to "re-use" the graphics in a multi-player environment. By changing the colors, the graphics, and customization, it becomes an added benefit when stored inside each unique avatar's data structure.

***Note:*** Review completed examples in the ***Ruins of Able-Wyvern Source code Appendix.***[15]

This data information becomes the descriptive variables about the native abilities and skills of the visualized gamer's avatar. We will use these characteristics to process many outcomes in the Artificial Intelligence state machine.

**Sample: Avatar metadata**

```
    function PersonClass(
        p1, p2, p3, p4, p5, p6, p7, p8, p9, p10,
        p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,
        p21,p22,p23,p24,p25,p26,p27,p28)
        {
        this.PID = p1;              //default - 0
        this.CID = p2;              //default - 1
        this.Name = p3;             //default: Common Adventurer
        this.Score = p4;            //0
        this.TempScore = 0;
        this.Category = p5;         //Warrior
```

[14] http://labs.phaser.io/edit.html?src=src/textures/texture%20from%20base64.js
[15] https://makingbrowsergames.com/p3gp-book/index12.html

```
this.Health = p6;           //Healthy
this.Race = p7;             //Folks
this.Stmn = Number(p8);     //12
this.ModStmn = Number(p9);
this.Fatigue = Number(p10);
                            //p11? - future use
this.Coor = Number(p12);    //12
this.Psych = Number(p13);   //8
this.ModIQ = Number(p14);   //8
this.Renown = Number(p15);  //1
this.HGold = Number(p16);   //0
this.HGem = Number(p17);    //0
this.Movemnt = p18;         //10
this.MegaSQ = 1;
this.Room = 6;
this.Food = Number(p19);    //1
this.WSRaw = Number(p20);   //2
this.WSCmbt = p21;          //NO
this.BSRaw = Number(p22);   //2
this.BSCmbt = p23;          //NO
this.AtkFlag = 0;
this.MisFlag = 0;
this.PryFlag = 0;
this.HitFlag = 0;
this.EngFlag = 0;
this.MovFlag = 0;
this.Target = 6;
this.TLoc = 2;
this.TotalAP = Number(p24); //2
this.Shield = p25;          //Shield Name?
this.Arrows = Number(p26);  //0
this.AName = p27;           //Body Armor Name
this.WName = p28;           //Primary Weapon Name

// PersonClass Inherited Methods:
this.ModMove = function () {
   return this.Movemnt-(this.A[0].MoveMod+this.A[2].MoveMod);
};
this.ModCoor = function () {
    return this.Coor-(this.A[0].CoorMod+this.A[2].CoorMod);
};
this.Level = function () {
    return (((this.Stmn+this.Coor+this.Psych)-26)/6);
```

```
        };
        this.WS = function () {
            return ((this.Stmn*2)+(this.WSRaw*5));
        };
        this.BS = function () {
            return ((PersonClass.prototype.ModCoor*2)+(this.BSRaw*5));
        };
        this.PS = function () {
            return ((PersonClass.prototype.ModCoor*2)+(this.WSRaw*5));
        };
    }

    // End PersonClass
    //=======================================
```

***Live Phaser III Demonstration: Ruins of Able-Wyvern™***[16] Watch the developer's console.

## Deeper Dive 3.19+ Tweens

The updated Tween system introduced in release v3.19 ***is a huge overhaul and has extended the framework capabilities significantly;*** I would advise a review of ***any released games*** using the old Phaser III tween system (pre-v3.19+) before migrating to this newest Phaser III. ***Tweens are fully documented.***[17]

Some new Tween Events are ***'COMPLETE'*** or ***'REPEAT';*** these allow triggered actions without creating callbacks. Another example from v3.19+ is that tweens can implement both ***'to'*** and ***'from'*** values. This is a handy addition whenever you'd like to start from a specific frame in any tweened asset property. `Tween.hasStarted` alerts you concerning a running tween. There's even a new Tween seeking function that provides a search to any point in time across a tween.

Other useful tools newly added in Phaser III.19 are:

- ***'StaggerBuilder'*** — This provides "staggered offsets" to a collection of tweening targets. You might use this while staggering targets across grid layouts and in preferred directions by merely setting a starting value.

[16] https://makingbrowsergames.com/p3gp-book/_p3-arrav15/
[17] https://photonstorm.github.io/phaser3-docs/Phaser.Tweens.html

- ***`Shader.setRenderToTexture`*** — provides a redirection of a shader to its own frame-buffer or WebGL texture instead of using display lists. You might even consider piping one "output" shader as the input to a following shader!
- ***`RenderTexture.snapshot`*** — is the answer to a popularly requested feature. This new feature allows a "snap-shot" on any rendered texture in a point in time and then convert that snap-shot to an image asset for the Texture Manager or as a newly saved image in the file system. I've been waiting for this feature for years!

## 4.2 Task #4: Moving Game Elements

Wouldn't it be nice to click any `arrow` key ***— or virtual "arrow key for mobile devices —*** and have our avatar character respond? Phaser has some nice built-in support just for that purpose. Inside the `main.create()` function, let's add the following line of code to define a keyboard input. We will use it to detect which arrow key was pressed and then have our character avatar reacted to it:

```
// Line 69 - NEW Input Manager v3.24+
// See https://labs.phaser.io/index.html?dir=input
cursors = this.input.keyboard.createCursorKeys();
```

***Note:*** Add a mouse with `this.input.mousePointer` (always refers to the mouse if present). This is the safest method if you only need to monitor the mouse.

Phaser's `main.update()` function checks for input events; remember, `update()` attempts to run at 60 times per second. The `main.update()` function is our game loop, which continues to run until we exit this game phase. So any animation, state, display changes, or game events will be in here.

**Phaser III Game Loop per Scene (as of 20170815; subject to change)**

```
    Game.step();
    └ MainLoop.step
      ├─ All Active Scenes:
      ├─ Scene.sys.begin (called once per active state)
      |  └─ Iterates Scene.children,
      |         if child exists, call child.preUpdate
      ├─ While (frameDelta within step range)
      |  ├─ Scene.sys.update
      |  └─ Scene.update
      ├─ Renderer.preRender
      ├─ Scene.sys.render
      ├─ Update Manager Start (Scene.sys.updates)
      ├─ Game.renderer.render (if Scene is visible)
      |  ├─ Renderer set-up (blend mode, clear canvas, etc)
      |  ├─ Batch Manager Start
      |  └─ SceneManager.renderChildren
      |     └─ Iterates all children, calling child.render on each
      ├─ Update Manager Stop (Scene.sys.updates)
      └─ Scene.sys.end (resets frame delta and panic flags)
```

Let's turn our attention to the speed and velocity of our avatar. We should set a fixed movement speed; you might want to ***"tinker"***[18] with this number until it "feels" correct and proper. We should also set our "velocity" parameter to zero; because doing so, will prevent the avatar's movement until an `arrow` key is pressed. Place the following snippet in the `mainMenu update()` function.

**Example 4.2: Prototyping Movement Properties in v3**

```
    // ============
    // Example 4.2: Prototyping Movement Properties
    //    frame refresh and display updates
    // ============
        cursors = this.input.keyboard.createCursorKeys();
        speed = 250;
        player.setBounce(0.2); // our player will bounce from items
        player.setCollideWorldBounds(true); // don't go out of the map
        player.body.velocity.x = 0;
        player.body.velocity.y = 0;
        console.log("Movable Black character avatar");
```

[18] http://dictionary.cambridge.org/us/dictionary/english/tinker

```
75      // Example 4.2: ends
76      // ============
```

With these parameters set, let's use an `if statement` to determine which `arrow` key was pressed, and then assign a velocity to our character avatar. Our validation should look something as follows in the `mainMenu update()` function:

**Example 4.3: Movement - Arrow Keys Integration**

```
106     // ============
107     // Example 4.3: Movement Arrows Integration begins
108     // NOTE: combination arrow directions are now
109     //    possible with this format
110     // ============
111     player.body.velocity.x = 0; //nothing pressed.
112     player.body.velocity.y = 0; //nothing pressed.
113
114     if (cursors.left.isDown){
115         // if the left arrow key is down
116         player.body.setVelocityX(-speed); // move left
117     }
118     if (cursors.right.isDown){
119         // if the right arrow key is down
120         player.body.setVelocityX(speed); // move right
121     }
122     if ((cursors.down.isDown)){
123         player.body.setVelocityY(speed); // move down
124     }
125     if ((cursors.up.isDown)){
126         player.body.setVelocityY(-speed); // move up
127     }
128     // Example 4.3: ends
129     // ============
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson05.html***

Let's test our new code; refresh the `index.html` page from your web-server; then, in the browser, press an `arrow` key to move the ***"black" avatar box (because there's no texture assigned)*** around the stage. **Press two or three `arrow` key combinations**

**simultaneously and learn what happens.** Adjust the speed variable and observe how it affects your character's movement. Later in this book, we will consider creating various power-up attributes that will increase the speed, or magic spells that might even slow down our avatar. Phaser handles game collisions automatically for us. Add this new line of code to keep the player's avatar inside the visible game stage. You'll discover the ***"black" avatar box*** cannot penetrate the room's walls — yet all the others can, and the ***"black" avatar box*** glides through all the other objects easily.

```
// See line 71
player.collideWorldBounds(true);
```

## Deeper Dive: Phaser III Input Manager

Phaser III handles inputs differently than v2.x.x. In v3.14+, "move events" are ***a new feature — that is completely rewritten as 20181021.*** `Phaser.Input` is the Input Manager for all types of user input; it includes the mouse, the keyboard, mobile touch, and "Game-Pad". The Input manager is updated automatically from the core game loop.

### Quoted from Dev Log 90 & Dev Log 133!

***updated in Dev Log 20181203 for v3.16+***[a]
The Input Manager consists of two parts: The Global Input Manager, which is owned by the Game itself, and the Input Manager, which is a Scene level system.

The Global Input manager is responsible for monitoring and processing user input, regardless of the input method. It starts and handles the DOM event listeners for the keyboard, mouse, and mobile-touch inputs. It then queues these events which are processed every game step.

At the moment we have completed the development of the **Keyboard Handler (as of 20180804; rewritten and updated again 20181203), Mouse Handler, and Touch Handler (v3.16+).** Gamepad and Pointer Lock will be added shortly. (ed.: as of 20171207, Input Manager is 70% completed and new rewrite completed 20181203 v3.16.1.)

These events are dispatched whenever a pointer is in the processing of moving across an interactive object. It doesn't have to be pressed down or dragging, it just has to be moving. As part of the event, you are sent the local coordinates of the pointer within the sprite. So you could use it for a 'sliding' UI element that you control by just sliding a finger up and down it, such as a volume meter.

## Callbacks and Events

In v2.x.x nearly all input was handled via Signals. You'd listen to a signal bound to a specific sprite to know if the pointer was pressed down on it.

**In v3.16+,** you can use both callbacks and events. The events belong to the Input Manager itself, ***not the game objects.*** So, you could listen for a Pointer Down event from the Input Manager. As part of the event properties, you are given a list of all the Game Objects that the pointer went down on, as well as the top-most one in the display list.

***The callbacks, however, belong to the Game Objects.*** You can set a callback for every type of input event: ***"over", "down", "up", "out", "move", and the drag events: "start", "drag", and "end".*** Callbacks are invoked on ***a per-Game Object basis*** and are sent a bunch of useful arguments as well. Depending on the type of game you're building you may favour one approach over the other, or maybe just out of personal preference too. By having both options available though it gives you the flexibility to decide, rather than enforcing it upon you.

```
//Phaser v3 method is extremely easy to activate
var mySprite = this.add.sprite(400, 300, 'texture').setInteractive();
mySprite.setOrigin(0,0); //set sprite anchor to upper left corner
```

## NEW in v3.16.x (JAN 2019!)

The ***Key class now extends EventEmitter*** and emits two new events directly: down and up. This means you can listen for an event from a Key you've created, i.e.: yourKey.on('up', handler).

***The order has also now changed.*** If it exists, the Key object will dispatch its "down event" first. Then the Keyboard Plugin will dispatch ***keydown_CODE*** and finally the least specific of them all, `keydown` will be dispatched.

[a] https://phaser.io/phaser3/devlog/133

### Deeper Dive: Future Proofing your source code.

Not every gamer uses a "qwerty" keyboard; in fact, there are differences between USA and UK keyboards. Many game developers assume the use of "WASD" as substitute arrow keys. Wikipedia summarizes keyboard layouts as, "A keyboard layout is any specific mechanical, visual, or functional arrangement of the keys, legends, or key-meaning associations (respectively) of a computer, typewriter, or another typographic

keyboard.

- Mechanical layout: The placements and keys of a keyboard.
- Visual layout: The arrangement of legends (labels, markings, engravings) that appear on the keyboard keys.
- Functional layout: The arrangement of the key, their associations, as determined by the software, on all the keyboard keys."

***KeyboardEvent.keyCode - Updated Sept 23, 2016, 12:45:21 PM***[a]
***Deprecation Warning: This feature has been removed from the Web standards. Though some browsers may still support it, it is in the process of being dropped. Avoid using it and update existing code if possible; see the compatibility table***[b] ***at the bottom of this page to guide your decision. Be aware that this feature may cease to work at any time.***

The ***KeyboardEvent.keyCode*** read-only property represents a system and implementation dependent numerical code identifying the unmodified value of the pressed key. This is usually the decimal ASCII (RFC 20[c]) or Windows 1252 code corresponding to the key. If the key can't be identified, this value is 0.

The value of a "keypress" event is different between browsers. IE and Google Chrome set the `KeyboardEvent.charCode value`[d]. Gecko sets 0 if the pressed key is a printable key, otherwise, it sets the same keyCode as a keydown or keyup event.

You should avoid using this if possible; it's been deprecated for some time. Instead, you should use `KeyboardEvent.code`[e], if it's implemented. Unfortunately, some browsers still don't have it, so you'll have to be careful to make sure you use one which is supported by all target browsers. Google Chrome and Safari have implemented `KeyboardEvent.keyIdentifier`[f], which was defined in a draft specification but not the final spec.

Web developers shouldn't use the keyCode attribute for printable characters when handling keydown and keyup events. As described above, the keyCode attribute is not useful for printable characters, especially those input with the `Shift` or `Alt` key pressed. When implementing a shortcut key handler, the keypress event is usually better (at least when Gecko is the runtime in use). See Gecko Keypress Event for details[g].

[a] https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode
[b] https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode#Browser_compatibility
[c] http://tools.ietf.org/html/20
[d] https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/charCode
[e] https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/code
[f] https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyIdentifier
[g] https://developer.mozilla.org/en-US/docs/Gecko_Keypress_Event

***Exercise:*** More examples about how to "Future Proof" your ***game source code in this article.***[19]

***Exercise:*** Read more about the differences ***between US and UK keyboards here***[20]

***Exercise:*** Investigate various ***International keyboard layouts used by your gamers.***[21]

### Deeper Dive: Configuring the Keyboard (Phaser v3.16+ updated)

The keyboard is typically another input source. New in Phaser v3.16.1 added the `KeyboardPlugin.resetKeys` as a method that resets the property state of any `Key object` created within a Scene through its Keyboard Plugin. This is automatically called during the scene's ***"`shutdown`"*** method as a part of the Keyboard Plugin. What this means is, as the plugin begins its shut down process or when stopping a Scene, the `KeyboardPlugin` will reset the property state of any key held inside the plugin. It furthermore clears the queue of any pending events.

> "New in v3.16 (JAN 2019) is the ability to receive a global ***"`keydown`"*** or ***"`keyup`"*** event from any key on the keyboard. Previously, it would only emit the event if it came from one of the keys listed in the `KeyCodes` file. Now, those global events ***will fire for any key, regardless of location."*** Read More about ***all the changes in v3.16+ here***[a]
>
> [a] https://phaser.io/phaser3/devlog/134

## 4.3 Task #5: Things that go bump ...

You noticed, by now, that our avatar runs through other objects and stops at the edge of the room (the `camera.view`). Let's fix that.

### Walls and Camera boundaries

Let's now place some walls and immovable objects in our game prototype. For now, we'll put walls on all four sides of the game stage, and then place a few inner walls

[19] http://brianchang.info/2016/01/23/how-to-future-proof-your-code.html
[20] https://en.wikipedia.org/wiki/British_and_American_keyboards
[21] https://en.wikipedia.org/wiki/Keyboard_layout

as decorations. We can use our "box" function to create these walls. You'll observe that our avatar already stops at the edge of the ***"camera.view";*** the walls will provide a rational reason for it to stop at the edge of the screen.

Return to the `main create()`, and add the following code which will construct a wall along the top of the game stage.

**Example 4.4: World Boundaries Integration**

```
94      // ============
95      // Example 4.4: World Boundaries Integration begins
96      // ============
97      //Create Room Walls using rectangles
98          //this.Room = this.physics.add.staticGroup();
99          Room = this.physics.add.group();
100
101         // Creating rectangles; review console in this experiment
102         this.NorthWall = this.add.rectangle(400, 7, 800, 16,0x999999);
103         this.EastWall = this.add.rectangle(793, 234,16,800,0x999999);
104         this.WestWall = this.add.rectangle(7, 234,16,800,0x999999);
105         this.SouthWall = this.add.rectangle(400, 493,800,16,0x999999);
106         console.log("Room external walls created.");
107         console.log("NorthWall: Ext? "+Object.isExtensible( NorthWall ));
108         console.info(NorthWall);
109         console.log("Room external walls created.");
110         console.info(Room);
111         );
112     // Example 4.4: ends
113     // ============
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson06.html***

What did we just do? Firstly, we created a "Room" as a ***group*** for our "wall box" rectangular sprites. We could have created a "Static Group" also. This lets us assign properties to a collection (to the group) of wall boxes. For example, we just enable a physics reaction system on the entire group instead of on each individual wall box. Isn't Phaser too cool?! A "group" permits you to ***(Quote from "Making your first Phaser III game")***[22]***"... group together similar objects and control them all as one single unit.***

[22] http://phaser.io/tutorials/making-your-first-phaser-3-game/part4

***You can also check for collision between Groups and other game objects. Groups are capable of creating their own Game Objects via handy helper functions like "`create`". A "Physics Group" will automatically create physics "`enabled`" children, saving you some leg-work in the process. … When a Physics Sprite is created it is given a "`body`" property, which is a reference to its "Arcade Physics Body". This represents the sprite as a physical body in Phaser's Arcade Physics engine. The body object has a lot of properties and methods that we can play with."***

> ***Quote:***[a] "… In ***Arcade Physics*** there are two types of physics bodies: ***Dynamic and Static.*** A dynamic body is one that can move around via forces such as velocity or acceleration. It can bounce ***and collide*** with other objects and that collision is influenced by the mass of the body and other elements.
>
> ***In stark contrast, a Static Body*** simply has a position and a size. It isn't touched by gravity, you cannot set velocity on it and when something collides with it, ***it never moves.*** Static by name, static by nature. And perfect ***for the ground and platforms (ed.: walls, doors, treasure chest)*** that we're going to let the player run around on.
>
> [a] http://phaser.io/tutorials/making-your-first-phaser-3-game/part4

We created a "North wall" and assigned a rectangle box along the entire game stage width starting at position (0,0). This wall's thickness is 16 pixels. Lastly, ***"`NorthWall.body`"*** is immovable because it is a member of a ***"`staticGroup`";*** this way when another moving piece hits this wall, they will ricochet off. If we didn't have this parameter, then anyone colliding with this wall would move the wall too.

***Hint:*** There's a trick to setting-up the South, West, and East wall groups. They ***cannot*** overlap; otherwise, the walls will send needless update messages and results in a sluggish game. For the "South wall", copy the North wall but change the (0,0 … to (0, config.height - 16 …. The West and East wall groups ***can not*** overlap either of the North or South walls. This means that the West and East wall need to be 32 pixels shorter than the North nor South walls since both of those are only 16 pixels wide.

## Interior Decoration

Lastly, let's add some interior walls to our dungeon-studio room. I'll let you decide how wide and where you'd like to place them. Remember that you can use the ***"`config.height`" and "`config.width`"*** as reference points inside the game stage; it's also

smarter to use relative positions instead of hard-coded and fixed pixel locations. To find the absolute middle of your game stage, take the ***world's width then divide by two*** to get the central X-coordinate, and then take the world's height divided by two to get the central Y-coordinate. Here's a sample:

**Example 4.5: Interior Boundaries Integration**

```
113     // ============
114     // Example 4.5: Interior Boundaries Integration begins
115     // ============
116         Internal1 = this.add.rectangle(120,105,16,180,0xCCCCCC);
117         Internal2 = this.add.rectangle(214,250,400,16,0xCCCCCC);
118         console.log("Created 2 internal walls.");
119         // add all wall to the Room Group
120         Room.addMultiple(
121                     NorthWall,
122                     EastWall,
123                     SouthWall,
124                     WestWall,
125                     Internal1,
126                     Internal2);
127         // debug feedback
128         console.info(Room);
129         console.log("Room Grp obj: Ext? "+Object.isExtensible( Room ));
130         console.info(Room);
131 
132         // separate group for monsters and treasure
133         Tribe = this.physics.add.staticGroup();
134         Tribe.add(monster);
135 
136         // what to do when
137         this.physics.add.collider(player, Room, bumpWall, null, this);
138 
139         //On collision with the monster
140         this.physics.add.collider(player,monster,bumpMonster,null,this);
141     // Example 4.5: ends
142     // ============
143     };
144     function bumpWall(){
145         player.body.velocity.x = 0;
146         player.body.velocity.y = 0;
147     }
148 
```

```
149     function bumpMonster(){
150         player.body.velocity.x = 0;
151         player.body.velocity.y = 0;
152     }
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson07.html***

You've noticed that our avatar was running through and into the room's walls. Well, we fixed that by adding the highlighted code above. Anytime our avatar bumps into an exterior or interior wall, we set its "velocity" to zero in the ***"bumpWall"*** function. You'll notice our avatar still plows through the monster; we'll create another ***"this.physics"*** collider to handle that situation. Yes, we could use the same ***"bumpWall"*** function, but, to provide more flexibility in our gameplay, we should create a separate function.

There's still one small bug in our code. Can you find it? I knew you could! When our avatar slides into the monster, our hero pushes him out of the room through the walls. A couple of ways we could correct this: 1) we could add the monster into the Room's ***"staticGroup",*** and it would adopt all the characteristics of the walls. Well, I don't think our monster is a "wall-flower" … oh no! So let's create a separate group for monsters and treasures.

## Deeper Dive on Game Objects hit areas.

All Game Objects ***(GO)*** in Phaser III now have a ***"hitArea"*** and a ***"hitAreaCallback"*** properties. By default, these are ***set to NULL.*** You can either call `setHitArea` directly on a Game Object, which would return a reference into the GO itself (allowing you to further chain methods through delegation), or you could call the `setHitArea` from the Input Manager — this provides a bit more flexibility. For example, you could pass ***an array (or Group) of GOs*** to `enable` or `enableBody`.

Currently, the ***"setHitArea"*** method takes two arguments: 1) an assigned shape object and 2) a callback function to invoke when a pointer slides over its shape. This "shape hitArea" could be any of the geometry objects available. ***In this example,***[23] there're 5 sprites each of which has their own differently described shape as unique geometry hit area — a circle, rectangle, ellipse, triangle, and finally a polygon. There are interesting side-effects about using ***"Shapes"*** as their "hit areas". Firstly, more than

[23] http://labs.phaser.io/edit.html?src=src/input/mouse/shape%20hit%20tests.js

one Game Object can share the same "shape hit areas". ***This example***[24] demonstrates 400 sprites aligned within a grid, however, all of them share the same ***"Rectangle"*** shape hit area. Each sprite does not create its own unique ***"rectangle"*** hit area. This preserves memory usage in your game — in other words, the less unique objects created, the better!

**Sample: One Shared Hit Area**

```
    function create () {
        // Create a little 32x32 texture to use to show where the mouse is
        var graphics = this.make.graphics(
            { x: 0, y: 0, add: false,
                fillStyle: { color: 0xff00ff, alpha: 1 }
            });

        graphics.fillRect(0, 0, 32, 32);
        graphics.generateTexture('block', 32, 32);
        var highlighted = this.add.image(16, 16, 'block');
        var hitArea = new Phaser.Geom.Rectangle(0, 0, 32, 32);
        var hitAreaCallback = Phaser.Geom.Rectangle.Contains;

        //  Create 400 sprites aligned in a grid
        group = this.make.group({
            classType: Phaser.GameObjects.Image,
            key: 'bobs',
            frame: Phaser.Utils.Array.NumberArray(0, 399),
            randomFrame: true,
            hitArea: hitArea,
            hitAreaCallback: hitAreaCallback,
            gridAlign: {
                width: 25,
                height: 25,
                cellWidth: 32,
                cellHeight: 32
            }
        });
        this.input.on('gameobjectover', function (pointer, gameObject) {
            highlighted.setPosition(gameObject.x, gameObject.y);
        });
    }
```

[24] http://labs.phaser.io/edit.html?src=src/input/mouse/mass%20sprite%20test.js

## Doors, Knobs, and Buttons

There are two ways to create doorways: 1) Doors are immovable "static" objects, and when an avatar collides with it, the avatar appears to have moved into a new room or a new game phase level; **OR** 2) Doors are "clickable buttons" that provide the same transition actions of entering into a new room. In our game prototyping, I like doors to have both actions. Providing both options gives our players a choice of keeping their hands on their keyboard or mouse while playing.

**Example 4.6: Phaser III Doors as Buttons**

```
133     // Phaser III - clicking on a doorway
134     // ============
135     // Example 4.6:  Doors as Buttons
136     // ============
137     // Creating door rectangles; review console in this experiment
138     // placed on a wall with 2px extended into the room
139     doorN = this.add.rectangle(35,0,60,18,0x000000)
140             .setInteractive({ useHandCursor: true })
141             .setOrigin(0);
142     this.physics.add.existing(doorN);
143     doorN.enableBody = true;
144     this.physics.add.collider(player, doorN, changeRooms, null, this);
145 
146     /**
147     OR (following is NOT Optimized code!)
148     this.doorN = this.add.image(400,252,'woodenDoor')
149         .setInteractive({ useHandCursor: true })
150 
151         this.doorN.setFrame(1);
152         this.doorN.setOrigin(0.5,0);
153         this.doorN.setScale(0.7,0.7);
154         this.doorN.name = "north";
155 
156         this.doorN.on('pointerover', function (pointer){
157             console.info(this.doorN.name + " over. ");
158             this._toolTip.setText(GAMEAPP._toolTip);
159             this.doorN.setFrame(2);
160         }, this);
161         this.doorN.on('pointerout', function (pointer){
162             console.info(this.doorN.name + " out. ");
163             this._toolTip.setText("");
164             this.doorN.setFrame(1);
```

```
165         }, this);
166         this.doorN.on('pointerdown',  function (pointer){
167             console.info(this.doorN.name + " clicked down. ");
168             this.doorN.setFrame(0);
169         }, this);
170         this.doorN.on('pointerup',  function (pointer){
171             console.log(this.doorN.name + " click released.");
172             changeRooms(this);
173             console.log("north door entered");
174         this.doorN.setFrame(1);
175         }, this);
176     }
177     */
178     // check for collision
179         this.physics.add.collider(player,doorN,changeRooms,null,this);
180 
181     console.log("Northern Door created.");
182     // Example 4.6: ends
183     }
184     // ... .
185 
186     function changeRooms(){
187         console.log("Leaving Room via Northern Door.");
188         // change scene to a new room
189         // Refer to Part IV - Project Walk-Thu: Rogue Prince Quests™
190     };
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson08.html***

The special ***"Button sprite"***[25] does not exist in Phaser III as it does in Phaser v2.x.x; we must take a sprite, image, or graphic and simply chain (aka "append") the ***".setInteractive()"*** or if you'd like the "hand cursor" to appear then use ***".setInteractive({ useHandCursor: true })".*** Now our visual component can accept "Pointer events" automatically. What did we just do? This created a Northern exit button-sprite with physics and clickable input. In other words, a gamer could slide into the door and transition into the next room as a "scene" change ***OR — and since traveling is not much fun! Admit it! All that traveling in Diablo … did you really enjoy all that??!! —***

[25] https://medium.com/@braelynnn/extending-a-phaser-class-to-make-reusable-game-objects-93c11326787e

the gamer could simply "click" on the door, as any regular button, and enter the next room as a "scene" change.

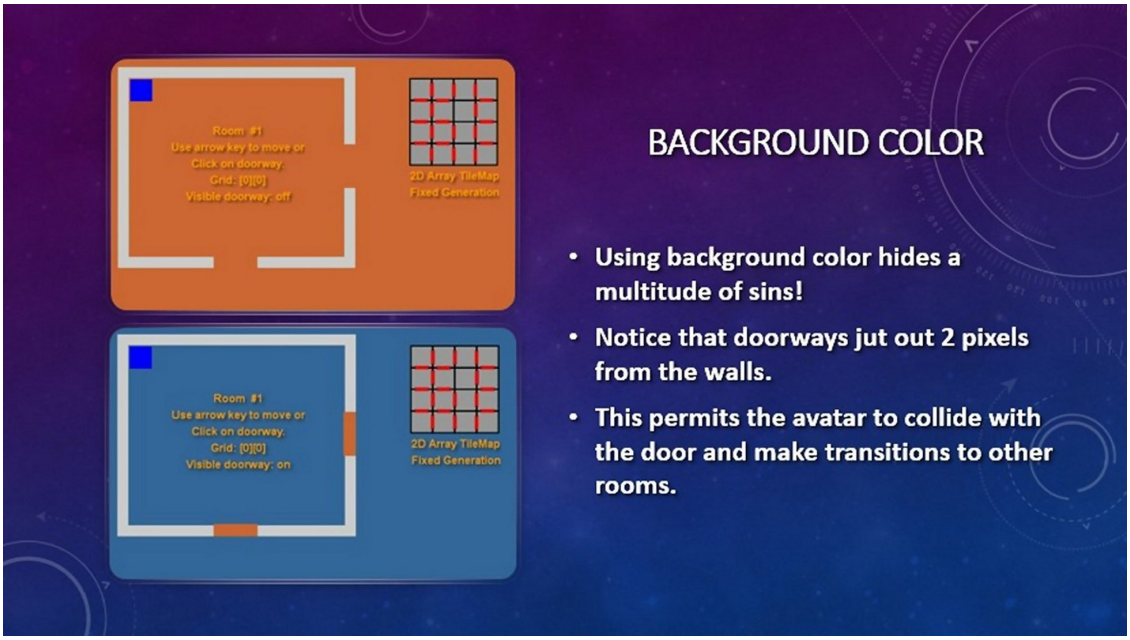
- We should assign a "name" for the door for further processing such as animations.
- We'll eventually assign a spriteSheet frame to display; but for now, we'll just use rectangle prototypes.
- We assign physics to stop the avatar on collision and make the door immovable.
- Finally, we assign "the clickable" (when mouse up or down event trigger) for the door; when clicked we move to the next room. During the game `update`, we watch when the avatar touches the door and launch the room transition function.

Buttons have four internal states that could have different and uniquely separate visual elements, frames, or activated sound effects. ***This also reminds me of Adobe's Flash button movieClips.*** Frames can be specified as either an integer (i.e., the frame ID#) or a string (i.e., the "frame name"; again very similar to Flash Labeled time-line); these same values can be used in a Sprite's construction. Buttons respond when the mouse is:

- **"`pointerover`"** — whenever a Pointer moves over (aka "'hover'") a Button game object. Mobile devices use only the "down-state" below.
- **"`pointerout`"** — whenever a Pointer was previously over a Button and then "moves out" away from it. Mobile devices use only the "up state" below.
- **"`pointerdown`"** — when the Pointer is pressed down while over a Button game object (or "touched" on a touch-enabled device).
- **"`pointerup`"** — whenever a "pressed down" Button was released again.

***Note:*** I highly recommend ***this button plugin***[26] from "rexRainBow" or ***William Clarkson's style for buttons.***[27]

[26] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/button/
[27] https://phasergames.com/how-to-make-buttons-in-phaser-3/




**Sample from Part III: 2D Array and Door Placement**

***Exercise: Try the live v2.x.x demonstration here[28] or v3.16+ demonstration here[29]***

**Sample: Move into New Rooms**

```
    // ===================================
    //Main Door click handler
    function newRoom(door) {
        // 2 Options:
        // - reset this phase with new room characteristics OR
        // - have a "repaint" function to adjust the entered room.
        // Option 1: this.scene.restart();
        // Option 2: separation of concerns - new function
        Rooms2D.LastRoom = Rooms2D.CrntRoom;
        player.setPosition(64, 64);
        var LastDoor = door.name;
        console.log('Last Door Used: '
                    + door.name +" | Toggle: "
                    +bumpToggle);
    switch (door.name) {
        case "North":
            //Rooms2D.CrntRoom -= 4; // or GRID_ROWS or MT.length
```

[28] https://makingbrowsergames.com/book/ch6/index.html
[29] https://makingbrowsergames.com/p3gp-book/_p3-2DRooms/

```
            Rooms2D.CrntRoom = Rooms2D.LastRoom - 4;
            Rooms2D.CrntRoomY -= 1;
            //Leave via North; enter a new room from its South-side
            Rooms2D.pPosX = config.width / 3;
            Rooms2D.pPosY = 320;
        break;

        case "East":
            Rooms2D.CrntRoom += 1;
            Rooms2D.CrntRoomX += 1;
            //Leave via East; Enters a new room from its West-side
            Rooms2D.pPosX = 50;
            Rooms2D.pPosY = config.height / 2;
        break;

        case "South":
            Rooms2D.CrntRoom += 4; // or GRID_ROWS or MT.length
            Rooms2D.CrntRoomY += 1;
            //Leave via South; enter a new room from its North-side
            Rooms2D.pPosX = config.width / 3;
            Rooms2D.pPosY = 50;
        break;

        case "West":
            Rooms2D.CrntRoom -= 1;
            Rooms2D.CrntRoomX -= 1;
            //Leave via West; enters a new room from its East-side
            Rooms2D.pPosX = 340;
            Rooms2D.pPosY = config.height / 2;
        break;
    }

    player.setPosition(Rooms2D.pPosX, Rooms2D.pPosY);
    console.log("New Room #: " + Rooms2D.CrntRoom + ";
                    Door Clicked: " + door.name);
    /**
    // sfx camera fadein/out
    this.cameras.main.once('camerafadeincomplete', function (camera) {
        camera.fadeOut(1000);
    });
    this.cameras.main.fadeIn(1000);
     */
        resetRoom();
```

```
        bumpToggle = false;
    };
```

***Try the live v2.x.x demonstration here***[30] ***or v3.16+ demonstration here***[31]
Download the Phaser III source code from:
***https://makingbrowsergames.com/p3gp-book/p3_2DRoomsDemo.pdf***

Let's review the room state transitions. We delivered to this function the door our avatar used. We then record the current room and the room the avatar is leaving (`GAMEAPP.LastRoom`) so that we can manage a "return path" or "the back-azimuth".[32] The "if" statement could be a "switch" statement; either way, we determine which door was used to change rooms. We send some "debug" info to the console to watch our code and then perform some "magic special effects" — similar to slide transition in a business meeting PowerPoint presentation — to transition into the new game state (aka room). All this is explained later in detail.

## Deeper Dive: Writing Optimized Code

Another item, that should help you in developing a game at a rapid pace, is learning to write simple, modularized code. Here are some guidelines for doing so:

- ***Make your code easily readable:*** The closer your code looks like your native language, the easier it becomes to read, debug, and maintain. This means using a descriptive method, function, and variable names so if someone else were to read your source code, they would easily be able to tell what your intent was.
- ***Minimize code repetition:*** Whenever you notice similar code in more than one place such as doors and walls above, immediately consolidated it into a separate method; this lets you call it from wherever it is needed. Having your common code, in a single place, makes it easier to modify and maintain, and debug. By putting it inside a method with a clearly understandable name, your code becomes easier to read.
- ***Convert code into reusable modules:*** If your code could be used in most of your game products, ***abstract it out***[33] into separate functions or file modules for easy reuse.

***Exercise:*** Read ***"Clean Code"***[34] by Robert Martin converted in JavaScript.

[30] https://makingbrowsergames.com/book/ch6/index.html
[31] https://makingbrowsergames.com/p3gp-book/_p3-2DRooms/
[32] https://en.wikipedia.org/wiki/Azimuth
[33] https://en.wikipedia.org/wiki/Abstraction_principle_(computer_programming)
[34] https://github.com/ryanmcdermott/clean-code-javascript

## Deeper Dive: Buttons as a "Class" or "Scenes"?!!?

The definition of "Ninja", per Doug Crockford's meaning, is "... someone who finds a mistake in the language's design, ***decides it's cool, and then abuses it." Now, let me show you a "ninja trick" on buttons.***

Obviously, a game phase will have a lot of UI controls elements in its heads up display (HUD) and menus. It only becomes natural to make a ***"Button Class" cookie-cutter***[35] and ***"stamp" out***[36] all our pretty buttons for our user's interface(s). Well, you could make an "OOP Class", but ... Phaser.Scenes are also an "OO Class" that can run in parallel with their own physics, camera, and managers for loading and input. ***SO, why not make every button (or HUD Menu chock-full***[37] ***of buttons) their very own Phaser.Scene?!?*** Think of the possibilities ... ninja!

## Deeper Dive: Button size considerations

> ***Quote:***[a] Apple's ***iPhone Human Interface Guidelines***[b] recommends a minimum target size of 44 pixels wide x 44 pixels tall. Microsoft's Windows Phone UI Design and Interaction Guide suggests a touch target size of 34px with a minimum touch target size of 26px. ...
>
> A touch target that's 45 — 57 pixels wide allows the user's finger to fit snugly inside the target. The edges of the target are visible when the user taps it. This provides them with clear visual feedback that they're hitting the target accurately. They're also able to hit and move to their targets faster due to its larger size. This is consistent with ***Fitt's Law,***[c] which says that the time to reach a target is longer if the target is smaller. A small target slows users down because they have to pay extra attention to hit the target accurately. A finger-sized target gives users enough room to hit it without having to worry about accuracy. ...
>
> For users who use their thumbs, 72 pixels does marvels. They're easier and faster to hit because they allow the user's thumb to fit comfortably inside the target. This makes the edges visible and easy to see from all angles. This means that users don't have to reorient their thumb to the very tip to see it hit the target. Nor do they have to tilt their thumb to the side to hit it. One tap with their thumb pad is enough to do the trick.
>
> Another study on Touch Key Design for Target Selection on a Mobile Phone[d] also found that the number of errors decreased as the touch key size increased. In addition, it was provided that the larger the touch key size, the higher the success rate and pressing convenience.
>
> [a] https://www.smashingmagazine.com/2012/02/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/

[35] https://medium.com/@braelynnn/extending-a-phaser-class-to-make-reusable-game-objects-93c11326787e
[36] https://medium.com/javascript-scene/introducing-the-stamp-specification-77f8911c2fee
[37] https://en.wiktionary.org/wiki/chock_full

[b] https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/
[c] https://en.wikipedia.org/wiki/Fitts%27s_law
[d] https://makingbrowsergames.com/p3gp-book/_Touch_key_design_for_target_selection_on_a_mobile_.pdf

## Deeper Dive: Adding Buttons & Mobile Touch

By default, Phaser III starts with only 2 pointers (just enough for 2 fingers to smudge your cell-phone display at the same time). To add more pointers use `addPointer`; this tells Phaser to add more pointers to the Input. The most recently activated pointer is a reference from ***"game.input.activePointer".*** Phaser defines "active" as the pointer generating the most recent event on the mobile device. On a non-surface desktop, this would be the mouse. On an iPhone, for example, it would be the most recent finger actively touching the screen.

Pointers are issued as each new finger is pressed on the screen sequentially. So, if you pressed 3 fingers down, then ***"pointer"*** 1, 2, and 3 would become active. If you then removed your 2nd finger, then ***"pointer2"*** would become inactive; but, ***"pointers"*** 1 and 3 are still active. If you put another finger down, then that touch fills-in the 2nd gap and becomes ***"pointer2"*** again.

In ***Phaser v3.16.1***[38] the **Touch Manager** was "... rewritten to use declared functions for all touch event handlers, rather than bound functions. This means they will now clear properly when the `TouchManager` is shut down. There is a new Input constant `TOUCH_CANCEL` which represents canceled touch events."

```
// Phaser III
game.input.addPointer();
game.input.x || .y = the most recently active pointer coordinates.
```

***Warning: Note:***[39] For iOS, you'll active the minimize app gesture as soon as you use the 6th finger — and there's nothing Phaser can do to stop it.

[38] https://madmimi.com/p/6f870d
[39] http://labs.phaser.io/index.html?dir=input/multitouch/&q=

## 4.4 Task #6: When Worlds Collide ...

There's one small glitch; our avatar character can walk ***through*** the walls. Save this idea for feature enhancements and doors later, but let's fix this for normal gameplay.

A "collision" occurs when two different things touch. You'll discover this concept in many arcade games. Take PacMan for example; during the gameplay whenever PacMan bumps into a dot, the dot disappears. When PacMan touches a ghost, a life is subtracted.

About now, you might be thinking that we'll write a series of `if statements`; but, Phaser v3.x.x anticipated all this and does everything for us inside the `mainMenu update()` function. I just love Phaser! Here's the code we should add:

```
//Line 138
this.physics.add.collider(player, Room, bumpMonster, null, this);
```

Yeap! that's it! Our character avatar will bounce off the walls, and Phaser III handled all that detection automatically. But, how do we handle the situation when our avatar bumps into an opponent ***or a door?*** When the player's avatar bumps into an opponent, let's follow the PacMan example and subtract a life from our character. For now, the player's avatars only have one life to live — so as to illustrate this next feature. Inside the `mainMenu update()` function, add this code at the end:

**Example 4.7: Collision Detection Integration**

```
130     // ============
131     // Example 4.7: Collision Detection Integration
132     // Step 2) Generate sensors | listeners | observers that trigger it.
133     // When overlapping, unlike collide, the objects are NOT automatically
134     //  separated nor have any physics applied,
135     //  they are merely tested for an overlap condition results.
136     // ============
137     this.physics.add.collider(player, Room, bumpWall, null, this);
138     this.physics.add.collider(player, monster, bumpMonster, null, this);
139
140     // using overlapping without a collider
141     // var isOverlapping = Phaser.Geom.Rectangle.Overlaps(rectA, rectB);
142     If (Phaser.Geom.Rectangle.Overlaps(player, monster)){
143         // transition into combat scene.
```

```
144         handlePlayerDeath();
145         // we use this for rectangles and
146         // must change when the final artwork is available.
147     }
148 
149     // ============
150     // Using SceneConfig for physics detection
151     // ============
152     var config = {
153         // ...
154         physics: {
155             default: 'arcade',
156             arcade: {
157                 // x: 0,
158                 // y: 0,
159                 // width: scene.sys.game.config.width,
160                 // height: scene.sys.game.config.height,
161                 // gravity: {
162                 //    x: 0,
163                 //    y: 0
164                 // },
165                 // checkCollision: {
166                 //    up: true,
167                 //    down: true,
168                 //    left: true,
169                 //    right: true
170                 // },
171                 // fps: 60,
172                 // timeScale: 1,     // 2.0 = half speed, 0.5 = double speed
173                 // overlapBias: 4,
174                 // tileBias: 16,
175                 // forceX: false,
176                 // isPaused: false,
177                 // debug: false,
178                 // debugShowBody: true,
179                 // debugShowStaticBody: true,
180                 // debugShowVelocity: true,
181                 // debugBodyColor: 0xff00ff,
182                 // debugStaticBodyColor: 0x0000ff,
183                 // debugVelocityColor: 0x00ff00,
184                 // maxEntries: 16,
185                 // set false if amount of dynamic bodies > 5000
186                 // useTree: true
```

```
187             }
188         }
189         // ...
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson09.html***

This code snippet tells Phaser that when the avatar `overlaps` an antagonist, consult the `handlePlayerDeath` function. The `handlePlayerDeath` function is a new additional block of code written outside the `mainMenu update()` and `mainMenu create()` functions. Notice also in Phaser III, we could set `overlapBias` inside the game's configurations.

Instead of treating the monster as an object, we could have treated it as a ***"zone".*** See how to use ***Phaser III Zones here***[40].

So, we have "thingies" checking whenever an avatar is in contact with a monster. When the avatar moves into a monster or whether the avatar is "overlapping" a monster. Which is better? Do we need both? Well, if you'll remember, we wanted to include a treasure chest inside the monsters group. We could designate them separately. The only thing to take away from this is that one method is using a "collider" to determine when two "physics enabled" objects are in contact or separate from each other while the other validation (i.e., overlapping) is only used when "separation" is not a concern. In our case, consider using "overlapping" when the player is touching a treasure chest. There might arise a situation when the player is touching both a treasure trove ***and the monster using AI (See chapter on Artificial Intelligence)*** Return and add this new snippet of code:

**Example 4.8: Collision Results Determination**

```
181     // ============
182     // Example 4.8: Collision Results Determination
183     // Step 2) Insert NEW function for character's death
184     //         function to calculate the outcome.
185     // ============
186     function handlePlayerDeath(player,enemy){
187         //kill off the avatar
188         //player.destroy();
```

[40] https://codepen.io/samme/pen/yqJoym?editors=0010

```
189         //change to Game Over scene
190         // This method used in AR Series
191         //game.scene.stop();
192         //game.scene.start("gameOver").bringToTop();
193         window.open("lesson11a.html", "_self");
194     }
195     // Example 4.8: ends
196     // ============
197     // preserve everything else below .....
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson10.html***

Our new function — `handlePlayerDeath` — accepts both our avatar and the opponent it is touching as input parameters. The ***Phaser III JavaScript Game Framework*** already has a pre-programmed `kill()` function that removes any graphics sprite from our game stage. ***Ab-bra Cabrera, POOF!*** Our avatar disappears — all by simply defining a separate function to take care of all that "touching" — we'll have none of that here ;)

***Exercise:*** Reflect on what we just learned, and apply it to:

- bullets hitting a target object;
- an avatar "picking up" an item;
- PacMan touching a pill or ghost;
- intersecting with doors; and
- touching treasure troves.

## 4.5 Task #7: It's curtains for you …


**New Game Over Scene**

***It's curtains***[41], our avatar died; the game is over. I'm starting to ***"tear-up"***[42] …. Our game locks-up, because there is no character-graphics symbol to process. So, let's move our game into a new phase called "Game Over". We simply define another new "game phase" with its own `gameOver create()` and `gameOver update()` functions. ***Many Phaser game developers, at this point, will create a new JavaScript file for this "Game Over" phase;*** however, we want to keep this simple for now, and just add this into our current `game.js` file instead.

**Example 4.9: New Game Over State**

```
23      // ============
24      // Example 4.9: New Game Over State begins
25      // Step 3) Transition to the new game menu function for resolution.
26      // ============
27      var gameOverState = {
28          create: function(){
29          }, //comma very important here
30          update: function(){
31          }
32      };
```

For Phaser to recognized this new game-transition, we must add it to the list of game scenes either in the game's configuration ***— or as I recommended earlier — as a new script inside the index page.*** We have a couple of development path options:

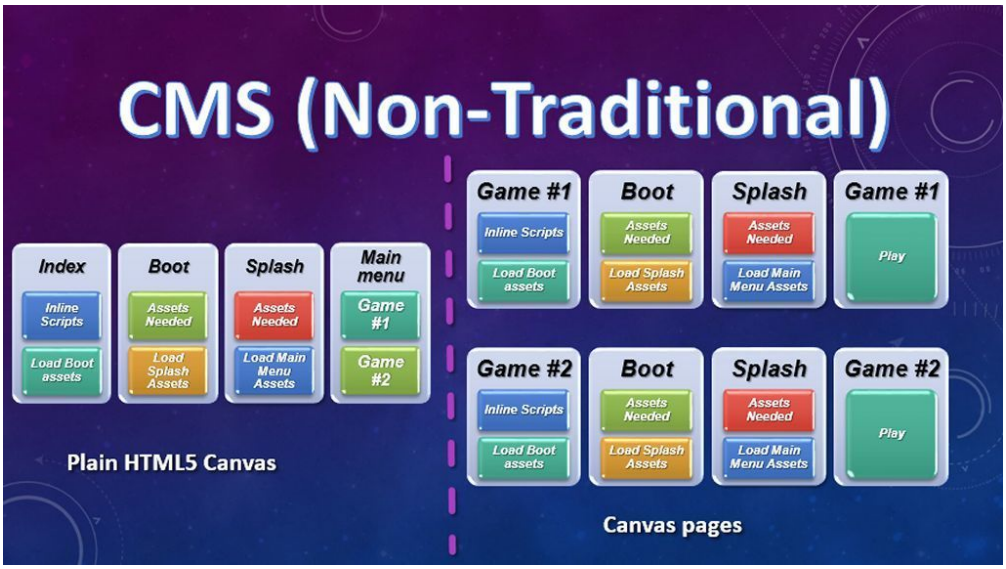
[41] https://forum.wordreference.com/threads/its-curtains-for-you.1509930/
[42] http://www.macmillandictionary.com/us/dictionary/american/tear-up_2

1. type the new scene into the `config.scene` array; ***or***
2. add it using the `game.scene.add('gameOver', gameOver);` ***or***;
3. launch a new html page from the CMS ***or***
4. create a separate script file for this new "game phase" and tell the index.html to load this additional external script.

***Which is better?*** You've learned that we've automated our "rose bouquet" process; anytime an external "game phase" (i.e., module file) is discovered, it is automatically added into our ***"`game.scenes`".*** This is the **"D.R.Y."-est approach** and is already in place. The first two options ***"muck around"***[43] with our development regimen. But to prove the point of ***adding "cruft"***[44] into our game, add the following code and conduct some experiments:

```
// See Line 50
game.scene.add("gameOver", gameOverState); //mucking around!
```

***OR***
add ***"`gameOver: gameOver`"*** into config.scene array. //mucking around!




**Launching a New Game Phase as a separate HTML5 page.**

Inside this ***`gameOver create()` function,***[45] we will learn how to define a text label that displays "Game Over" — a simple "Heads Up Display" (HUD). We'll place this text in the middle of the game stage. Here's our code snippet to do this:

[43] https://idioms.thefreedictionary.com/muck+around
[44] https://en.oxforddictionaries.com/definition/cruft
[45] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11a.html

**Example 4.10: Elementary HUD Creation**

```
    // ============
    // Example 4.9: New Game Over State begins
    // Step 3) Transition to the new game menu function for resolution.
    // ============
    var gameOver = {
    create: function(){
        // ------------
        //Example 4.10: Elementary HUD Creation begins
        // ------------
        var label = this.add.text(
            config.width/2,     //centering HUD horizontally
            config.height/2,    //centering HUD vertically
            "Game Over \n Press the SPACE bar to start again",
            {font: "22px Arial", fill: "#FFF", align:"center"});
        label.setOrigin(0.5,0.5);
        //Example 4.10: ends
        // ------------
    //......
    }, //comma very important here
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11.html***

If you ran this, you'll find the error message "gameOver is not defined". Why? ***Because our new game phase is an object literal.*** So! What does that mean? Object Literals ***are not hoisted to the top of the compiled code as functions are.*** This is a critical concept to understand because JS needs to know about "thingies" before it can use a "thingy". It's worth mentioning also that JavaScript, at runtime, ***internally changes our code*** and moves all variable declarations to the top of its function. This is known as ***variable hoisting.*** Variables declared using ***`let`*** in ES6+ will have block scope and ***will not get hoisted.*** So, if we try to access those variables outside of their block scope, we'll get a reference error saying the variable is not defined. ***ES6+ "`const`"*** variables are similar to the ***"`let`"*** keyword with this additional feature — once they are declared and defined, their state value cannot change. But I digress. Returning to our original problem, the fix is not to sort our code blocks alphabetically as we have been doing, but to move our new "gameOver" ***to the top of the JS file ourselves.*** Many web developers use tools to "blindly" automate this hoisting process for them; and have forgotten to pay attention to this as a result.

Similarly to our graphic's placements, we can place text wherever we choose. Furthermore, we can update that text information using the ***"gameOver update()"*** function. ***This hint is the foundation for building future "heads-up displays" (HUD).***

We told the gamer to press the "SPACE" bar to restart the game; so, we had better create a function to accept that input. In the ***"gameOver create()"*** function, let's insert this code:

**Example 4.11: Collecting User Input**

```
57      // ============
58      // Example 4.9: New Game Over CMS page begins
59      // Step 3) Transition to the new game menu function for resolution.
60      // ============
61      function create() {
62          // ------------
63          //Example 4.10: Elementary HUD Creation begins
64          // ------------
65          var label = this.add.text(
66              config.width/2,     //centering HUD horizontally
67              config.height/2,    //centering HUD vertically
68              "Game Over \n Press the SPACE bar to start again",
69              {font: "22px Arial", fill: "#FFF", align:"center"});
70          label.setOrigin(0.5,0.5);
71          //Example 4.10: ends
72          // ------------
73          //Step 2) Generate sensors/listeners/observers that trigger it.
74          // ------------
75          // Example 4.11: Collecting User Input
76          // ------------
77          var spaceBar = this.input.keyboard.addKey
78                              (Phaser.Input.Keyboard.KeyCodes.SPACE);
79              this.input.keyboard.on('keydown', playAgain, this);
80      };
81
82
83      function update() {
84          //not used
85      };
86      // ------------
87      // Example 4.12:Responding to User Input
88      // ------------
89      function playAgain() {
90          // return to previous game phase
```

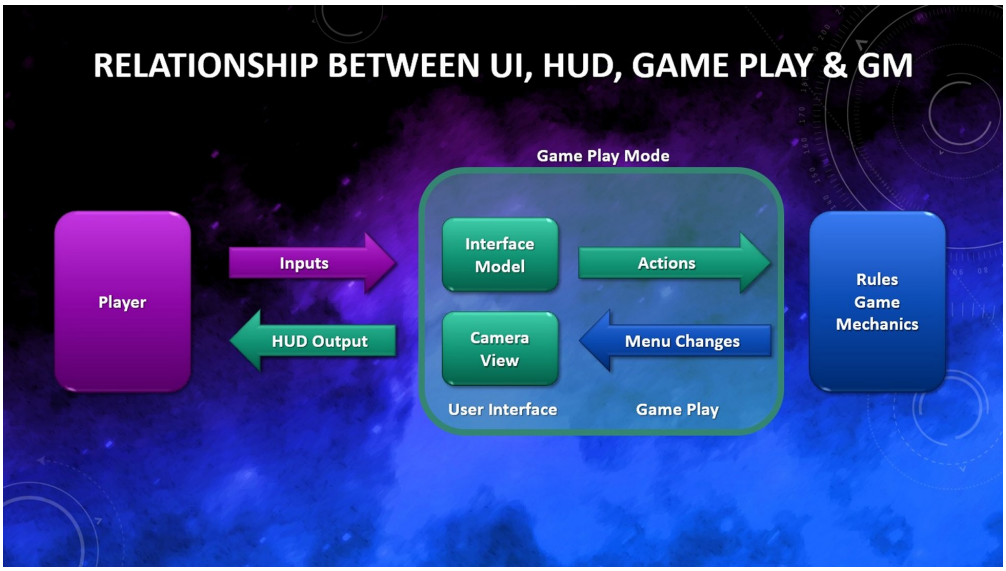
```
91          window.open("lesson11.html", "_self");
92      };
```

***Note:*** Refer to this resource file: ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11.html***

Let's not forget to insert some code in the ***"gameOver update()"*** function to deal with that "SPACEBAR" input signal. That's how easy it is with Phaser. When Phaser processes the ***"playAgain()"*** function whenever the space-bar is down, Phaser returns to the game-play phase.

***Exercise:*** Test what we've just added. Bump into the opponent. Does the game go to the "Game Over" scene? If you press the ***"SPACEBAR",*** does the game move to the initial game launch?

## 4.6 Other Game Mechanics Categories




**Review Game Design System™ (Single Player) from Chapter 2**

Other "Game Framework Mechanisms" (GFM) (i.e., displayed elements, widgets, and input controls) support several more (or less) well-defined categories, along with basic ***Game-Play***, ***Game Rules (aka heuristics)***, mode (i.e., single-player or multi-player),

and genre (i.e., groupings of similar "Game Mechanics"). ***Game Mechanics (chapter 5)*** helps define a game's rules and Game-Play strategies. Many of the following may or may not have visual representation; they may be simply ***abstract data structures,***[46] or ***sentinel variables***[47]. Yes, by mixing and matching various Game Framework Mechanisms (GFM) or Game Mechanics (GM) in a game, it changes a game into a new product release.

***Need Proof?*** Return to ***this file***[48] and change Line 209 ***"`handlePlayerDeath();`" to "`combatEncounter()`"*** and watch how you ***have simply entered a "new game rule" to modify the gameplay!*** Yes! These rules could be in external JavaScript Modules ***that are loaded dynamically on-demand during play. More about that process in the coming chapters in Part II.***

***Hint:*** You must use my file (not yours!) because you don't have the function ***"`combatEncounter()`"*** nor the new game phase ***"`combat`"*** created yet.

Here are a few suggestions from Chapter 5 ***Game Mechanics & Rules*** (aka ***Heuristic***[49]). Many of these could be re-usable components and generalized prototypes for any game. We could "mix and match", "pick and choose" various combinations to generate new "Game Mechanics" (GM) from the following components. ***We could even go so far as to randomly select and combine these mechanics.*** (See Game Recipe™ Automation Tool at the end of this chapter.)

- ***Action points:*** is a budget of activity allocated to restrict what a player may do within their game turn.

```
currentActions -= 1;
```

- ***Agents, Goals, and behaviors:*** (See AI chapter and ***Apple's Game-Play Kit***[50]) Use this simulation to let game characters move themselves based on high-level goals and react to their surroundings.

[46] https://computersciencewiki.org/index.php/Abstract_data_structures
[47] https://en.wikipedia.org/wiki/Sentinel_value
[48] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11.html
[49] https://en.wikipedia.org/wiki/Heuristic
[50] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Agent.html#//apple_ref/doc/uid/TP40015172-CH8-SW1

- ***Artificial Intelligence Strategist:*** (See AI chapter and ***Apple's Game-Play Kit***[51]) Use "MinMax" to provide computer opponents the power of decisions. "MinMax" is a classic AI algorithm that is well suited for turn-based games. Additionally, it could be built into richer systems if you stop thinking about game-turns and start thinking about state transitions
- ***Auction or bidding:*** Players make competitive bids to determine which player gets the privilege to perform particular actions during a game turn. Bids are wagered with some type of collected resource within the game (e.g., game money, points, etc.)
- ***Capture/eliminate:*** the number of tokens a player has on the game board is related to his current strength in the game. How tokens are captured is the mechanics using movement into the same area (immediate elimination or deterministic combat), jumping over and across an opponent as in checkers, producing a "checkmate" event from which the opponent has no movement options. Many online games define the capture mechanics as a "kill count or wounds" that reflects the sum of opponent tokens eliminated during the game.
- ***Catch-up:*** This mechanics is designed to provide increased barriers as the player progress closer to final victory goals. The idea is to allow trailing gamers an opportunity to catch-up and win. This appears in racing games that have a fixed finish line. The opposite approach is to make the leading player more capable of achieving victory (e.g., Monopoly-style games). In such cases, this is desirable in zero-sum games.
- ***Dice as `Randomizers`:*** (See ***Apple's Game-Play Kit***[52]) The most common use is to randomly determine an outcome of a game interaction. This is a deeper problem than most folks will admit. You are often looking for a specific statistical distribution (***NOTE:*** if you haven't played ***AD&D***[53], then think about the `bell curve` derived from rolling two six-sided dice, the most frequently appearing results ***(68% of the time)***[54] lays in the center). ***Apple's Game-Play Kit*** provides all you need, with a variety of cost models (how expensive it is to generate the next random number, versus how actually random it is). Use these robust, flexible implementations of standard algorithms as the building blocks for many kinds of game mechanics.

```
results = currentSkill - randomDiceRoll;
if (results <= to CharacterSkill){ ... Do something ... }
```

[51] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Minmax.html#//apple_ref/doc/uid/TP40015172-CH2-SW1

[52] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RandomSources.html#//apple_ref/doc/uid/TP40015172-CH9-SW1

[53] https://en.wikipedia.org/wiki/Editions_of_Dungeons_%26_Dragons

[54] http://www.compensationcafe.com/2014/04/ding-dong-the-wicked-bell-curve-is-dead.html

- ***Game modes:*** One of the most common examples of a "game mode" is the single-player versus multi-player choice in online games. Multi-Player game can further be subdivided into cooperative or competitive play. Changing modes during a game increases the difficulty and provide additional challenge, or as a reward for player successful action. For example, power-ups are temporary gaming modes or that change only one or more game rules such as pellets in Pac-Man.
- ***Heuristics & Rule Systems:*** (See AI chapter and ***Apple's Game-Play Kit***[55]) Separating game mechanics from its "display code" will optimize your gameplay rendering cycles. Implementing "fuzzy logic reasoning" (such as ***"A*" or "A-Star"***[56] and ***code samples here***[57] ) adds realistic behaviors to your computer-controlled game components.
- ***Movement:*** How game tokens are permitted to move (physics), and when (action points), is controlled by movement mechanics. The current game area may affect movement (e.g., forest areas are more difficult to cross than open prairies.
- ***Playing Cards:*** Decks of cards act as a "randomizer" and/or to act as tokens to keep track of states in the game. Players draw cards and retain them for later use in the game, sometimes without revealing them to other players. When used in this fashion, cards form a game resource. See ***Dice and Random Number Generators (RNG) below***
- ***Resource management:*** an accounting system that monitors the collection (i.e., income) and expenditure (i.e., expenses) of assets. The game will have heuristics that define how players can collect, accumulate, spend, or exchange their resources. Skillful resource management under such game mechanic rules allows players to influence the outcome of the game.
- ***State Machines:*** (See AI chapter and ***Apple's Game-Play Kit***[58]) — Use this architecture to untangle complex procedural code in your gameplay designs. States can capture intent (for example 'am I hunting, hiding, fleeing' using "Path-finding" algorithms such as **A-star**) or overall state (running, jumping, waiting), and of course, you may have more than one state machine in operation. ***Apple's Game-Play Kit*** provides support for both grid-based and one open-world Path-finding models.
- ***Tile-laying:*** Many games use a grid on a world surface to form a tessellation. Usually, such grids have patterns or symbols on their surfaces, which combine when the playing surface is displayed. ***The grid defines the movement rules; how the grid is drawn is a "Game Framework Mechanism".***
- ***Turns:*** A game turn is an important fundamental concept; it could be an abstract representation to regulate gameplay or denote a passage of time or distance in a game set aside for certain player actions to happen before moving to another

[55] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RuleSystems.html#//apple_ref/doc/uid/TP40015172-CH10-SW1

[56] https://www.redblobgames.com/pathfinding/a-star/introduction.html

[57] https://www.geeksforgeeks.org/a-search-algorithm/

[58] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/StateMachine.html#//apple_ref/doc/uid/TP40015172-CH7-SW1

turn. In simulation games, turns represent time and distance in an abstract fashion. War-games usually specify an amount of time each action simulates and are executed sequentially or simultaneously. Even in real-time computer games, there are often certain periodic effects that could be considered the surviving hint of the turn concept.

```
gameTurn += 1;
```

- ***Resource Deployment (aka technology tree):*** is a game mechanics where players allocate a limited number of token resources (lumber, gold, iron, "workers") to multiple stations that provide various defined actions. This is commonly used in Tower Defense games.

***Note:*** ***Apple's Game-Play Kit***[59] provides seven core areas of functionality, which you can combine or use independently to create your game. Because ***Apple's Game-Play Kit*** is independent of high-level game engine technologies, you can combine it with any of those technologies to build a complete game such as: ***"SpriteKit" for 2D games,***[60] ***"SceneKit" for 3D games,***[61] or a custom or third-party game engine using ***"Metal"***[62] or ***"OpenGL ES".***[63] For games with less demanding graphics needs, you can even use ***Apple's Game-Play Kit***[64] with ***"UIKit"*** (in iOS or tvOS) or ***"AppKit"*** (in OS X).

## 4.7 The Finish Line: You're AWESOME … Gloat, Gloat …

If the game is fully operational at this point, ***it's "Miller Time"***[65], but remember to be humble and kind![66] Celebrate! You have a fully function Phaser v3.x.x game prototype that:

[59] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/index.html#//apple_ref/doc/uid/TP40015172-CH1-SW1
[60] https://developer.apple.com/documentation/spritekit
[61] https://developer.apple.com/documentation/scenekit
[62] https://developer.apple.com/documentation/metalkit
[63] https://developer.apple.com/documentation/opengles
[64] https://developer.apple.com/documentation/gameplaykit
[65] http://www.urbandictionary.com/define.php?term=miller%20time
[66] https://www.youtube.com/watch?v=awzNHuGqoMc&list=PLuvCpe8H09C8ViEKyO4Qzo25JUaziIEpK

- accepts inputs.
- moves various game components, and,
- reacts to internal objects.

Using the Phaser.io documentation and the following remaining chapters in this book, we'll have an ***"AWESOME SAUCE"***[67] game collection. ***Making one game a month is now a reasonable and achievable goal using our Game Recipe™ Automation tool.***

## 4.8 Chapter Source Code & Demo

book website: https://makingbrowsergames.com/p3gp-book/

Complete Chapter ***Source Code in the online appendix.***[68]

***Play III Game Prototype Demo thus far***[69]

- ***Example 2.4 Bare-bones Index Page - Traditional Method***[70]
- ***Example 2.5: Starting the Game.js***[71]
- ***Example 3.1a: Creating State Objects in Game.js - traditional method***[72]
- ***Example 4.1: Prototyping a Visual Avatars***[73]
- ***Example 4.2: Prototyping Movement Properties in v3***[74]
- ***Example 4.3: Movement Arrows v3 Integration***[75]
- ***Example 4.4: World Boundaries Grouping***[76]
- ***Example 4.5: World Boundaries Integration***[77]
- ***Example 4.6: Interior Boundaries Integration***[78]
- ***Example 4.7: Collision Detection Integration***[79]
- ***Example 4.8: Collision Results Determination***[80]
- ***Example 4.9: New Game Over State***[81]

[67] http://www.urbandictionary.com/define.php?term=Awesomesauce
[68] https://makingbrowsergames.com/p3gp-book/tools.html
[69] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/
[70] https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html
[71] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson02.html
[72] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson03.html
[73] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson04.html
[74] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson04.html
[75] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson05.html
[76] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson06.html
[77] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson07.html
[78] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson08.html
[79] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson09.html
[80] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson10.html
[81] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11.html

- ***Example 4.10: Elementary HUD Creation***[82]
- ***Example 4.11: Collecting User Input***[83]
- ***Example 4.12: Responding to User Input***[84]

## 4.9 Summary

Examples:

- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html***
- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/index.html***
- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/index-OLOO.html***

Here's an inventory of what we've learned thus far.

- Game Prototyping uses simple graphics and focuses on game mechanics.[85]
- Created Game Prototype that accepts inputs.
- Created Game Prototype that moves various game components.
- Created Game Prototype that reacts with internal objects.
- Created a web page to launch our Phaser Prototype.
- Learned about Content Delivery Networks.
- Discovered various game phases and states to modularize[86] our game.
- Learned the 9 Phaser essential functions of which the "create", "update" and "render" are the most active.
- Studied a typical Skeleton state file.
- Reviewed the traditional game menu states.
- Discovered a Phaser game can use multiple physics engines, but only one physic engine is assigned to one graphics sprite.
- Created a gamer's representation in the game world.
- Learned how to generate sprite graphics from code.
- Attached speed and velocity to moving game objects.
- Attached various input signals to manipulate game objects.
- Attached reactions to immovable and movable objects.
- Learned how to trigger various behaviors.
- Created game stage boundaries.
- Discovered how to transition game between states.

[82] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson12.html
[83] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11a.html
[84] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11a.html
[85] http://www.lostgarden.com/2006/10/what-are-game-mechanics.html
[86] http://www.dictionary.com/browse/modularize

## 4.10 Chapter References

- ***How to Prototype a Game in Under 7 Days***[87]
- ***MDN Game development***[88]
- ***Game Design Concepts 5.1: Prototyping***[89]
- ***Plain English Guide to JavaScript Prototypes***[90]
- ***JavaScript Classes***[91]
- ***https://www.nolo.com/legal-encyclopedia/types-databases-that-cant-be-protected.html***
- ***https://www.bitlaw.com/copyright/database.html***
- ***https://data.research.cornell.edu/content/intellectual-property***
- ***https://en.wikipedia.org/wiki/Sui_generis_database_right***
- ***https://www.michalsons.com/blog/the-rights-to-a-database/2937***

[87] https://www.gamasutra.com/view/feature/130848/
[88] https://developer.mozilla.org/en-US/docs/Games
[89] https://learn.canvas.net/courses/3/pages/level-5-dot-1-prototyping
[90] http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/
[91] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

# 5. Dem's fightin' words

In this chapter, we will develop several methods for "conflict resolution" (aka "combat") between the player's avatar and their antagonist(s). We will also add several new game prototype enhancements:

- ***Launching Web Sockets,***
- ***Dynamic menus,***
- Melee, hand-to-hand, and ranged combat,
- Tactical movement styles,
- ***"Tile Maps" for tactical movement,***
- Conflict resolutions: who, what, when, how
- Story narratives,
- Post Combat.

***Note:*** Refer to these resource files:
***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/***

## 5.1 Launching Web Sockets

We talk more in depth about WebSockets and how to use them properly in ***Massive Multi-player Games***[1]; but for now, here's how to include WebSockets in your game products. You can also experiment with our ***MMoG server.***[2]

[1] https://leanpub.com/rrgamingsystem/
[2] http://mmog.pbmcube.net/

**Example 7.1: Launching Web Sockets.**

```
29      <!-- Creating Client WebSocket -->
30      <script >
31      function WebSocketTest() {
32          if ("WebSocket" in window) {
33              alert("WebSocket is supported by your Browser!");
34              // Let us open a web socket on local development site;
35
36              //TODO: Change the URL to point to your live production server.
37              //Classic OOP style creates object/function:
38              var ws = new WebSocket("ws://localhost:9998/echo");
39
40              //============================
41              // 4 WebSocket Protocol Msg
42              //============================
43              ws.onopen = function onopen(event) {
44                  // Web Socket is connected, send data using send()
45                  ws.send("Test Message sent");
46                  alert("Test Message away, away ... Captain!");
47              };
48              ws.onmessage = function onmessage(event) {
49                  var received_msg = event.data;
50                  alert("Incoming Messages ... brace for impact, Captain!");
51              };
52              ws.onclose = function onclose() {
53                  // websocket is closed.
54                  alert("Connection is closed...");
55              };
56              //============================
57              // End of WebSocket Protocol
58              //============================
59          } else {
60              // The browser doesn't support WebSocket
61              alert("WebSocket NOT supported by your Browser!");
62          }
63      }
64      </script>
```

***Note:*** Refer to these resource files:
***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson01.html***

## 5.2 Dynamic Combat Menus

What would be extremely nice, is to tell the player when they can "attack". So let's put some ***dynamic menu buttons*** that appear only when an avatar is engaged in ***"hand-to-hand or Melee conflicts.*** BUT ONLY, if they have a readied melee weapon. When an avatar is ***not "touching"*** (i.e., engaged) in melee combat, AND has a missile weapon readied then let's show a "FIRE" button to launch a missile attack.

**Example 7.2: Dynamic Combat Menus**

```
609     //=========================================
610     function combatEncounter(){
611       // ============
612       // Example 7.2: Changing Game Scenes
613       // Step 2) Exchange character's death to Combat
614       //         function to calculate the outcome.
615       // Step 3) Transition to the new game menu function for resolution.
616       //         New Combat Scene begins with a scene transition
617       // ============
618
619       // previously, Chapter 4 method moved to a new page
620       // window.open("lesson11a.html", "_self");
621
622     /**
623     // Chapter 7.2 methods move to a new scene and conducts "Combat"
624     // using Phaser III new scene.transition feature
625     // NEW Phaser III scene.transition feature
626     scene.scene.transition({
627         target: nextScene, // the next Scene key to transition into
628         data: null,        // a data object containing anything you wish
629                            // passed into the target's init or create methods.
630         moveAbove: false,  // move the target Scene above this current
631                            //  scene before the transition starts
632         moveBelow: false,  // move the target Scene below this current
633                            // scene before the transition starts
634         duration: 1000,    // delay processing duration in ms
635         sleep: false,      // true = to sleep current scene;
636                            // false = to stop current scene
637         allowInput: false, // true = to enable input system of current
638                            //   scene and target scene
639         onUpdate: null,    // function to call; example "this.transitionOut"
640         onUpdateScope: scene
```

```
641     })
642     */
643
644         console.log("%c Entering combat game phase. \n
645                 The main Scene goes to sleep!   ",
646                 "color:white; background:blue");
647
648         game.scene.sleep('main').sendToBack();
649         game.scene.start('combat');
650
651     };
```

***Hint:*** This complete source is ***available from the website Lesson 2.***[3]

In the example above, we created our "attack" and "fire" buttons and placed them in separate "containers" — an "engaged" and "disengaged" containers, then we positioned "engaged container out of sight". In the ***"`update`"*** essential function, we will swap the container's locations as soon as the antagonists collide — become "engaged" in melee. This is a "carry-over" from my former Flash game development days when I simply created everything and moved the inactive "movieClips" off stage.

I could have just toggled the "visibility" of each button. Phaser Gaming Frameworks give us more flexibility than Flash when creating Dynamic Menus. There are several design options we could use for games discussed later; some of these options are:

1. OR, we could create "menu option sets" inside of ***"containers" (Read more details both pros and cons here.);***[4] then, move those buckets of content onto and off of the stage.

```
// Menu HUD containers
engageC=this.add.container(1000,0,[attackButton,attacktxt,disEngtxt]);
disengageC = this.add.container(0,0, [fireButton,firetxt]);
```

[3] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson02.html
[4] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container.html

2. OR simply toggle this Menu HUD container's visibility. These design options are discussed later.
3. Create Phaser III HUD Scenes and shuffle those "HUD Scene stacks" to the front/back OR create/sleep then this following command is required:

```
    this.input.setGlobalTopOnly(true);
```

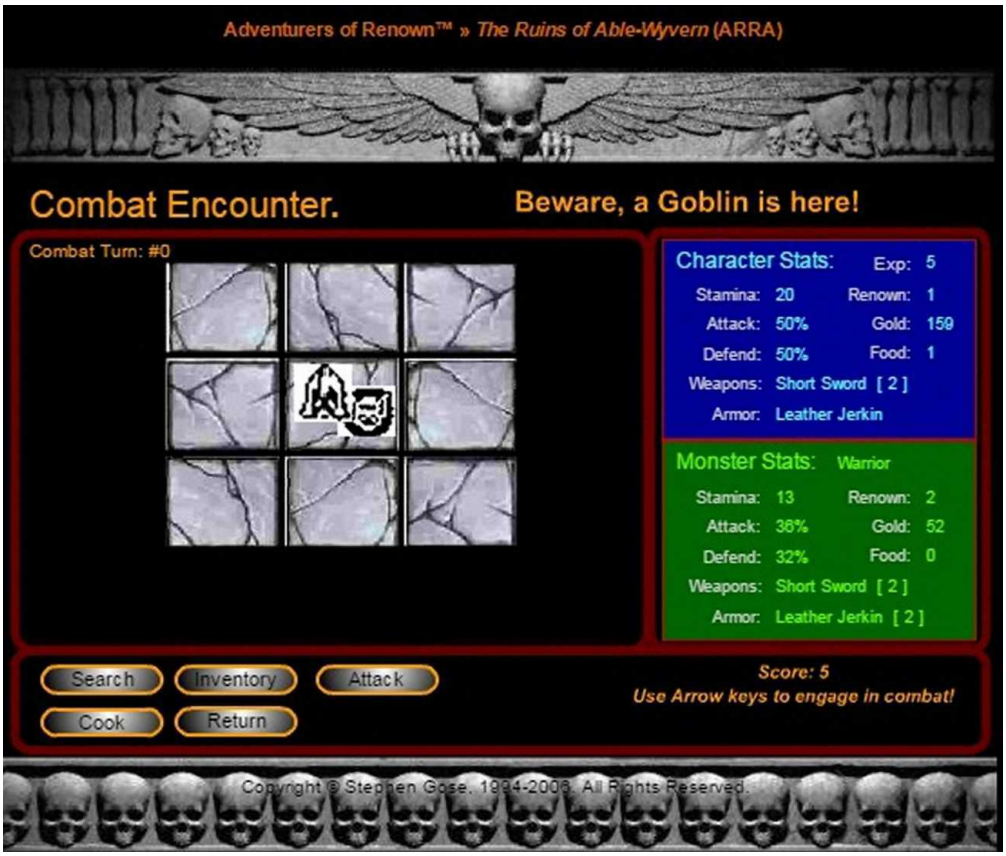
**Example 7.3: Dynamic Combat Menus supporting function**

```
695     // NEW melee combat: Dynamic Menu; engaged in melee
696     //=========================================
697     function meleeEncounter(){
698         //=========================================
699         // Example 7.3:  Dynamic Combat Menu Buttons
700         // Design Options:
701         //=========================================
702         // 1) Follow Flash games placing menu buttons in & out of the stage
703         //
704         // 2) Create containers to mimic former Flash Movie Clips
705         //  placing menu inside then shuffle HUD container(s) on & off stage
706         // a. Phaser III using "Groups"? Problems are:
707         //  1) "Group membership is non-exclusive."
708         //  2) "Groups aren't displayable, can't be positioned,
709         //      rotated, scaled, nor hidden."
710         //
711         // b. Phaser III using Container(s)? Problems are:
712         //  Read pros and cons from Phaser III docs
713         //  https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container.\
714 html
715         //
716         // 3) Create Phaser III HUD Scenes
717         //  shuffle HUD Scene stacks to the front/back OR create/sleep
718         //  then this following command is required:
719         //  "this.input.setGlobalTopOnly(true);"
720         //
721         // 4) Create simple click-able text menus -
722         //  i.e., retro dumb terminal or BBS style.
723         //=========================================
724         // Design Options #2
725         engageC.setX(0);
```

```
726         disengageC.setX(1000);
727         gameStatus = 1;
```

Review this source code at either:

- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/***
- ***https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/***




**Attack button only appears when engaged in melee Combat**

## 5.3 So, Give Me Some Space …

### "The Four Virtues of a good tactical turn-based combat system"

So let's get specific. There is a veritable cornucopia of techniques that game developers have used in the past to make their turn-based combat systems sparkle with tactical possibilities, and I want to see new RPGs start using them with greater regularity. Perhaps the ***most powerful technique is simply to:***

1. ***Use space.*** Adding a spatial dimension to combat increases its complexity exponentially without making it substantially harder for the player to understand. Most people have played games like ***Candyland***[a] or ***Monopoly***[b], to say nothing of ***Checkers***[c] and ***Chess***[d]. Everyone (even your mom) intuitively understands the concept of moving pieces between spaces.

By using space in your battles, you add a new dimension to combat both figuratively and literally: ***the concept of attack range*** comes into play, and the ***player gains direct control of actions like fleeing and protecting weaker characters*** behind stronger ones.

Of course, you aren't required to have a ***grid-based (or hex-based) map*** with movable characters to create a good tactical combat system, ***but it's an awfully effective way*** to introduce complexity using simple rules. This alone will put your game ***far ahead of most jRPG combat systems.***[e]

[a] http://amzn.to/2l1flud
[b] http://amzn.to/2mfZ5uZ
[c] http://amzn.to/2mg3YUT
[d] http://amzn.to/2l1qgJA
[e] http://amzn.to/2m1WdWm

***"Engage or Not to engage that is the questions."*** This one question adds all of the ***"The Four Virtues of a good tactical turn-based combat system";*** by simply adding an "arena space" for antagonists to resolve their conflicts.


**Upon entering a room, *what's a girl to do?***

We will develop two different versions of combat along with these ***spacial aspects:*** (not special; space … the final frontier!)

- ***Grid-less:*** similar to our method of movement created in the last chapter; and,
- ***Grid-ed:*** regulating tactical movement using squares, hexagons, or squishes.

Why? Because a conflict between opponents can take place as either:

- ***"hand-to-hand / Melee" conflicts.*** In the last chapter, we developed source code to recognize when objects were ***touching one another.*** That source code will become our ***"hand-to-hand" (aka "engaged" or "melee") combat.*** There is a subset of this combat I use by the same name. It is the situation when two opponents are "wrestling" and "grappling on the ground". Shorter weapons and strength have the advantage here. I treat this differently than normal melee in my games.
- ***"Ranged" conflicts.*** Ranged combat is a different story; it involves sending a "missile" toward an opponent. Should the missile strike home, our opponent is "hit"; the missile should disappear, and we reduce a life — or "hit points" — from

our targeted opponent. Sound familiar? We did all this before. ***In ranged combat,*** the missile is a ***newly created game entity*** that speeds toward its intended target. If it "touches" that target, we calculate the outcome. If it misses, the projectile continues forward until it smashes into something (e.g.: the world boundaries? a team member? another opponent?); and at that time, it is removed from play. ***That was not revealed previously.***

## Melee Weapons

If two antagonists are "touching", they are considered engaged in "melee" combat. They will use one-handed or two-handed weapons. In ***"hand-to-hand*** combat, "shorter" weapons, such as pistols, knives, daggers, brass-knuckles, will have normal weapon damage; one- and two-handed weapons will have a disadvantage in weapon damage.

## Ranged Weapons

Phaser v2.x.x has 10 different styles for ranged weapons in their online examples; these styles have their own separate class function; as Phaser III "matures" I am certain these 10 different styles will appear in Davey's scheduled book. I told you that Phaser has thought of everything. There are several 'combination' weapons — which are essentially mixtures from those established 10 styles. This ***Phaser III `Weapon Plugin` by "rexrainbow"***[5] facilitates creating a "bullet pool" and manager for projectiles. "Weapons fire" plugin generates sprites — as bullets — with a few extra properties and "secret sauce". Each bullet-projectile has its own `Arcade Physics` property enabled. This class follows the same template process ***seen in Phaser III Labs***[6]. Consult the Rexrainbow's documentation about his ***`Weapon Plugin` here***[7] and see all the extraordinary options we could use. Let's create a generic missile object to toss around in combat. This template will represent both "thrown" and "fired" missiles.

Launching thousands of bullets is fun in single-player games; but, you might consider using a limitation in multi-player versions. Multi-player performance will improve by using ***"pools" of available bullets***[8] instead of "creating" and "destroying" bullets in an endless loop per player.

The important concept to remember is when using "ordinary sprites" from a pool, you should toggle the sprite's "active" and "visible" properties on and off. This will

[5] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/
[6] http://labs.phaser.io/edit.html?src=src/pools/bullets.js
[7] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/
[8] http://labs.phaser.io/index.html?dir=pools/&q=

save CPU processing. Since bullet sprites have physics enabled, you should also toggle its "body.enable" property too. Using the ***"enableBody()"*** and ***"disableBody()"*** methods sets all of these three (3) properties at the same time. Another benefit from using ***"enableBody()"*** is that it includes a ***"reset"*** option that will synchronize the sprite's location. You could set all the sprites inside a physics group with these parameters.

```
{ active: false, visible: false, enable: false }
```

### *Phaser III Top-down shooter examples from labs.phaser.io:*[9]

- ***Average Focus***[10] — click while running the source code and observe how the avatar spins to follow the target.
- ***Combat mechanics***[11] — click while running the source code and observe how the avatar spins to follow the target. Sight the target over or beyond the antagonist and then click to fire your weapon.
- ***Player Focus***[12] — click while running the source code and observe how the avatar spins to follow the target. At first, this example seems to mimic the "Average Focus" but use the "WASD" keys to move the avatar while targeting.
- ***Target Focus***[13] — click while running the source code and observe how the avatar spins to follow the target. At first, this example seems to mimic the "Player Focus" but use the "WASD" keys to move the avatar while targeting. You'll notice that the camera stays focused on the target instead of the avatar.

**Phaser III Sample: Projectile Template**

```
1     this.bulletGroup = this.physics.add.group();
2
3     function bullet(){
4         var bulletPoints = this getDirFromAngle(this.player.angle);
5         console.log(bulletPoints);
6
7         var bullet = this physics.add.sprite(
8                 this.player.x + bulletPoints.tx * 30,
9                 this.player.y + bulletPoints.ty * 30,
10                'bullet');
```

[9] http://labs.phaser.io/index.html?dir=games/topdownShooter/&q=
[10] http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_averageFocus.js
[11] http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_combatMechanics.js
[12] http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_playerFocus.js
[13] http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_targetFocus.js

```
        bullet.angle = this player.angle;
        bullet.body.setBelocity(
                bulletPoints.tx*100,
                bulletPoints.ty*100)

        this.bulletGroup.add(bullet);
    }

    function getDirFromAngle(angle){
        // Description:converts degrees to radians
        // ????? var rads = angle*Math.PI/180; ?????
        // Use 3.142 and stop wasting battery power!
        // var rads = angle * 3.142/180;    // OR better still!!
        var rads = angle * 0.01745;         // just pre-calculate it!
        var tx = Math.cos(rads);            // see chapter footnotes
        var ty = Math.sin(rads);            // see chapter footnotes
        return {tx,ty}
    }

    this.physics.add.collider(
                this.bulletGroup,
                <targetGroup here!>,
                <what happens?>, null, this);
```

To preserve CPU processing and battery, pre-calculate math formula. Refer to ***sine and cosine here***[14]. ***One radian equals***[15] 180°/ π = 57.30°. Use this ***online calculator***[16] to help reduce CPU workload at runtime and thereby save battery power. Refer to ***this article***[17] for a gentle introduction (or reminder) on degrees, radians, and angles.

A common mistake in game development is using "PI". The standards state that it is an approximation which is approximately 3.1415926535897932. Do you truly need that much accuracy? We're not sending "men to the moon." Why burn CPU processing and consume battery power when you could have simply provided the number 3.1415 and not make the game calculate for each bullet. It's small things, such as this, that accumulate and slow down your game's responsiveness.

[14] http://www2.clarku.edu/faculty/djoyce/trig/cosines.html
[15] https://ee.stanford.edu/~hellman/playground/hyperspheres/radians.html
[16] https://www.rapidtables.com/calc/math/Cos_Calculator.html
[17] https://betterexplained.com/articles/intuitive-guide-to-angles-degrees-and-radians/

In the example template above, notice that the missiles are created inside of a "Group object" with ***"Arcade Physics"*** already enabled. You can do anything with a Group (such as move it around the display list, etc.) that you normally would do with sprites. "Bullets" can have textures and even animations. You can control the speed, angle, and rate at which they are fired, and even set additional properties such as gravity. Just keep in mind that each bullet is a game object that requires CPU processing, memory, and rendering onto the display. To conserve on these, I chose to simply have a "FIRE" button to abstract missile combat.

Here is a Phaser III ***Bullet plugin documentation here***[18] and ***source code at GitHub***[19]

***And other sample tutorials with (FREE!) source code:***

1. ***Zenva Game Academy: "How to make a Tower Defense Game"***[20]
2. ***William Clarkson: "Phaser 3 Physics for beginners - Endless Bullets"***[21]
3. ***Phaser3 Labs — Defenda!***[22]

***Hint:*** We will revisit these missile functions when we create "magic missiles".

Since our Game Prototype uses the "top-down or Bird's Eye" perspective, gravity won't play a part in our ranged combat. If our Game Prototype used a "side-scroller" view, gravity would add "juice"[23] to our game. We'll touch on different game perspective views later (i.e., 3rd person and 1st person).

### Is your game 'juicy' enough?

***Now what exactly does that mean?*** "Juicy things are things that wobble, squirt, bounce around, and make little cute noises; it's sort of a catch-all phrase for things that make a game more satisfying to interact with," Jonasson explained during his presentation at ***GDC Europe.***[a] "Juice is typically auditory or visual, but it doesn't really need to be … it's about the maximum output for the minimum input."

Here are some ways to enhance missile combat:
* Turn it side-ways and create a vertically scrolling shooter instead.

[18] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/
[19] https://github.com/rexrainbow/phaser3-rex-notes/blob/master/examples/bullet/bullet.js
[20] https://gamedevacademy.org/how-to-make-tower-defense-game-with-phaser-3/?a=47&campaign=Phaser3GamePrototyping
[21] https://phasergames.com/phaser-3-physics-beginners/
[22] http://labs.phaser.io/edit.html?src=src/games/defenda/test.js
[23] http://www.gameanalytics.com/blog/squeezing-more-juice-out-of-your-game-design.html

* Give the missile's acceleration; instead of `Velocity` then watch them increase in speed over time.
* Give the missiles a "waypoint" in order to "home in" on targets.

[a] https://www.pocketgamer.biz/news/64630/gdc-europe-is-no-more/

## 5.4 OO!, OW! AH!, OW! Stayin' alive! Stayin' alive!

We have our two combat functions; let's now review ***Tactical movement styles.*** As mentioned earlier, a combat encounter could have two spacial (as in space the final frontier) aspects:

- ***Grid-less:*** similar to our method of movement created in the last chapter; and,
- ***Grid-ed:*** regulating tactical movement using squares, hexagons, or squishes.

### Grid-less Combat

***In short, 'Grid-less Combat" is a copy of the Part I examples.*** when our avatar bumps into an opponent, we transition our game into a "combat scene", instead of "killing the player" and declaring the game finished. Earlier, when our avatar died, we moved directly ***into the "gameOver scene";***[24] we will exchange that scene for our new "combat encounters" scene. Now, when bumping into an opponent, we will move into a new game phase that focuses on combat tactics, using the same techniques we learned before in Part I. In this new "Combat Encounter Scene" we will conduct our deadly conflict until a victory is determined. Of course, if our player's avatar is defeated, our game will go to the normal "Game Over" scene as we did once before.

The ***"`gameCombat`"*** scene will have its own ***"`gameCombat create()`"*** and ***"`gameCombat update()`"*** functions (as do all game phases; nothing new so far). These will handle our conflict and collect player's input and tactics ***from dynamic menus — this is new!*** We will provide feedback on selected tactics through a ***head-up display (HUD)*** and the ***story combat narrative.***

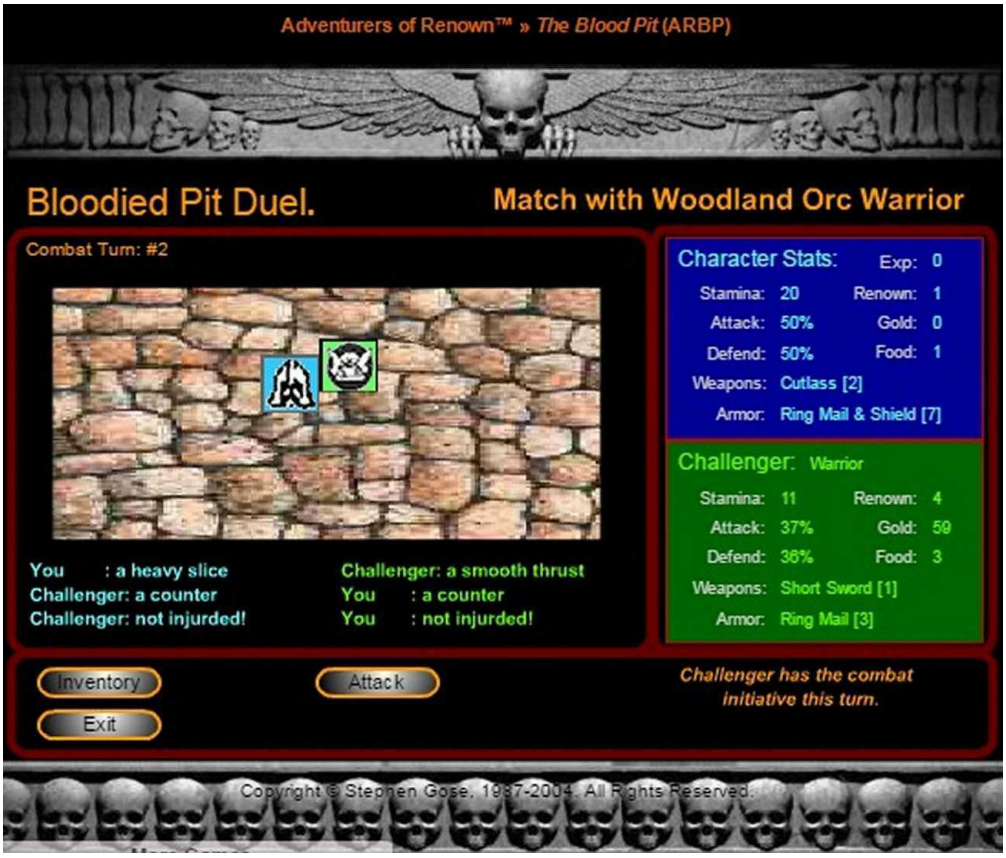
```
this.physics.add.collider(player, monster, bumpMonster, null, this);
```

[24] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11a.html

**Example 7.5: Grid-less Combat Movement Lines 81 - 292**

```
81      //Process conflict instead of "Game Over" phase
82      var combat = new Phaser.Class({
83          Extends: Phaser.Scene,
84          initialize: function combat(){
85              Phaser.Scene.call(this,{key:'combat'});
86              this.player;
87              this.monster;
88              this.attackButton;
89              this.fireButton;
90              this.label;
91              this.attacktxt;
92              this.firetxt;
93          },
94
95          preload: function preload(){
96              console.log(" %c\n Loading Combat phase. ",
97                              "color:white; background:green");
98              this.load.crossOrigin = 'anonymous';
99
100             //game background;static title and copyright
101             this.load.spritesheet('button',
102                     'images/spriteSheets/mmog-sprites-silver.png',
103                     {frameWidth:129, frameHeight: 30});
104         },
105
106     create: function(){
107         // ... similar to "mainExplore" class
108     }, //comma very important here
109
110     update: function(){
111
112         // ============
113         // Example 4.3: Movement Arrows Integration begins
114         // NOTE: combination arrow directions are now
115         //    possible with this format
116         // ============
117
118         player.body.velocity.x = 0;
119         player.body.velocity.y = 0;
120
121         if (cursors.left.isDown){
```

```
122             // if the left arrow key is down
123             player.body.setVelocityX(-speed); // move left
124         }
125         if (cursors.right.isDown){
126             // if the right arrow key is down
127             player.body.setVelocityX(speed); // move right
128         }
129         if ((cursors.down.isDown)){
130             player.body.setVelocityY(speed); // jump up
131         }
132         if ((cursors.up.isDown)){
133             player.body.setVelocityY(-speed); // jump up
134         }
135         // ============
136         // Example 7.4: Movement Arrows Integration & SPACE quites
137         // ============
138         // New keyboard listener
139         if(cursors.space.isDown){
140             // Option 1) go to
141             leaveCombat();
142         }
143         // Example 4.3: ends
144         // ============
145     });
```




**First round with Grid-less Combat & Narrative**

***Play Above Demonstration: https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1***

***Hint:*** Yes, we will optimize this code later. This is provided as only a quick demonstration review from chapter 4.

### Grid-ed Combat

Grid-ed Combat is similar to table-top board games such as checkers or chess. Using grid-ed gameboards add a new dimension to our conflicts, that is namely: ***maneuvers.*** We will create this gameboard using several methods such as ***"tiled-maps" OR the new "grid" features in Phaser III.***

## 5.5 Tactical Tiled-Maps

Tile-Maps, in my opinion, tends to abuse the "separation of concerns". Too many times have I seen tutorials, examples, and raw source code on GitHub combine the visual

display with its associated ***"metadata"***[25] in the game environment. ***Data is a subset of the "Game Mechanics" (GM) Component; it is not a subset of the "Game Framework Display" Mechanisms.*** By binding data into your sprites, you are locking yourself into a specific ***"Front-end Gaming Framework".*** I take the same approach with "Tactical Tiled-Maps" as I do with the gamers' Avatars — there is a "visual element" that is separate from its metadata elements. The metadata elements for game boards are, what I call, the "movement tables" (MT). Movement tables are a "super-set" to the Tiled-Maps spriteSheets (i.e., their visual display elements).

Approaching this topic, we are ***teetering***[26] on the edge of leaving this generic Game Prototype and entering into the realms of a specifically designed artwork theme. If we use a "generic" naming convention for our artwork files, it becomes a simple task to import new — and different — "game theme artwork" images with the same file names. This will overwrite the current game prototype box-artwork. We will do all this in later chapters. For now, we'll continue to use "rectangles" or "hexagons".

***Warning:*** Software such as ***Texture Packer,***[27] ***Adobe Animate*** (formerly called Flash CC)[28], or ***Shoebox***[29] all generate texture atlases, ***not sprite sheets.*** You must use the ***"`Loader.atlas`"*** instead. You can download a **FREE** Sprite Sheet Packer here[30].

## 5.6 Squares and Checkered Grids

Square-based tile-maps are the simplest game boards to create. We create boxes in the same fashion as we did for walls in chapter 1 — only these squares will be larger — to contain both our avatar and opponents. How big should these tiles be? We have several decisions to make:

- a grid-tile (aka cell) just big enough to hold a single avatar sprite. When the sprite's graphics are adjacent — "touching" another occupied square as we did in the previous chapter; they are engaged in "melee/hand-to-hand" combat.
- a grid-tile large enough to hold **both** our avatar ***and*** its opponent; this is similar to checkers or chess. They are engaged in "hand-to-hand" combat when both opponents occupy the same square space.

[25] https://labs.phaser.io/edit.html?src=src/components/data/change%20data%20event.js
[26] http://www.dictionary.com/browse/teetering
[27] https://www.codeandweb.com/texturepacker
[28] https://www.adobe.com/products/animate.html
[29] http://renderhjs.net/shoebox/
[30] https://www.codeandweb.com/free-sprite-sheet-packer

- should we create numerous individual 16 x 16 px backgrounds? Remember each object in a Scene must be processed every frame per second.
- or simply integrate the tiles into the single background image as I've done below?

**Example 7.6: Grid-ed Combat as individual background images**

```
1    //NEW! Grid Tile-Map configurations
2    var map;                //tile map as background
3    var layer;              //tile map layer
4    var tileSize =  64;     //twice avatar icon size? or same size?
5    var numRows = 4;        //adjustable for your game
6    var numCols = 4;        //adjustable for your game
7    var tileSpacing = 2;    //adjustable for your game
8    //var tilesArray = [];  //one way; thousand more to choose
```

**Example 7.7: Grid-ed Combat Squares *traditional method***

```
290    //New Combat Grid - generic square tiles
291    this.SQTilesFloor = this.add.group(); // optional staticGroup?
292    for(j=0;j<numRows;j++){
293       for(i=0;i<numCols;i++){
294        gameX = tileSize * i + tileSize/2 + tileSpacing;
295        gameY = tileSize * j + tileSize/2  + tileSpacing;
296        var tileGridSQ = this.add.rectangle(gameX,gameY,60,60,0x000000));
297        this.SQTilesFloor.add(tileGridSQ);
298       }
299    }
```

We still have one ***itty-bitty problem***[31] in our tiled combat encounter — the avatar is still sliding across those tile images of the checkerboard and ignores any movement restrictions. If your game requires turn-based limited movement, then we need to resolve this. We can solve this easily by modifying the `combat update()` section that monitors the player's movement input. Here are several solutions:

1. Each time an arrow key is pressed, we "jump" the avatar to the next grid-ed tile. We can also fix the bug of "the sliding avatar off the grid-field" by simply counting and storing the number of rows and columns (i.e.: locations on the grid) where the avatar currently is (i.e., metadata).

[31] https://www.merriam-webster.com/dictionary/itty-bitty

2. ***Or,*** we could place "invisible" walls around the grid to prevent the avatar from leaking outside the combat area. This ignores movement restrictions of the tiles.
3. ***Or,*** we could use a more traditional approach using tiles map layers and external JSON data files from the following Mozilla Developers' References below.
4. ***Or,*** we could simply place our avatar in every square and make it visible and invisible as determined by our movement path. ***An example of this method is here***[32].

**Pure JS Sample: Grid-ed Movement**

```
    //for example, snapping a grid coordinate of 43 to the nearest
    //multiple of the gride size:

    var dx = 43;        //distance of x from sprite or point.
    var gridSize = 32;  //assuming every grid square is 32px
    var columnX = Math.round((dx / gridSize) * gridSize);

    //This provides a snapped sprite column as the first column grid.
```

***Exercise:*** Download and study ***this source code***[33].

### *References from Mozilla Developers:*

- ***Square Grid Tile Maps samples:***[a] A collection of resources used by Mozilla developers for developers, technical evangelizing, and similar such content.
- ***JavaScript for game development:***[b] A compilation of materials to learn JavaScript and make HTML5 games.
- ***JS Game development examples for Tilemaps"***[c] Examples of tilemaps implementation with the Canvas API.
- ***HTML5 games workshop:***[d] A workshop that teaches how to develop HTML5 games with JavaScript and Phaser. It is meant to last a full day, although it includes sufficient guidance for people to finish it at home if only a short session with a coach is possible.

[a] https://github.com/mozdevs/gamedev-js-tiles
[b] https://github.com/mozdevs/js-for-gamedev

[32] https://makingbrowsergames.com/starterkits/jump2cap/Peg-Examples/trixAttacksMagix-Phaser/index-mobile-wctam.html#game
[33] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/group/#create-game-objects

[c] https://github.com/mozdevs/gamedev-js-tiles
[d] https://github.com/mozdevs/html5-games-workshop

## Deeper Dive: Phaser III Grids

Phaser III has a new feature that makes grids very easy to build. Here's a demonstration from ***labs.phaser.io and a plugin.***

- ***labs.phaser.io demo***[34]
- ***rexrainbow UI plugin***[35]

### *Grid Plugin*

var gridSizer = scene.rexUI.add.gridSizer(x, y, width, height, column, row);

* column : Amount of column grids.
* row : Amount of row grids.
* x, y : Position of gridSizer. Only available for top-gridSizer, children-sizers will be changed by parent.
* width : Minimum width. i.e. Width of this gridSizer will bigger then this value.
* height : Minimum height. i.e. Hieght of this gridSizer will bigger then this value.

Source code available here[a]

[a] https://github.com/rexrainbow/phaser3-rex-notes/tree/master/examples/ui-gridsizer

The new "Grid Shape" feature in Phaser III is actually a ***Game Object,***[36] and being such, you could add grids to any Scene, either inside ***Group(s)***[37] or ***Container(s).***[38] You simply treat it like any other Game Object in your game. You can even tween, scale, enable physics and input. See this example at ***labs.phaser.io.***[39] The "Grid" gives you

[34] http://labs.phaser.io/view.html?src=src/game%20objects/shapes/grid.js
[35] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/ui-gridsizer/
[36] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObject.html
[37] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Group.html
[38] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container.html
[39] http://labs.phaser.io/view.html?src=src/geom/rectangle/set%20empty.js

an easy way to render square shapes into your game(s) without using any textures, and furthermore, taking complete advantage of WebGL.

The Grid only supports color fills and cannot be stroked. But using clever grid spacing correctly, you could accomplish a similar effect with cell outlines. Grids are available only ***if the Grid Game Object was built into your Phaser framework. (Read more here)***[40]

You can control the size of the overall grid and the width and height of each individual cell. You can also set a fill color for each cell as well as an alternate color. When the "alternate color" is set, the grid cells naturally alternate into a checker-board displayed effect. Optionally, you can set an outline color as your border around each cell. This setting draws lines between the grid's cells. If you specify an outline color with an alpha of zero, then it simply draws the cells as spaced out.

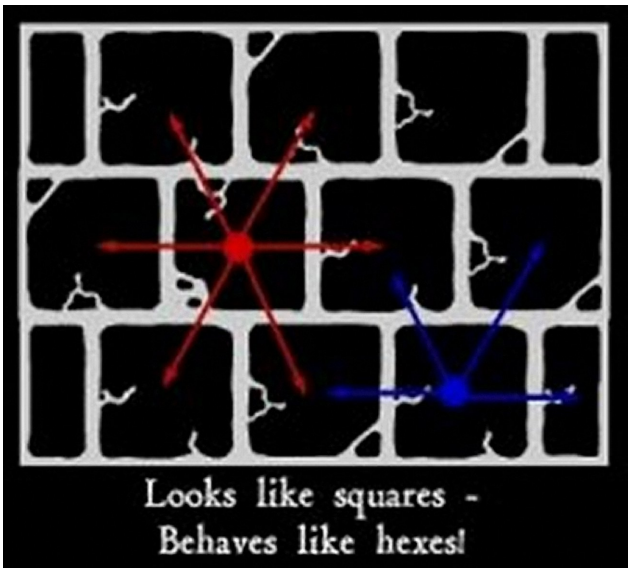
```
//https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Grid.html
new Grid(scene, x, y, width, height, cellWidth, cellHeight, fillColor, fillAlpha, outlineFill-
Color, outlineFillAlpha)
```

***Exercise:*** refer to https://rexrainbow.github.io/phaser3-rex-notes/docs/site/gridtable/ Grid Table plugin and ***sample code.***[41]

***Exercise:*** Using the Grid as a game construction tool! William Clarkson has developed a clever use of this feature. He uses the grid to align the placement of game objects during development. You can ***more about his strategy here.***[42]

[40] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObjectFactory.html#grid__anchor
[41] https://github.com/rexrainbow/phaser3-rex-notes/blob/master/examples/gridtable/gridtable.js
[42] https://phasergames.com/scaling-games-in-phaser-3/

## Hexagonal Grids




**Creating a Hexagonal Mega-Squares in a map grid**

Many war-game simulations from the 1960s to present-day use hexagonal game-board grids instead of square-based maps. Square-based grids share an edge with only four other neighboring squares; but, they also touch another four neighbors at just one point in the diagonal directions. This frequently compounds movement distance along grids since diagonal movements are harder to equate properly to cardinal directions. You are limited either to the four cardinal directions or eight cardinal directions with squares. However, with hexagons, you have a compromise of equidistant movement along with six directions. Hexagons don't touch any neighbor at only a point; movement to adjacent places are only across borders. Hexagons have a small perimeter-to-area ratio. Unfortunately, in our square pixel-screened world of computers, hexagons are harder to use. Amit J Patel[43] has collected some articles that may help you turn common square-grid algorithms into hex-grid algorithms.[44] Let me present the following resource for hexagonal grid maps.

### *Red Blob Games*

Hexagonal grids are used in some games but aren't quite as straightforward or common as square grids. I've been collecting hex grid resources for nearly 20 years and wrote this guide to the most elegant approaches that lead to the simplest code, largely based on the guides by Charles Fu[a] and Clark Verbrugge.[b] I'll describe the various ways to make hex grids, the relationships between them, as well as some common algorithms. Many parts of this page are interactive; choosing a type of grid

[43] http://www-cs-students.stanford.edu/~amitp/gameprog.html#hex
[44] http://www-cs-students.stanford.edu/~amitp/game-programming/grids/

will update diagrams, code, and text to match.

[a] http://www-cs-students.stanford.edu/~amitp/Articles/Hexagon2.html
[b] http://www-cs-students.stanford.edu/~amitp/Articles/HexLOS.html

**Example: Dynamically Created Hexagonal Grid - *traditional method***

```
1     //New Combat Grid - simplistic hexagon grid tiles using mega-squares
2     this.HXTilesFloor = game.add.group();
3     var hxOffSetY = 0;
4     var spacingX = tileSize * 0.75;
5     for(j=0;j<numRows;j++){
6         for(i=0;i<numCols;i++){
7             // odd columns are pushed down half a square
8             if ((i % 2) == 1){
9                 hxOffSetY = tileSize * 0.5;
10            }else{
11                hxOffSetY = 0;
12            }
13            gameX=tileSize*i+tileSize/2 +tileSpacing ;
14            gameY=tileSize*j+tileSize/2 +tileSpacing+hxOffSetY;
15            var tileGridHx = this.add.sprite(
16                        gameX+(config.width/2),gameY,
17                        box({length:60,width:60,color:'#333'}));
18            this.HXTilesFloor.add(tileGridHx);
19        }
20    }
```

The illustration above shows the hexagonal grid in an East-to-West orientation. The combat demo has the hexagonal grid in a North-to-South orientation. Adjusting for either is a simple matter of ***"off-setting"*** either the rows or columns by "half an area". The example above uses a "modulo division" to learn if we are laying down an "odd row or column" or an "even row or column".

## Deeper Dive: Real hexagonal grids

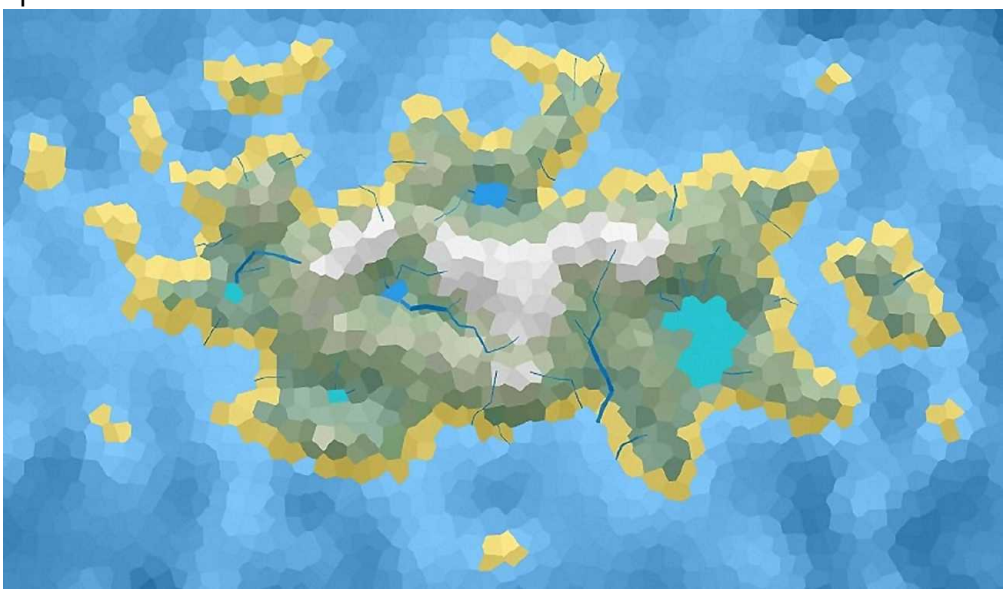
As you may have already guessed, "rexrainbow" has a wonderful Phaser III plugin that creates ***hexagonal grids.***[45] in either North-South or East-West orientations.

[45] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/board-hexagonmap/#hexagon

```
var tileXYArray = scene.rexBoard.hexagonMap.hexagon(board, radius);
var out = scene.rexBoard.hexagonMap.hexagon(board, radius, out);
```

### Squishes

This is just a short introduction. Squishes are mixed polygonal areas; they restrict movement into adjacent areas only and neighboring areas are not predictably arranged as grids are. In my opinion, squishes are better used for strategic war-games instead of tactical encounters. But then, game developers are creative types, and I may have to ***"eat my words"***[46] at some point to come. If you would like to see a Phaser Plugin for squishes ***download from here***[47]. Here's an illustration of squishes from the Phaser Squish Generator:



**Phaser Squish Plugin Map generator**

## 5.7 Rules of Engagement: Take 5 paces, turn, and ...

### Been there ... done that ...

The history and evolution of the ***"tactical role-playing game"*** (aka "TRPG") is a game genre that incorporates elements of traditional role-playing games and emphasizes

[46] http://idioms.thefreedictionary.com/eat+words
[47] http://luckylooke.github.io/phaser-islandjs-plugin/

low-level tactical combat rather than high-level strategic gameplay. Tactical RPGs ***tend not to feature multi-player play,*** and a distinct difference between tactical RPGs and traditional RPGs is the ***lack of exploration.*** Later, we will introduce this high-level strategic RPG play. In Japan, these games are typically known as "Simulation RPGs" (□□□□□□□□ RPG?, abbreviated as SRPG).

***Note: Ruins of Able-Wyvern™ (ARRA)*** has included both tactical combat and strategic roleplay since 1993.

***Learning the ropes***[48] about RPGs will provide us the ***"who's done what and why it was successful."*** To gain this foundational knowledge, do the following three exercises to understand why I selected these forms of combat resolution.

***Exercise:*** 1. Research other combinations on ***conducting combat here.***[49]
***Exercise:*** 2. Research Controversy and Criticisms of preferred ***conflict resolution across cultures here.***[50]
***Exercise:*** 3. Explore ***what has happened with RPG and what gamers currently expect.***[51]

## 5.8 "Where's the beef?"

How do we conduct combat and resolve a duel between two (or more) antagonists? All combat systems boil down to just a ***few basic questions:***[52]

- What are the chances an avatar has to strike an opponent successfully?
- If a successful strike occurs, how much injury was inflicted? and finally,
- What might other events happen to the participants?

It's time to write some code for various forms of combat resolution and then add some dynamic menu. ***Here are six (6) different ways to resolve combat:***

[48] http://idioms.thefreedictionary.com/learn+the+ropes
[49] https://en.wikipedia.org/wiki/Role-playing_video_game#Combat
[50] https://en.wikipedia.org/wiki/Role-playing_video_game#Cultural_differences
[51] https://en.wikipedia.org/wiki/Role-playing_video_game
[52] http://www.roguebasin.com/index.php?title=Thoughts_on_Combat_Models

- ***Click fest!*** Originally deployed in ***Ruins of Able-Wyvern™ (ARRA)*** and ***Blood Pit™ (version 1).***[53] Each click is a separate combat turn with attack and defense.
- ***"Guitar Hero"*** style combat — as modified by my game series ***Red Fountain Swordsman™.*** This is an interesting version of combat that ties directly into a player's personal coordination — refer back to ***GM Skills*** mechanics. This combat-style is your choice if you're interested in limiting the advancement of an avatar to its owner's natural capabilities. You can find this combat style in the Bonus Content download file in the Bonus Games directory.
- ***"Drama Theater"*** as seen in so many ***online games currently;***[54] acting out the attacks as in ***"Street Fighter".***[55] This combat system is heavily dependent on artwork animation.
- The ***Society for Creative Anachronism***[56] ***virtual trainer, game design by Steve Echols***
- ***"En Guard!"***[57] a rival to D&D and still wildly popular today.
- ***Yeap! Ya betcha' 'ur life!*** — a "never-before-seen" combat system that "gambles" on avatars. You can find this combat style in the Bonus Content download file in the Bonus Games directory.

## Click-fest

***Ruins of Able-Wyvern™ (ARRA)*** Gaming System originally had a single "Fight" button. Each click on this button represented one combat game-turn round — an exchange of offensive and defensive moves per each antagonist. Later, I migrated the game to the dynamic menu-style you've seen earlier in this chapter. If antagonists are "touching" then an "Attack" button becomes available for melee combat. If antagonists are "not touching" then only missile weapons are used.

In addition to these "combat buttons", other tactical menu options become available. The first of these actions is "Exchange" weapons. Clicking this button allows an avatar to switch weapons in anticipation of either "hand-to-hand" grappling or disengaging from their opponent while anticipating missile combat. The other combat options are defensive in nature and permit the combatants to focus all their attention on parrying, blocking in melee combat, or "dodging" incoming missiles. These combat options appear dynamically according to the current combat situation using a "finite state machine" (FSM).

[53] http://localhost/_GIS/GISUS-MakingBrowserGames/makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/
[54] https://www.battleon.com/aq-play.asp
[55] http://gamequeryjs.com/
[56] http://www.sca.org/
[57] https://en.wikipedia.org/wiki/En_Garde!

**Pure JS Combat Finite State Machine**

```
49      // Design notes: switch seems to be faster than the if statement combat.
50      // This is original "switched" version rv_8
51      // ====================================
52      // Menu Finite State Machine  (as Global FSM - Ch. 8)
53      // ====================================
54      // Default is English; easily replace with native language text.
55      // Also used as simple text buttons with hit area.
56      var menu = [[]];         // [gameStateNdx] [available menu options]
57      menu[0][0] = "Exit";     // always available
58      menu[0][1]="Disbelieve"; // always available; a simple magic spell for everyone
59      // "disengaged" game state index
60      menu[0][2] = "Dodge";    // available during "disengaged"
61      menu[0][3] = "Fire";     // available during "disengaged"
62      menu[0][4] = "Throw";    // available during "disengaged"; throw single-handed w\
63  eapon
64      menu[0][5] = "Spell";    // available during "disengaged"; cast magic spell
65
66      // "engaged" game state index;
67      menu[1] = ["","Attack","Defend","H-2-H","Disengage","Exchange"];
68      // alternate format
69      //menu[1][0] = "";         // available during "melee"
70      //menu[1][1] = "Attack";   // available during "melee"
71      //menu[1][2] = "Defend";   // available during "melee"
72      //menu[1][3] = "H-2-H";    // available during "melee"
73      //menu[1][4] ="Disengage"; // available during "melee"
74      //menu[1][5] = "Exchange"; // available during "melee"
75
76      // pre & post combat ( i.e., "disengaged" ) game state index;
77      menu[2] = ["","Cook","Inventory","Search","1st Aid","Learn"];
78      //  alternate format
79      //menu[2][0] = "";          // always available
80      //menu[2][1] = "Cook";      // available pre or post combat
81      //menu[2][2] ="Inventory";  // available pre or post combat
82      //menu[2][3] = "Search";    // available post combat
83      //menu[2][4] = "1st Aid";   // available during "melee" as potions?
84      //menu[2][5] = "Learn";     // available post combat; gain experience
```

As mentioned earlier, combat has an engaged, disengaged status, and a pre- or post-combat state. It becomes a simple matter to use a 2D array. The first dimension — the columns — is the "combat status sentinel" of engaged, disengaged, or pre/post-combat and the rows hold the various commands a player will issue inside that status.

Separating our commands in this way always helps us in developing Massive Multi-player online Games (MMoG) and placing our HUD options in various containers.

A "finite state machine" (FSM) is truly invisible to the gamers as they should ***"Pay no attention to that man behind the curtain." as the Wizard of OZ once said.***[58] We will study further details about the FSM and recursive feedback in later chapters. Suffice it to say for now, that a single click launches several algorithms:

- ***_combatRound:*** Is the administrative controller. It determines who goes first in this round of combat. This combat initiative could be modified by previous injuries, emotional predigest, winning, or losing. This function then calls the combat round narrative.
- ***_combatNarrative:*** Is the work-horse for the combat turn. It generates a unique narrative that occurred during this single combat action. It generates the random events, compares combat skills with a helper function, evaluates the success or failure of each antagonists' attack and defense, and lastly calculates physical damage imposed upon the body or equipment (i.e., damage to weapons or shield/armor). All this from one simple click of a button.

***Exercise: Download and study the*** `combat.js` ***in the online Source code Appendix.***[59]

## Guitar hero - Time to get it Right!

- Red Fountain Swordsman (play the original flash game here)[60]: Take the same martial-arts classes in swordsmanship as the Winx club specialists of Red Fountain. Practice your sword skills and timing to become the perfect swordsman specialist like Brandon and Sky.
- ***Play the Phaser demonstration here.***[61]

***Note:*** There's a similar game to Guitar-Hero constructed in Phaser. I encourage you to download and review their GitHub **OOP** source code.[62]

[58] https://medicalxpress.com/news/2017-01-attention-curtain-human-brain-important.html
[59] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/js/state/combat.js
[60] https://www.renown-games.com/winx/red-fountain-swordman/index.html
[61] https://makingbrowsergames.com/book/_rfs-Phaser/
[62] https://github.com/PBMCube/banjo-hero

The Guitar Hero was a series of musical-rhythm games first published in 2005 by RedOctane and Harmonix, and distributed by Activision, in which players use a guitar-shaped controller to simulate playing guitar. In the original Guitar Hero game, a player tried to press the correct guitar string at the correct time. It was a game that used "Timing Elements"[63] tied into the gamer's personal coordination skills — refer to ***Chapter 5 GM Skills.*** It mimicked many features of an actual guitar, including fast-fingering riffs, "hammer-ons", pull-offs, and a "whammy bar" to alter the notes' tones. The game was transcribed into Adobe Flash[64], from which came my idea for this version of combat — the ***Red Fountain Swordsman*** game.

***Hint:*** This complete source is available only in the Bonus Download content. Developer's Demo located at https://makingbrowsergames.com/book/_rfs-Phaser/

## Days of our Lives - Drama Theater

Conflict — as seen in so many current online games — acts out the conflict as an animated movie such as that seen in "Street Fighter". These conflict scenes solicit tactics from the gamer then act out the chosen strategies compared to the antagonists'. The best game, in my opinion, that demonstrates this, is "Adventure Quest"[65]. I had many visits and exchanges with them in their earlier years at the turn of the millennium (2000 - 2001); AQ sky-rocketed when they hired a professional artist years later to support their online efforts! Their groundbreaking methods have become the "bread and butter"[66] of today's RPG combat as seen in this online course below which mimics their combat style.

[63] http://www.roguebasin.com/index.php?title=Time_Systems
[64] http://guitarflash.com/
[65] http://www.battleon.com/
[66] http://dictionary.cambridge.org/us/dictionary/english/bread-and-butter


**Sample of Single Page Combat from Zenva**

- ***Single Page Combat from Zenva RPG Online Course***[67] FREE Source Code available in this excellent tutorial.
- ***Advanced Phaser 3 – Build an RPG***[68] Master advanced skills in HTML5 Game Creation as you build an RPG

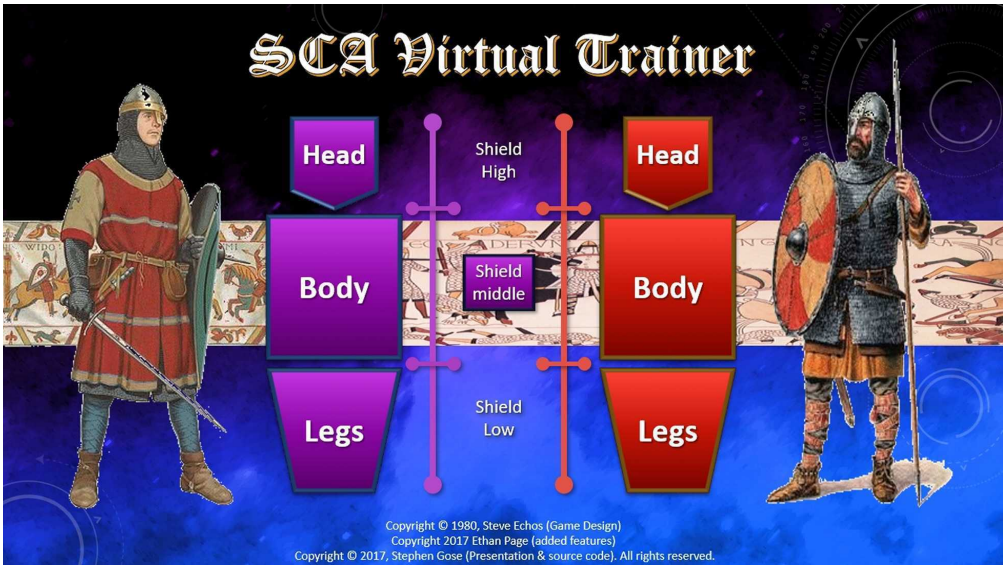
## SCA Virtual "Fighter Practice" by Steve Echos

> The Society for Creative Anachronism is an international organization dedicated to researching and re-creating the arts and skills of pre-16th-century Europe. Their "Known World" consists of 20 kingdoms, with over 30,000 members residing in countries around the world. Members, dressed in clothing of the Middle Ages and Renaissance, attend events which feature tournaments, royal courts, feasts, dancing, various classes & workshops, and more.

One of my life-long friends, Steve Echos, introduced me to a game he invented to train S.C.A. warriors in their live combat. He has graciously allowed me to publish his system for your enjoyment.

[67] https://academy.zenva.com/course/rpg-game-development-with-phaser/?a=47&campaign=Phaser3GamePrototyping

[68] https://academy.zenva.com/product/advanced-phaser-3-build-an-rpg/?a=47&campaign=Phaser3GamePrototyping




**Rock, Paper, Scissor in a deadly combat system**

***How it works: "premeditate and execute".***

The game is a simple "rock, paper, scissors" style of combat; but is extended into both hands — a weapons hand and a shield hand — instead of using the traditional one hand.

1. Each player selects where to defend their body with their shield hand.
   * A "Norman" shield will protect 2 adjacent locations. For example, a player with a Norman shield protects "high" then both their "head" and "body" are protected while their legs are exposed to any attacks.
   * A "Saxon" shield only protects 1 area while the adjacent areas are exposed to attack.
2. Each player selects where to attack their opponent as: "high" (the head), "middle" (the body), or "low" (the legs).
3. Players reveal their choices; a successful hit on the head or body will kill their opponent's avatar. A hit on the legs is crippling; the player cannot move.

This is a wonderful combat system for online multi-player or table-top gaming. For a single-player game, it is a simple matter of listing all the possible combat actions for the computer's AI; and then, perform random (or semi-intelligent) actions. ***Download the pseudocode and flowchart.***[69] I encourage you to combine this "SCA Virtual Trainer" with this interesting game called ***Color ZAP (book)***[70] by ***William Clarkson***[71] for a single-player combat system.

[69] https://makingbrowsergames.com/p3gp-book/p3-sca/code2flow_rps.pdf
[70] http://amzn.to/2njpDQn
[71] https://williamclarkson.net/courses/phaser-basics/

**SCAVT game: lines 292-318**

```
292     // Optional Game Turn Results (GTR) display formatting
293     // 1) Place GTR on a modal JavaScript alert pop-up
294     // - alert(composedNarration);
295     // - use jQuery ui ...
296     //      function showPopUp ( ) {
297     //          //JQuery method to call the modal in Semantic UI.
298     //          $('.ui.modal').modal('show');
299     //      }
300
301     // 2) Place GTR on a modal Phaser Text pop-up with a continue button.
302     // Others solutions are jquery, but the canvas focus will be lost.
303     // it's better to place HUD in layers, containers, or groups then
304     // change the visibility on or off (simply without animations).
305
306     // 3) Place GTR on a sliding HUD panel on either side
307     //      https://jqueryui.com/
308     //      https://codyhouse.co/gem/css-slide-in-panel
309     //      https://codepen.io/jasesmith/pen/raqBpm
310     //      http://wowslider.com/html5-slider-sunny-fade-demo.html
311     //      https://davidwalsh.name/css-slide
312
313     // 4) Place GTR in a 2-column table or single column on each side
314     // 5) Place GTR into a historical logging journal
315     // - only reset journal upon new game sessions
316     // - email the journal or preserve it:
317     //      - in localStorage permanently or
318     //      - sessionStorage per game play.
319     //  var emailLog += composedNarration;
```

- ***Demonstration single-player game here (v3.16+)***[72]
- ***Demonstration multi-player game (v3.16+)***[73]

## En Guard method

En Guard was a rival to Dungeons and Dragons (D&D) and still wildly popular today. Launched in 1975, it provided an "alternate RPG combat system" using tactics rather than its "dice-rolling contender" D&D. The ***En Guard combat system*** is a ***Queued Turn***

[72] https://makingbrowsergames.com/p3gp-book/p3-sca/
[73] https://leanpub.com/rrgamingsystem

***System (explained in detail here)***[74] — or a ***premeditate and execute.*** In this approach, a gamer selects an attack option (cut, slash, lunge); each option is broken down into "fix segments of time" to perform that action. For example, a "Cut" would require 4-time segments: ***x-C-x-x.*** As you see, the "cut" happens in the 2nd time segment, the "x" are movement actions into and out from that single maneuver. A Player has ***12-time segments (TS)*** per combat round; this "cut" action consumed 1/3 of the time so the player could select other tactical options.

**En Gard Combat Turn and timing segments**

| Turn | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | X | C | X | X | X | P | X | X | C | X | X | X |
| B | X | P | X | X | C | X | X | X | C | X | X | X |

- C = Cut (X-C-X-X)
- P = Parry (X-P-X)
- X = Rest or time to maneuver

Two antagonists are dueling. In time segment (TS) #2 B successfully parried A's cutting attack. In TS #5, B's attack was successful since A's parry came too late. Both combatants struck simultaneously in TS #9.

This is a wonderful combat system for online multi-players or table-top gaming. For a single-player game, it is a simple matter of listing all the possible combat actions for the computer's AI; and then, perform random (or semi-intelligent) actions.

The ***EnGard game system. Product info is available here.***[75]

## Yeap! Ya betcha' 'ur life!

This is an interesting twist on combat systems since a gamer is ***"betting their skills and avatar's life in the process".*** So why not turn the combat system into a "gambling" game such as a "slot machine"?? There are GA-zillion examples for "slot machines with the Phaser Framework"; all you would need is some interesting artwork. Each lever pull could spin up the attacks and defenses for both attackers and antagonists. The various combinations would translate into the combat rounds results. Studying the up-coming chapter on Artificial Intelligence, you could skew each "slot machine pull" using a probability table. Refer to my favorite author — Emanuele Feronato — for source code and more ideas.[76]

[74] http://www.roguebasin.com/index.php?title=Time_Systems
[75] https://amzn.to/2Ncimw6
[76] http://www.emanueleferonato.com/2010/04/13/17-jquery-powered-web-games-with-source-code/

## 5.9 Story narrative

Nothing is more compelling than a background story explaining why your hero is questing. The ***Ruins of Able-Wyvern™ (ARRA)*** and the ***Rescue of NCC Pandora™ (ARNCCP)*** Gaming System (source code in the appendix) provides such a narrative during combat and for quests.


**Randomly Generate background stories**

***Hint:*** This complete source is available in the online ***Source code Appendix***[77].

[77] https://makingbrowsergames.com/book/index13.html#13.3

## 5.10 Frisking, Fondling, or Groping


**Our Avatar wins in Combat & Salvages rewards**

After a combat encounter, our avatar has the option to search and rescue booty.

## 5.11 Chapter Source Code

https://makingbrowsergames.com/book/index.html

***Book Combat demo (online)***[78]

***Hint:*** Use the Developer's Console to study the internal game operations.

[78] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/

## 5.12 Complete Combat Prototypes

We have moved the Source Code Appendix onto the supporting website and removed it from the 1st to 6th editions. This allows us to update code changes dynamically for Phaser v3.x.x as it matures.

- ***Combat Systems Game Prototype Library***[79]
- ***Chapter 7 online Examples***[80]

## 5.13 Summary

Here's what we've fought for thus far...

- Learned the 4 virtues of a good tactical combat system.
- Separated Conflict spacial aspects;
- Developed various modes of combat: ranged, melee, and hand-to-hand.
- Provided gamers with correct weapon usage in various combat modes.
- Created a dynamic menu responding to the current state of conflict.
- Analyzed the Phaser Weapons function.
- Researched the 3rd Person missile demo.
- Discovered how to juice up games.
- Developed two distinct tactical movement styles and matching tile-maps.
- Contain combatants with a combat arena.
- Programmed several inputs to control a player's avatar.
- Researched the Grid-less Combat demo.
- Develop tactical maneuvers as an added feature.
- Researched the Grid-ed Combat demo.
- Discovered the proper "separation of concerns" for Tiled-Maps.
- Developed movement tables as a super-set of Tiled-Maps.
- Deployed square grids for various moves and combat engagement.
- Deployed hexagonal grids with either a vertical or horizontal orientation.
- Researched the Hex Grid Combat demo.
- Deployed squishes as a movement and tile-map.
- Discovered how to import new graphics art under the "separation of concerns" game prototyping.
- Found resource tools to develop movement tables and Tile-Maps.
- Analyzed the difference in tile-map tools and importing into Phaser.

[79] https://makingbrowsergames.com/p3gp-book/index-combat.html
[80] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/

- Examined various Mozilla Game Developers References for Tiled-Maps creation.
- Examined various Red Blob References for square and hexagonal Tiled-Maps creation.
- Researched 6 different conflict systems.
- Studied popular conflict systems across various ethnic and cultural groups.
- Analyzed the "Click-fest" finite state machine.
- Learned about minimum button sizes for mobile games.
- Adapted the "Guitar Hero" style of conflict.
- Discovered the Society for Creative Anachronism and their "Virtual Fighter Practice" combat system.
- Discovered "En Guard" queued turn system.
- Learned to adapt and innovate gambling games into a combat system.
- Discovered how to enhance conflict systems with narrative storylines.

## 5.14 Footnotes

- To preserving CPU processing and battery, pre-calculate math formula. Refer to ***sine and cosine here***[81]. ***One radian equals***[82] 180°/ π = 57.30°. Use this ***online calculator***[83] to help reduce the CPU workload.
- T, A. (2017, January 17). Finger-Friendly Design: Ideal Mobile Touchscreen Target Sizes - Smashing Magazine.[84] Retrieved May 19, 2017,
- EnGard game system. Product info[85]
- Color ZAP (book)[86]
- Phaser Plugin for squishes.[87]
- Hex grid resources[88]
- References from Mozilla Developers: [89]
- Square Grid Tile Maps samples[90].
- "The Four Virtues of a good tactical turn-based combat system"[91]
- JavaScript for game development[92]
- JS Game development examples for Tilemaps[93].
- HTML5 games workshop[94]
- How to write a Rogue-like game in 15 Steps[95]

[81] http://www2.clarku.edu/faculty/djoyce/trig/cosines.html
[82] https://ee.stanford.edu/~hellman/playground/hyperspheres/radians.html
[83] https://www.rapidtables.com/calc/math/Cos_Calculator.html
[84] https://www.smashingmagazine.com/2012/02/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/
[85] http://www.engarde.co.uk/useful.html#Top
[86] http://amzn.to/2njpDQn
[87] http://luckylooke.github.io/phaser-islandjs-plugin/
[88] http://www-cs-students.stanford.edu/~amitp/gameprog.html#hex
[89] https://github.com/mozdevs
[90] https://github.com/mozdevs/gamedev-js-tiles
[91] http://sinisterdesign.net/12-ways-to-improve-turn-based-rpg-combat-systems/
[92] https://github.com/mozdevs/js-for-gamedev
[93] https://github.com/mozdevs/gamedev-js-tiles
[94] https://github.com/mozdevs/html5-games-workshop
[95] http://www.roguebasin.com/index.php?title=How_to_Write_a_Roguelike_in_15_Steps

- Rogue Basin articles on combat[96]

[96] http://www.roguebasin.com/index.php?title=Articles#Combat_2

# 6. Game Mechanism Components

This section was in the former Phaser III Game Design Workbook (5th edition), and is preserved here as I update that book to its new 6th edition and this book's content.

## 6.1 Phaser III inline script - Reviewed

**Phaser III Game Mechanics in *main.js***

```
    // ================================================
    // ------------------------------------------
    // Main game Handler methods
    // ------------------------------------------
    //**TODO**:
    //        refactor and adjust for your game deployment
    //        remove console debug information on public deployment
    // ================================================
    main: function(){

        //Phaser v3.16+
        game = new Phaser.Game(config);
        console.log("Game prototype (Phaser.Game): Ext? "+Object.isExtensible( Phase\
r ));
        //console.log(Object.values(Phaser));
        //console.log(Object.getPrototypeOf(Phaser));
        console.log(Object.getOwnPropertyDescriptors(Phaser));

        // add all game states
        for( var stateName in window.GAMEAPP.state ){
            console.log("Crnt State: "+stateName);
            game.scene.add(stateName, window.GAMEAPP.state[stateName]);
        }

        //Dynamically assigned Game Mechanics from JSON Data file.
        gameMechanics();

        //move to next game phase
```

```
29          console.log("Leaving GAMEAPP.main -> boot");    //debug
30          game.scene.start('boot');
31      },
```

View the entire annotated **"main.js" here.**[1]

Download your v3 Project Template and ***demo***[2] from ***https://makingbrowsergames.com/p3design/v3.15.x-standardTraditional.zip***

## Phaser v2.x.x inline script - Reviewed

```
1       var gameWidth = 1024;  //golden ratio set-up
2       var gameHeight = 640;
3
4       //create an "instance" of our Phaser Game framework to use and call
5       // from within our game.
6       var game = new Phaser.Game(
7                   gameWidth,
8                   gameHeight,
9                   Phaser.AUTO,
10                  'game');
11      game.state.add('Boot', BasicGame.Boot);
12      game.state.add('Credits', BasicGame.Credits);
13      game.state.add('Game', BasicGame.Game);
14      game.state.add('GameOver', BasicGame.GameOver);
15      game.state.add('Languages', BasicGame.Languages);
16      game.state.add('MainMenu', BasicGame.MainMenu);
17      game.state.add('MoreGames', BasicGame.MoreGames);
18      game.state.add('Preloader', BasicGame.Preloader);
19      //Now pass control over to the Boot state.
20      game.state.start('Boot');
```

## Adding Display objects

Game Objects — with its texture, animation capability, input events, and physics — are called "sprites". Sprites are an indispensable component in your game; they are used

[1] https://makingbrowsergames.com/p3design/project-starterKit-demo/js/main.js
[2] https://makingbrowsergames.com/p3design/project-starterKit-demo/

for nearly everything that gamer sees. In contrast, an image, in the Phaser JavaScript Framework, is a "lighter" Game Object with a texture and input but does not have attached physics reactions nor animation handlers.

Sprites are computer graphics that are moved around the screen or otherwise manipulated as a single entity by the gamer input controls (aka mechanisms). It is the player's representation (aka avatar) in the game's activities. At its most basic composition a Sprite consists of:

- a set of coordinates and a texture that is rendered to the canvas.
- contain additional properties
  - allowing for physics motion (via Sprite.body),
  - input handling (via Sprite.input),
- events (via Sprite.events),
- animations (via Sprite.animations), and
- camera culling and more.

Phaser III example
this.<displayNameAssigned> = this.add.<Phaser-object-types>(this, x,y,key,frame)[a]

Phaser v2.x.x example
this.<displayNameAssigned> = game.add.<Phaser-object-types>(this, x,y,key,frame)[b]

[a] http://labs.phaser.io/edit.html?src=src/game%20config/multiple%20game%20instances.js
[b] http://phaser.io/docs/2.6.2/index#display

`game.add`, from the `GameObjectFactory` library, is the quickest way to create common game objects. These newly created objects are automatically attached to their appropriate "Manager", "World", or manually specified parent "Group". Phaser III currently supports different ***objects***[3] through the Game Object Factory compared to the former v2.x.x.

***NOTE:*** Refer to Phaser API documentation for the various display objects. And, review Part 2 - Loading Assets[4] in the official Phaser Tutorial Making your first Phaser game[5].

All sprites are typically loaded for each game phase during the "Boot", "Preload" game phases, or inside each scene's "preload essential function". Adding visual game pieces

[3] https://phaser.io/phaser3/contributing/part5
[4] http://phaser.io/tutorials/making-your-first-phaser-3-game/part2
[5] http://phaser.io/tutorials/making-your-first-phaser-3-game

should appear in the internal “pre-load” functions; placing the game pieces onto the game stage should appear in the internal “create” functions of each Phaser III scene.

**Phaser III example**

```
    "use strict";
    /** game load assets **/
    var load = new Phaser.Class({

        Extends: Phaser.Scene,

        initialize: function load (){
            Phaser.Scene.call(this, { key: 'load' });
        },

        preload: function (){
            console.log("Entering load -> preload");        //debug
            this.load.image('loadScene', 'assets/images/loadScene.jpg');
            // we have preloaded assets required for Loading group objects
            //        from the Boot state.
        },

        create: function(){
            // loading has finished - proceed to where? demo state? languages?
            this.add.image(0, 0, 'loadScene').setOrigin(0);
            //this.input.once('pointerdown', function () {
                console.log('From load to language');
                this.scene.start('language');

            //}, this);
        }

    });
```

***Exercise:*** Download this ***skeleton scene***[6] file as your template.

Adding Player(s) and Opponent(s) sprites will follow similar mechanisms. Static elements, those having a fixed position, would use images. Now that the sprite sheet is preloaded, place the sprite in the canvas by changing the “create function” this way:

[6] https://makingbrowsergames.com/p3design/project-starterKit-demo/gamePhase-skeleton-class-template.js

**Phaser v2.x.x example**

```
    preload: function(){
        //I want physics and animation handlers
        this.sprite = game.add.sprite(200, 150, "spriteName");
        //no physics nor animation handlers present
        this.rock = game.add.image(100, 100, "imageName");

        //this has animation frames.
        game.load.spritesheet(
                "spriteName2",          //assigned variable name
                "spriteGraphics.png",   //graphics file name
                spriteWidthSize,        //dimensions
                spriteHeightSize
            );
        },
```

Adding Player(s) and Opponent(s) sprites will follow similar mechanisms. Static elements, those having a fixed position, would use images. Now that the sprite sheet is preloaded, place the sprite in the canvas by changing the "create function" this way:

**Phaser v2.x.x example**

```
    create: function(){
        //  The difference between an Image and a Sprite
        //is that you cannot animate nor add physics to an Image
        game.add.image(
                100,    //x coordinates
                100,    //y coordinates
                "rock"  //your assigned variable name
        );
    }
```

***Hint:*** You could download and create all your game tokens and place them in the "wings" of your stage — just as actors waiting to enter a stage play. The hidden secret is `game.make`. It is the quickest way to create common game objects ***without adding them to the game world display!*** Phaser currently (as of v2.6.2) supports 20 objects[7] through the Game Object Creator.

[7] https://phaser.io/docs/2.6.2/Phaser.GameObjectCreator.html

***Exercise:*** Create an add sprite/image for each control mechanism on your main menu scene.

**Phaser III Example: Creating for Credits Scene**

```
1    var credits = new Phaser.Class({
2
3        Extends: Phaser.Scene,
4
5        initialize: function credits (){
6            Phaser.Scene.call(this, { key: 'credits' });
7        },
8
9        preload: function (){
10           this.load.image('creditsScene', 'assets/images/creditsScene.jpg');
11       },
12
13       create: function (){
14           this.add.image(0, 0, 'creditsScene').setOrigin(0);
15           this.input.once('pointerdown', function () {
16               console.log('From credits to menu');
17               this.scene.start('menu');
18
19           }, this);
20       }
21
22   });
```

***Exercise:*** Create an add sprite/image for each control mechanism on your main menu scene.

***NOTE:*** Refer, as we did earlier, to your ***Bonus Content***[8] /MMM-js-v0001/jsrc/Boot.js and Preloader.js files. I am using a button atlas to manage my "multi-state" buttons.

[8] https://makingbrowsergames.com/p3design/bonusDownLoads/MMM-JSource.zip

***Hint:*** You could download and create all your game tokens and place them in the "wings" of your stage — just as actors waiting to enter a stage play. The hidden secret is `game.make`. It is the quickest way to create common game objects ***without adding them to the game world display!*** Phaser currently (as of v2.6.2) supports 20 objects[9] through the Game Object Creator.

You could create your own sprites; but, dozens of websites that offer "royalty-free" graphics. It's your choice. ***#1 and #2 below are my strongest recommendations***; the following recommendations are offered in alphabetical order only with no preference suggested.

1. GameDevMarket.net[10] Collection of Music, Sound effects (sfx), 2D/3D/GUI Art.
2. http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/
3. http://spriteme.org/
4. ***GUI game kits***[11]

- http://hasgraphics.com/free-sprites/
- http://opengameart.org
- http://spritedatabase.net/
- http://tsgk.captainn.net
- http://untamed.wild-refuge.net/rmxpresources.php?characters
- http://www.bogleech.com/games.html
- http://www.cgtextures.com/
- http://www.gameartguppy.com/
- http://www.hellsoft.net
- http://www.lemog.fr/lemog_textures/index.php
- https://lostgarden.home.blog/tag/free-game-graphics/
- http://www.makeflashgames.com/tutorialshtml5/draw-image.php
- http://www.pygame.org/wiki/resources
- http://www.retrogamezone.co.uk/
- http://www.rpg-palace.com/visual-resources/tilesets-rmxp
- http://www.spiralgraphics.biz/packs/
- http://www.spriteland.com/
- http://www.spriters-resource.com/
- http://www.sprites-inc.co.uk/
- http://www.videogamesprites.net/
- https://www.makegameswith.us/academy/art/set
- https://www.pinterest.com/eduardoonguard/free-game-sprites-and-assets/
- https://www.scirra.com/forum/kenney-s-free-assets-20-000-assets_t93518

[9] https://phaser.io/docs/2.6.2/Phaser.GameObjectCreator.html
[10] https://www.gamedevmarket.net?ally=GVgAVsoJ
[11] https://craftpix.net/categorys/2d-game-kits/?affiliate=112238

***Warning:*** Be selective! Be careful to match similar themes, artistic style, and palettes when choosing "royalty-free" artwork. The hours you spend — ***time IS MONEY*** — trying to make everything match up might have been spent better as a cash transaction to an artist for hire. Always verify the quality requirements you need; and more importantly, whether the graphics are offered as a non-commercial or commercial license for game projects. Many artists seek projects that recognize their contributions.

***Exercise:*** Review the entire list(!!!) and select (write down!) your preferences for your game's theme. If your selection is non-commercial, contact the artist anyway and explain what you would like to do. Many artists are ***very*** eager to display their works. It's another "feather in their bonnets" ... ***negotiate!***

## Adding Control Mechanisms

In Phaser III, an ***"Image"*** Game Object is a light-weight static graphics in your game, such as logos, background scenery, or any other ***non-animated*** elements. Images can accept input events and have physics bodies that could perform tweens, tints, and scroll across the stage. The primary difference between an ***"Image"*** and a ***"Sprite"*** is that you cannot animate an ***"Image"*** — Images do not have an Animation component.

The "Input Manager" is the control mechanism for all types of player inputs such as the mouse, keyboard, touch, ***"MSPointer",*** and all input sub-systems. The Phaser "core game loop" will update its Input manager automatically.

- When an input device is "just pressed" or "just released", the minimum default sampling is ***200 milliseconds.***
- When an input (eg. Keyboard, Mouse, Touch) is "enabled", it is processed provided that each element is also enabled.
- When not enabled, ***all input sources are ignored.*** To disable just one specific input type — for example, the Mouse— you would use `input.mouse.enabled = false`.

***NOTE:*** How often should the input pointers be checked for updates? A value of 0 means every single frame (\60 fps or 16.667 ms); a value of 1 means every other frame (\30 fps or 33.33 ms) and so on. These are approximations based on your game "workload". More on this below in ***Advanced Concepts: Animations***.

## Adding Buttons & Mobile Touch

Phaser v2.x.x has built-in functions for handling buttons; however, that is ***NOT*** the case with Phaser v3.x.x (caveat: as of 20180729). The fact is that buttons — as clickable game objects — are just as simple to make in Phaser v3.x.x. The "button" behavior in Phaser III are separate functions (such as hovering state, animations, and other special visual effects (sfx)) are just "decorations".

Everything created inside a "Scene" comes from the "Phaser > GameObjects > GameObject". So any text we add to our v3 buttons comes from ***"Phaser.GameObjects.Text"*** and the button itself from ***"Phaser.GameObjects.GameObject".*** To prepare an object to become a clickable button, we turn on one simple game object property.

```
//Phaser v3.x.x enabling clickable input and emit input events directly.

<objectName>.setInteractive();
<objectName>.on(EVENT, CALLBACK, this)

//listens for a pointer down event anywhere on the game canvas
this.input.on('<objectName>', this.onObjectClicked, this)

//assigning a callback function
onObjectClicked('pointer', '<objectName>')
```

We could set this on our button text label or we could set our button's body. If we choose a game object without any geometry background, such as our text label, Phaser v3.x.x will create a default rectangle for us. ***".setInteractive"*** cause the game object to send ***"events",*** but we need something to ***"listen"*** for those ***"events".*** The problem is understanding where are those ***"events"*** are coming from? How can we identify which button was pressed? We also want to filter any ***"events"*** coming from the ***"Scene"*** in general too.

Any ***"gameObject"*** with its ***".setInteractive()"*** enabled sends an ***Event Emitter 3 (EE3)***[12] which are faster than the former ***"EventEmitter2"*** events. We can listen and capture EE3 using the Phaser v3.x.x ***"on"*** method.

```
//Phaser 3.x.x button
var button = this.add.text(100,100, "I'm a button",);
button.setInteractive();
```

[12] https://github.com/primus/eventemitter3

```
button.on('pointerover',function (pointer) {

this.setTint(0x00FF00);
console.log('Over the button now');

});)
```

Using the code above we know our button is sending events. Other events available in Phaser v3.x.x are:

- ***"pointerout"*** — this is the opposite of ***"pointerover".*** It fires when the cursor leaves the game object geometry area.
- ***"pointerdown"*** — this triggered when a click or a touch occurs on the game object — when the mouse button is pushed down or when a finger touches down.
- ***"pointerup"*** — this is the opposite of ***"pointerdown".*** This fires when the mouse button is released or a finger is lifted off the game object.

Read more details about how to extend Phaser III with decorative ES6 frosting ***from this article***[13].

I am using "multi-state" buttons — a concept I learned while creating games using Adobe Flash. When a gamer "rolls over", "clicks" and "rolls off" my menu buttons, that button changes colors (i.e., provides a visual clue). I use the standard (international) colors of red (stop), green (go) to visually clue my gamers. Mobile devices are different to simulate the same button behavior since it is impossible to "roll over" a navigation button on a mobile device.

Buttons are deployed as sprite sheet animations. I have one frame for each state (visual clue) my button will display — over, clicked, off. To begin, I create my button animation in Flash. Notice the highlighting; it moves to give the illusions of animation. I used Texture Packer[14] to build the JSON atlas for my buttons

**Button spriteSheet Frames**

[13] https://snowbillr.github.io/blog/2018-07-03-buttons-in-phaser-3/
[14] https://www.codeandweb.com/texturepacker

***Hint:*** I found a tool called ***"Flash to Phaser"***[15]. It is simply marvelous! I salvaged all my flash animations from 265 games and converted them into HTML5 Phaser arcade games. This allowed me to keep my substantial investment in a ***"dead software package"***[16] and "re-tool" my current skill in ActionScript toward ES5/6/7/8/9/10.

## Phaser III *"Actions"*

Actions are a new set of functions in Phaser III; they perform tasks on groups of internal game objects. In Phaser v2.x.x, groups were used as game object pools; it permitted you to manipulate their content. In contrast, Phaser III offers the same convenience but is not limited to groups only! "Actions" provide access from anywhere, as long as you provide ***an array of game objects.*** For example, observe their impact on a "game object layer".

**Example from http://phaser.io/phaser3/api/actions**

```
    angle: function (value) {

    Actions.Angle(this.children.entries, value);

    return this;
    },

    /**
    Depending on the Action it can also be used dynamically,
    such as in an update function. Here we use the RotateAround
    Action to rotate all the children of the Layer around the given point:
    */
    function create (){

        layer = this.add.layer();

        for (var i = 0; i < 256; i++) {
            var image = this.add.image(Phaser.Math.Between(200, 600),
                Phaser.Math.Between(100, 500),
                'diamonds', Phaser.Math.Between(0, 4));

            layer.add(image);
```

[15] http://www.photonstorm.com/phaser/flash-to-phaser-jsfl-script
[16] https://www.online-tech-tips.com/computer-tips/flash-player-in-chrome-is-dead-in-2020-how-to-play-flash-files/

```
23          }
24      }
25
26      function update () {
27          layer.rotateAround({ x: 400, y: 300 }, 0.01);
28      }
```

## Components

Phaser 2 used components to avoid duplicating Game Object code; components were applied as "mixins" onto a Game Objects. ***In Phaser III, components*** perform the same tasks ***with additional capabilities*** of "getters" and "setters". ***Components are not restricted to gaming objects only,*** they can "attach" from anywhere. Typically, a game object describes which components are available; now they used as a mixin array. For example:

### What's the Difference Between Class & Prototypal Inheritance?

... you have to understand that there are three different kinds of prototypal OO.

Concatenative inheritance: The process of inheriting features directly from one object to another by copying the source objects properties. In JavaScript, source prototypes are commonly referred to as mixins. Since ES6, this feature has a convenience utility in JavaScript called `Object.assign()`. Prior to ES6, this was commonly done with Underscore/Lodash's `.extend()` jQuery's `$.extend()`, and so on... The composition example above uses concatenative inheritance.

Prototype delegation: In JavaScript, an object may have a link to a prototype for delegation. If a property is not found on the object, the lookup is delegated to the delegate prototype, which may have a link to its own delegate prototype, and so on up the chain until you arrive at `Object.prototype`, which is the root delegate. This is the prototype that gets hooked up when you attach to a `Constructor.prototype` and instantiate with `new`. You can also use `Object.create()` for this purpose, and even mix this technique with concatenation in order to flatten multiple prototypes to a single delegate, or extend the object instance after creation.

Functional inheritance: In JavaScript, any function can create an object. When that function is not a constructor (or `class`), it's called a factory function. Functional inheritance works by producing an object from a factory, and extending the produced object by assigning properties to it directly (using concatenative inheritance). Douglas Crockford coined the term, but functional inheritance has been in common use in JavaScript for a long time.

> As you're probably starting to realize, concatenative inheritance is the secret sauce that enables object composition in JavaScript, which makes both prototype delegation and functional inheritance a lot more interesting.
>
> When most people think of prototypal OO in JavaScript, they think of prototype delegation. By now you should see that they're missing out on a lot. Delegate prototypes aren't the great alternative to class inheritance—object composition is.

**Example from http://phaser.io/phaser3/api/components**

```
    var Image = new Class({
        Extends: GameObject,

        Mixins: [
            Components.Alpha,
            Components.BlendMode,
            Components.Flip,
            Components.GetBounds,
            Components.Origin,
            Components.RenderTarget,
            Components.ScaleMode,
            Components.Size,
            Components.Texture,
            Components.Transform,
            Components.Visible,
            ImageRender
        ],

        initialize:

        function Image (state, x, y, texture, frame) {
            GameObject.call(this, state, 'Image');

            this.setTexture(texture, frame);
            this.setPosition(x, y);
            this.setSizeToFrame();
            this.setOrigin();
        }

    });
```

### DOM

Phaser v3 contains access to all document object model within a single namespace. This provides v3 direct access into various DOM functions such as:

- AddEventListener
- AddToDOM
- CanvasInterpolation
- CanvasPool
- DOMContentLoaded
- ParseXML
- RemoveEventListener
- RemoveFromDOM
- RequestAnimationFrame
- TouchAction
- UserSelect

This is important since it provides the use of orphans `div` that can bind to the DOM upon request.

## Game Objects

How Game Objects Work (20170601)[17]

## System Components

Some system-wide objects have their own components folder. Phaser v3 is structured in this way to avoid having too many lines of code in any single module. In turn, this provides a faster lookup.

[17] https://phaser.io/phaser3/contributing/part6

**Example of System-wide Component**

```
    Tween.prototype = {

        calcDuration: require('./components/CalcDuration'),
        init: require('./components/Init'),
        loadValues: require('./components/LoadValues'),
        nextState: require('./components/NextState'),
        pause: require('./components/Pause'),
        play: require('./components/Play'),
        resetTweenData: require('./components/ResetTweenData'),
        resume: require('./components/Resume'),
        seek: require('./components/Seek'),
        setEventCallback: require('./components/SetEventCallback'),
        update: require('./components/Update')

    };
```

## 6.2 Tile Map

During the development of Phaser v3, they discovered four different approaches toward creating Tile-maps of which two proved to be equally powerful in different gaming scenarios. So the team decided to provide us a choice of two different — and most effective — Tile Mapping methods from all those studied. The difference between the two chosen Tilemap renderer can be found in how tiles are prepared for the GPU. The new Static Tilemap stores the map data in VRAM, thereby, avoiding data surrendered to the GPU every frame. Whereas the new Dynamic Tilemap is built especially for the ***"SpriteBach"*** renderer; it pushes data into the GPU every frame. If your data is constantly or even slightly changing, you will need this new dynamic approach.

### Tilemap Rendering - new Dynamic method

This approach ***(aka Dynamic Tile-map Renderer)*** follows the typical canvas rendering. It pushes vertices into the GPU (graphics processing unit) in the same manner as ***"BlitterBatch" or "SpriteBach".*** It works by filling the vertex data array with vertices every frame. Yes, this is slow and will produce lag as the CPU/GPU upload from the element buffers. The dynamic tilemap generates an array of metadata tiles that describe the tile that will be rendered. The tile description contains such properties

as alpha, tint, and visibility. This tile is also culled — based on the current camera — so we don't render unnecessary tiles. Finally, the tiles are rendered by adding them to the ***"`SpriteBach`".*** This generates all the vertices during runtime and loads them into the GPU.

"Why do — or retain — this?", you say. It is ***flexible; you could modify any tile-map in "real-time" — an important consideration when the map must change constantly as in an "endless runner" game.*** The recommendation is to keep the view-port small since you won't be able to quickly fill large areas.

You would use this Dynamic method under the following circumstances:

- Animated Tiles can be animated by way of modifying their tile ID.
- Per-Tile Tint support (for WebGL only on a single tile)
- Per-Tile Alpha and Visibility support (on a single tile)
- Real-time modifications

## Tilemap Rendering - new Static method

This approach ***(aka Static Tile-map Rendering)*** is similar to the method above with this exception. Instead of filling and updating the vertex buffer each frame, this method only performs an initial load — similar to rendering static meshes stored in VRAM. Doing so, avoids synchronization between the CPU and GPU by simply submitting graphics library drawing commands with "N to N+M" vertices, and since tiles are in sequential order, thus facilitating "back-face culling"[18]. However, there are drawbacks to this method; it does not permit "Dynamic rendering" as experienced in the method above. However, what you lose in rapid updates; you gain in extremely fast displays. Here are the test results for your consideration:

> ### Quote from Devlog 82:
>
> We used what we would consider an 'extreme' test map of 150 x 10,000 tiles (1.5 million tiles in total) and is rendered ***in a single draw call at a solid 60fps on dedicated GPUs.*** On my rubbish Intel HD GPU, it managed it at 45fps. Not many games will need 1.5 million tiles, however, so if you bring this approach back down into the realm of a normal title, it should cope even with integrated graphics.

***IMPRESSIVE, Yes, indeed!***

[18] https://en.wikipedia.org/wiki/Back-face_culling

## 6.3 Phaser III Systems

### v3 Boot

The new ***"boot"*** is comprised of several methods both old and new.

- ***"Game Config":*** This accepts a configuration object into the Phaser.Game constructor.
- ***"Create Renderer":*** This method work with the ***"Device Manager"*** and the ***"Game Config"*** to correctly select the proper renderer — currently Canvas or WebGL.
- ***"Debug header":*** This method logs notifications to the console.log. It is significantly different from v2.x.x. You could use this to promote your game or hide it completely from the curious.
- ***"Game":*** is the standard point of entry into ***"Phaser.Game".*** It is now responsible for creating all global systems.
- ***"TimeStep":*** manages the ***"RequestAnimationFrame"*** ***or*** ***"SetTimeout".*** It also calculates the delta time values, handles visibility loss, delta resets, and calls the ***"Game step".***
- ***"Visibility Handler":*** listens for DOM page visibility, blur, and focus events. It sends those updates to the current Game Scene (formerly known as 'state').

**Example from http://phaser.io/phaser3/api/boot**

```
    var config = {
        width: 800,
        height: 600,
        resolution: 1,
        type: Phaser.WEBGL,
        parent: 'phaser-example',
        scene: {
        preload: preload,
        create: create
        },
        callbacks: {
            preBoot: function () { console.log('I get called before all
                        of the Game systems are created,
                        but afterthe Device is available')},
            postBoot: function () { console.log('I get called after
                        all of the Game systems are running,
                        immediately before raf starts')}
        }
```

```
    };

    var game = new Phaser.Game(config);

    /**
    Note that I could have simply inserted the config object directly.
    */

    var game = new Phaser.Game(
        //internalized configuration object
        {
            width: 800,
            height: 600,
            resolution: 1,
            type: Phaser.WEBGL,
            parent: 'phaser-example',
            scene: {
            preload: preload,
            create: create
        },
        callbacks: {
            preBoot: function () { console.log('I get called before all
            of the Game systems are created, but after Device is
            available')},
            postBoot: function () { console.log('I get called after all
            of the Game systems are running, immediately before
            raf starts')}
        } //End of config object
    );  //End of game instance
```

There is a limit of 255 arguments per MDN.[19] passed into a JavaScript function.

What attributes are pre-configured in the new v3 configuration object? Here's what we could currently find:

[19] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions

```
    //Phaser3 template modified from brunch Phaser
    // See <https://github.com/photonstorm/phaser/blob/master/src/boot/Config.js>
    // <https://github.com/digitsensitive/phaser3-typescript/blob/master/cheatsheets\
/game-config.md>

    var GAMEAPP = new Phaser.Game(
        //internalized configuration object
    {
    // For all newly added settings refer to
    // <https://github.com/photonstorm/phaser/blob/master/src/boot/Config.js>

    width: 800,       //maintain Golden Ration
    height: 500,      //maintain Golden Ration
    // zoom: 1,
    // resolution: 1,
    type: Phaser.AUTO,
    // parent: null,
    // canvas: null,
    // canvasStyle: null,
    // scene: {},     //new 20180401
    // seed: null,
    title: 'Phaser3 Game Prototyping Starter Kit',  //Game Title
    url: 'https://makingbrowsergames.com/',   //Game URL location
    version: 'semver 0.0.1.0',        //semver.org v1.0.0.html
    input: {
        keyboard: true,
        keyboard.target: window,//new 20180401
        mouse: true,
        mouse.target: null,     //new 20180401
        touch: true,
        touch.target: null,     //new 20180401
        touch.capture: true,    //new 20180401
        gamepad: false
    },
    // disableContextMenu: false,
    // banner: false
    banner: {
        hidePhaser: false,
        text: 'white',
        background: ['#e54661', '#ffa644', '#998a2f', '#2c594f', '#002d40']
    },
        //Frame Rate config
        //fps: {
```

```
44          //  min: 10,
45          //  target: 60,
46          //  forceSetTimeout: false,
47          //  deltaHistory: 10,
48          //  panicMax: 120        //new 20180401
49          //},
50
51          // pixelArt: false,
52          // autoResize: false,        //new 20180401
53          // roundPixels: false,        //new 20180401
54          // transparent: false,
55          // clearBeforeRender: true,
56          // backgroundColor: 0x000000,        // black
57
58          //  Callbacks
59          // callbacks:   {
60              //preBoot: NOOP,
61              //postBoot: NOOP,
62          //},
63
64          //Physics
65          //  physics: {
66          //      system: 'impact', // removed v3.23! 20200427
67          //      setBounds: true,
68          //      gravity: {},
69          //      cellSize: 64,
70          //      debug: false    //new 20180401
71          //  },
72          //default: false,       //new 20180401
73          //  Loader Defaults
74          loader: {
75                  //baseURL: '',     //site lock for game assets
76                  path: 'assets/',
77                  enableParallel: true,  //new 20180401
78                  maxParallelDownloads: 10, //varies by browser from 2 to 60
79                  crossOrigin: 'anonymous', //required for affiliate usage
80                  //responseType: ?,   //new 20180401
81                  asyn: true,
82                  //user: ?          //new 20180401
83                  //password: '',
84                  //timeout: 0
85          },
86          //images: {}            //new 20180401
```

```
87          //images.default: ?     //new 20180401
88          //images.missing: ?     //new 20180401
89
90        scene: [
91         //require('scenes/boot'),
92          //require('scenes/default'),
93          //require('scenes/menu')
94        ],
95
96      });
97
98
99      GAMEAPP.scene.add('boot', window.GAMEAPP.boot);
100     this.scene.add('default', window.GAMEAPP.default);
101     this.scene.add('menu', window.GAMEAPP.menu);
102
103     this.scene.start('boot');
```

### v3 Cache

As soon as the game boots, a global game-wide cache is created. This cache is the gatekeeper to the various subordinate caches created for each game asset and resource. For example, you could access any text by simply using `cache.text`. Here's an example of game resource caches created after booting.

- this.binary = new BaseCache();
- this.bitmapFont = new BaseCache();
- this.json = new BaseCache();
- this.physics = new BaseCache();
- this.shader = new BaseCache();
- this.sound = new BaseCache();
- this.text = new BaseCache();
- this.tilemap = new BaseCache();
- this.video = new BaseCache();
- this.xml = new BaseCache();

You can manage cache contents using common access methods of `.add`, `.has`, `.get`, or even `.remove`; you will use string-based keys for these methods.

**Examples from phaser.io/phaser/api/loader**

```
    function preload() {

        this.load.json('jsonData', 'assets/atlas/megaset-0.json');

    }

    function create() {

        console.log(this.cache.json.get('jsonData'));

    }
```

## v3 Device Manager

Inspecting the users' browser and its capabilities are now consolidated in this manager. It determines the base operating system, browser currently used to access the game, and various input support — such as audio, video, inputs, screen dimensions, and canvas features.

**Examples from phaser.io/phaser/api/loader**

```
    if (this.game.device.features.pointerLock)
    {
        // It does
    }

    if (this.game.device.os.iOS && this.game.device.os.iOSVersion > 9)
    {
        // Device is iOS9 or above
    }
```

## v3 Events

Events use new customized independent dispatchers throughout the game systems; you can also create your own events listeners and bindings. Access the new v3 is easy; simply use `this.events`. Events could be filtered in the case of stacked objects; events can have priorities or even stop event propagation.

**Examples from phaser.io/phaser/api/loader**

```
    //Dispatching a custom event via the States Event Dispatcher:
    //  Here is an Event ...
    var playerEvent = new Phaser.Event('shoot');

    //  We'll use the States own EventDispatcher to listen for,
    //        and dispatch the events

    //  And here is the listener
    this.events.on('shoot', handler);

    //  Dispatch the event
    this.events.dispatch(playerEvent);

    function handler(event) {

        console.log('Event Received by Handler:', event);

    }

    //Events can also have priorities and have their propagation stopped:

    //  Here is an Event ...
    var playerEvent = new Phaser.Event('shoot');

    //  And here are 2 listeners.
    //  The second listener has a higher priority than the first,
    //        so will be called first.

    //  We'll use the States own EventDispatch to listen for, and dispatch the event\
s
    this.events.on('shoot', handler1, 5);
    this.events.on('shoot', handler2, 10);

    //  Dispatch the event
    this.events.dispatch(playerEvent);

    function handler1(event) {

        console.log('Event Received by Handler One:', event);

    }
```

```
42
43      function handler2(event) {
44
45          console.log('Event Received by Handler Two:', event);
46
47          //  This stops the event getting any further, so handler1 will never fire
48
49          event.stopPropagation();
50
51      }
```

## v3 Input Manager

The input manager takes on the new role as a global input gatekeeper. It is now responsible for monitoring and processing all user inputs. The ***"config"*** provides the settings for the Input Manager to create various handlers. The Keyboard and Gamepad are directly handled as plugins by an InputPlugin which is a Scene system responsible for all input events with the parent Scene. The primary responsibilities of the Input Manager are:

- manage the input event queue
- create various pointers, and
- manage various hit tests and related operations.

> "Previously the Input Manager would create a Touch handler unless the Game Config had ***"input.touch"*** set to ***"false"*** (the default was true). If no such property is set, it no longer defaults to `true` and instead is set to whatever ***"Device.input.touch"*** returns. On "non-touchscreen displays" this means it will now only create a single Pointer, rather than two."

Phaser III handles movement differently than v2. In v3, ***"move events"*** are an updated feature. Quoted from DevLogs 90,

> The Input Manager consists of two parts: The Global Input Manager, which is owned by the Game itself, and the Input Manager, which is a Scene level system.
>
> The Global Input manager is responsible for monitoring and processing user input, regardless of input method. It starts and handles the DOM event listeners for the keyboard, mouse and touch inputs. It then queues these events which are processed every game step.

***Key handling has changed significantly in 3.16,***[a] Mouse Handler and Touch Handler. Gamepad and Pointer Lock will be added shortly.

These events are dispatched whenever a pointer is in the processing of moving across an interactive object. It doesn't have to be pressed down or dragging, it just has to be moving. As part of the event you are sent the local coordinates of the pointer within the sprite. So you could use it for a 'sliding' UI element that you control by just sliding a finger up and down it, such as a volume meter.

### Callbacks and Events

In v2 nearly all input was handled via Signals. You'd listen to a signal bound to a specific sprite to know if the pointer was pressed down on it.

In v3 you can use both callbacks and events. The events belong to the Input Manager itself, not the game objects. So, you could listen for a Pointer Down event from the Input Manager. As part of the event properties you are given a list of all the Game Objects that the pointer went down on, as well as the top-most one in the display list.

The callbacks, however, belong to the Game Objects. You can set a callback for every type of input event: over, down, up, out, move and the drag events: start, drag and end. Callbacks are invoked on a per-Game Object basis and are sent a bunch of useful arguments as well. Depending on the type of game you're building you may favour one approach over the other, or maybe just out of personal preference too. By having both options available though it gives you the flexibility to decide, rather than enforcing it upon you.

```
//Phaser v3 method is extremely easy to activate
var mySprite = this.add.sprite(400, 300, 'texture').setInteractive();
mySprite.setOrigin(0,0); //set sprite anchor to upper left corner
```

[a] https://github.com/photonstorm/phaser/blob/master/CHANGELOG.md#keyboard-input---new-features

## Deeper Dive: v3.16+ New Keyboard rewrite!

### Keyboard Input - New Features

***Quoted from v3.16+ Change Log***[a]

The specificity of the Keyboard events has been changed to allow you more control over event handling. Previously, the Keyboard Plugin would emit the global key-down_CODE event first (where CODE was a keycode string, like keydown_A), then it would emit the global keydown event. In previous versions, Key objects, created via

***"this.input.keyboard.addKey()",*** didn't emit events.

The Key class now extends EventEmitter and emits two new events directly: down and up. This means you can listen for an event from a Key you've created, i.e.: yourKey.on('up', handler).

The order has also now changed. If it exists, the Key object will dispatch its down event first. Then the Keyboard Plugin will dispatch keydown_CODE and finally the least specific of them all, keydown will be dispatched.

You also now have the ability to cancel this at any stage either on a local or global level. All events handlers are sent an event object which you can call ***"event.stopImmediatePropagation()"*** on. This will immediately stop any further listeners from being invoked in the current Scene. Therefore, if you call ***"stopImmediatePropagation()"*** in the Key.on handler, then the Keyboard Plugin will not emit either the ***"keydown_CODE"*** or keydown global events. You can also call ***"stopImmediatePropagation()"*** during the keydown_CODE handler, to stop it reaching the global keydown handler. As keydown is last, calling it there has no effect.

There is also the "stopPropagation()" function. This works in the same way as ***"stop Immediate Propagation"*** but instead of being local, it works across all of the Scenes in your game. For example, if you had 3 active Scenes (A, B and C, with A at the top of the Scene list), all listening for the same key, calling ***"stopPropagation()"*** in Scene A would stop the event from reaching any handlers in Scenes B or C. Remember that events flow down the Scene list from top to bottom. So, the top-most rendering Scene in the Scene list has priority over any Scene below it.

All the above also works for keyup events.

New in 3.16 is the ability to receive a global keydown or keyup event from any key on the keyboard. Previously, it would only emit the event if it came from one of the keys listed in the KeyCodes file. Now, those global events will fire for any key, regardless of location.

[a] https://github.com/photonstorm/phaser/blob/master/CHANGELOG.md#keyboard-input---new-features

Add a mouse with `game.input.mousePointer` (always refers to the mouse if present). This is the safest method if you only need to monitor the mouse.

Phaser's `mainMenu update()` function checks for input events; remember, `update()` attempts to run 60 times a second. The `mainMenu update()` function is our game loop, which continues to run until we exit the game. So any animation, state or display changes or game events will be in here.

Let's turn our attention to the speed and velocity of our avatar. We should set a fixed movement speed; you might want to ***"tinker"***[20] with this number until it "feels" correct and proper. We should also set our "velocity" parameter to zero; because doing so, will prevent the avatar's movement until an arrow key is pressed. Place the following snippet in the `mainMenu update()` function.

## v3 Loader

As in Phaser v2, the loader is still responsible for loading external game assets and resources. But in contrast to the global role in v2, v3 now has a separate loader per each game Scene (formerly called "state" in v2). Each game scene in v3 is responsible for loading its own resources and gaming assets when it starts. This scene loading runs in parallel; meaning that a scene will load its resources even if another scene is currently loading.

The ***"BaseLoader"*** class governs the loading process. It is responsible for queue management, dispatching events, and load management. The ***"BaseLoader"*** class handles the follow ***"filetypes"*** using the ***".addfile"*** method:

- AnimationJSON File
- AtlasJSON File
- Binary File
- BitmapFont File
- GLSL File
- HTML File
- Image File
- JSON File
- SpriteSheet
- SVG File
- Text File
- XML File

Each Scene can further use this.load.image, this.load.json, and this.load.atlas. You can also pass configuration objects to these methods.

[20] http://dictionary.cambridge.org/us/dictionary/english/tinker

**Examples from phaser.io/phaser/api/loader**

```
    //  Original image loader signature:
        this.load.image('bunny', 'assets/sprites/bunny.png');
    //  Object based
    this.load.image({ key: 'bunny', texture: 'assets/sprites/bunny.png' });

    //  Allows for arrays of objects
    this.load.image([
        { key: 'bunny', texture: 'assets/sprites/bunny.png' },
        { key: 'atari', texture: 'assets/sprites/atari400.png' },
        { key: 'logo', texture: 'assets/sprites/phaser2.png' }
    ]);

    //  Object based including XHR Settings
    this.load.image({
        key: 'bunny',
        texture: 'assets/sprites/bunny.png',
        xhr: {
            user: 'root',
            password: 'th3G1bs0n',
            timeout: 30,
            header: 'Content-Type',
            headerValue: 'text/xml'
        }
    });

    //  Auto-filename based on key:

    //  Will load bunny.png from the defined path, because '.png'
    //       is the default extension.
    this.load.image({ key: 'bunny' });

    //  Will load bunny.jpg from the defined path, because of the 'ext' property.
    this.load.image({ key: 'bunny', ext: 'jpg' });

    //  ----------------------
    //  Texture Atlas Examples
    //  ----------------------

    //  Original atlas loader signature:
    //  this.load.atlas(key, textureURL, atlasURL, textureXhrSettings, atlasXhrSetti\
ngs)
```

```
42
43      this.load.atlas('level1', 'assets/level1/items.png',
44              'assets/level1/items.json');
45
46      //  Object based
47      this.load.atlas({ key: 'level1', texture: 'assets/level1/items.png',
48              data: 'assets/level1/items.json' });
```

## v3 Sound

Managing sound and audio is completely different from v2.x.x. v2 used Audio Tags and Web Audio as similar system files. v3 now properly separates the Audio tags from Web Audio; this provides you the option to exclude legacy Audio tags from your game.

I'm excited about the ***dynamic sound generation feature in v3.*** This provides a better solution than v2.x.x and examples are given in ***Phaser Game Design Workbook (6th edition)***[21].

The current plans for Phaser III are to tie sounds into its parent Scene (aka: formerly known as v2.x.x "State"); this provides unique volume, sound effects, and audio contexts for each Scene. Unfortunately, legacy Audio is not part of the current scope, and more planning and research must be performed. All sound effects will be governed by the new Sound Manager.

**Examples from phaser.io/phaser/api/sound-manager**

```
1       var ctx = new AudioContext();
2
3       var explosionEffect = {
4           frequency: 16,
5           decay: 1,
6           type: 'sawtooth',
7           dissonance: 50
8       };
9
10      new Phaser.Sound.Dynamic.FX(ctx, explosionEffect);
11
12      window.addEventListener('mousedown', function () {
13
14          new Phaser.Sound.Dynamic.FX(ctx, explosionEffect);
```

[21] https://leanpub.com/phaserjsgamedesignworkbook

```
15
16      }, false);
```

## v3 Scene Manager

### From v2.x.x States to Scenes

Quoted from http://phaser.io/phaser3/devlog/89 and from

***How "States" work in Phaser III***[a] - R Davey

Since starting Phaser 3 development in earnest I have been carefully evaluating everything that is going into it. I'm not just porting over v2 classes for the sake of it. In fact, the vast majority of the code in v3 is brand new, written entirely from scratch. However, the changes don't end at just the API - this is also the time to carefully reflect on internal choices as well, including the naming of things.

One such thing is the State Manager. Phaser has always used the term 'state' since day 1 because it was inherited from the Flixel project before it. Yet as a term it has confused a number of developers over the years. So I did a little research to see what terminology other frameworks used and the results were quite surprising. The most common term was 'Scene', used to represent a collection of Game Objects (or nodes in some frameworks). Lots of the more visual game engines use the term 'Level' instead, and others like Game Maker use the term 'Room', but none used State.

As a result, I have changed the use of the term 'State' within v3 to 'Scene'. There is now a ***"SceneManager",*** all Game Objects have a property called 'scene' which indicates the scene responsible for them, and internally 'scene' is now used everywhere as well. It actually changes nothing with regard to features, but it does require a change in mindset. I've been typing in 'state' for so many years now that I'm still getting used to 'scene' instead :) But it's a more logical name and now was the time to change it. Game Objects will remain being called that however (just like in Unity) as I find the term 'node' too generic.

[a] https://phaser.io/phaser3/contributing/part5

Phaser III new "State" Manager maintains and runs multiple scenes in parallel. As mentioned earlier, former "States" are now complete scenes worlds in their own right (the "world" property is no longer used). In Phaser v2.x.x there was a concept of the "Game World", in which all Game Objects lived. This concept was removed in Phaser III and replaced with the "Scene", which maintains their own "worlds". Scenes are created in several ways:

- From Scene file payloads;
- From Classes;
- From Functions;
- From Instances; or
- From Objects.

Scenes (formerly known as v2.x.x "States") are managed by a global Manager; it parses, creates, and maintains all the game's scenes! The global "Scene Manager" is created during the ***"`Phaser.Game` boot phase".*** The global "Scene Manager" has four important properties it monitors:

- the game: a single reference to Phaser.Game.
- the settings: defined by the game's developer for each specific Scene — such as fps, width, height, scale, etc.
- the system ("sys"): as the game State System property.
- the children: all display level objects in this scene.

When a Scene needs to reference another Scene, it must do so through the ***"`scene.sys`"*** property. For example, to add an object to the display list, you should use ***"`scene.sys.add`"*** and not the former v2.x.x method ***"`state.add`".***

Files payloads can be referenced in the Scene config now, and the files will be loaded before the scene — meaning they're available even before the preload function (if set) is called. This provides the opportunity for loading in small JSON config files or small sets of assets required by preloader itself to use:

**Example from http://phaser.io/phaser3/api/scene-manager**

```
    /**
    Here all we do is defined two functions, preload and create:
    */

    var stateConfig = {
        preload: preload,
        create: create,
        files: [
            { type: 'image', key: 'sonic', url:
                    'assets/sprites/sonic_havok_sanity.png' }
        ]
    };
```

```
13
14
15      var config = {
16          type: Phaser.CANVAS,
17          parent: 'phaser-example',
18          width: 800,
19          height: 600,
20          scene: stateConfig
21      };
22
23      var game = new Phaser.Game(config);
24
25      //End of this example.
26
27      /**
28      In this example we're creating 2 States using State Configuration Objects,
29              which are passed to the Game constructor.
30      */
31
32      var backgroundStateConfig = {
33          key: 'background',
34          active: true,
35          create: createBackground,
36          render: renderBackground,
37          files: [
38              { type: 'image', key: 'face', url: 'assets/pics/bw-face.png' }
39          ]
40      };
41
42      var modalStateConfig = {
43              key: 'modal',
44              active: true,
45              renderToTexture: true,
46              x: 64,
47              y: 64,
48              width: 320,
49              height: 200,
50              create: createModal,
51              render: renderModal,
52              files: [
53               { type: 'image', key: 'logo',
54                      url: 'assets/pics/agent-t-buggin-acf-logo.png' }
55              ]
```

```
56      };
57
58      var gameConfig = {
59          type: Phaser.CANVAS,
60          parent: 'phaser-example',
61          width: 800,
62          height: 600,
63          scene: [ backgroundStateConfig, modalStateConfig ]
64      };
65
66      var game = new Phaser.Game(gameConfig);
```

## v3 Texture Manager

The ***"Texture Manager"*** manages all game textures as a singleton class; this means there should only be one active and bound to the ***"Phaser.Game"***. The ***"Loader"*** passes game assets and resources to the ***"Texture Manager"*** who will store them in their appropriate local cache. The ***"Texture Manager"*** has several parsers to support the following formats:

- Canvas
- Image
- Texture Atlas data in JSON arrays or JSON hash formats.
- Pyxel files
- Starling XML Atlas files and
- Sprite Sheets.

Game Objects have immediate access to their textures from the ***"Texture Manager"***. The ***"Texture Manager"*** furthermore has several utility functions using the internal Phaser Canvas Pool:

```
var canvas = this.textures.createCanvas('fill', 256, 256);

this.textures.get('mario_mask');
```

## v3 Tween Manager

Each Phaser v3 Scene owns a ***"Tween Manager"*** whose task is managing all tweens within the Scene. Making the ***"Tween Manager"*** subordinate to each Scene State allows you to create different moods or simply pause all tweens of a specified Scene. This is a dramatic change from v2.

Tweens are created from a configuration object passed into the ***"Tween Manager.add"***. These configurations are sent to the ***"TweenBuilder"*** who is responsible for the execution. The "builder" returns a single object for the manager to add to the local pool and manages its required updates. Tweens can support updating at different rates, different ease settings, and durations — even on the same target object. Tweens can perform "yo-yo" movement, play reverse sequence, and even seek a specific animation point.

**Example from http://phaser.io/phaser3/api/tween-manager**

```
    //A simple Tween that updates an Images x coordinate:
    var image = this.add.image(100, 100, 'block');

    var tween = this.tweens.add({
        targets: image,
        x: 600,
        ease: 'Power1',
        duration: 3000
    });

    /**
    This Tween updates two properties. Note how they have their own custom durations\
 and eases:
    */

    var image = this.add.image(100, 100, 'block');

    this.tweens.add({
        targets: image,
        x: { value: 700, duration: 4000, ease: 'Power2' },
        y: { value: 400, duration: 1500, ease: 'Bounce.easeOut' }
    });

    //An example of passing custom parameters to the ease function:

    this.tweens.add({
```

```
27          targets: image,
28          x: 600,
29          duration: 3000,
30          ease: 'Elastic',
31          easeParams: [ 1.5, 0.5 ],
32          delay: 1000
33      });
```

### Deeper Dive 3.19+ Tweens

The Tween system in release 3.19 ***is a huge overhaul and has extended the system capabilities significantly;*** I would advise a review of ***any released games using the old Phaser III tween system (pre-v3.19+) before migrating to this newest Phaser III. Tweens are fully documented.***[22]

Some new Tween Events are ***"COMPLETE"*** or ***"REPEAT";*** these allow triggered actions without creating callbacks. Another example from v3.19+ is that tweens can implement both ***'to'*** and ***'from'*** values. This is a handy addition whenever you'd like to start from a specific frame in any tweened asset property. `Tween.hasStarted` alerts you concerning a running tween. There's even a new Tween seeking function that provides a search to any point in time across a tween.

Other useful tools newly added in Phaser III.19 are:

- ***'StaggerBuilder'*** — This provides "staggered offsets" to a collection of tweening targets. You might use this while staggering targets across grid layouts and in preferred directions by merely setting a starting value.
- ***Shader.setRenderToTexture*** — provides a redirection of a shader to its own framebuffer or WebGL texture instead of using display lists. You might even consider piping one "output" shader as the input to another shader!
- ***RenderTexture.snapshot*** — is the answer to a popularly requested feature. This new feature allows a "snap-shot" on any rendered texture in a point in time and then convert that snap-shot to an image asset for the Texture Manager or as a newly saved image in the file system. I've been waiting for this feature for years!

## 6.4 Phaser3 Finish Line: You're AWESOME … Gloat!, Gloat!

### Phaser v3 Source Code & Demos

[22] https://photonstorm.github.io/phaser3-docs/Phaser.Tweens.html

### Quote from Devlog 85

I appreciate it's quite a simple game but I feel like it has shown off a number of Phaser 3 features clearly: Layer Actions, which you can easily create your own to extend in all kinds of directions. An easy to use Phaser Class construct and even Dynamic audio.

Because the Snake is a fully self-contained class there is nothing stopping you from taking the code and using it elsewhere or even making a version where you have to control multiple Snakes at once. You could add in objects to avoid, power-ups, flies or different kinds of food. There are a lot of ways it could be expanded.

I hope you enjoyed this little trip into the works of using Phaser 3 for an actual game. Next issue we'll resume with a normal Dev Log again, but if little tutorials like this prove to be popular then I'll gladly write more in the future.

Source Code and tutorial http://phaser.io/phaser3/devlog/85

## 6.5 v3 Animations

***NOTE:*** This is feature is not included in my Mozart's Music Match game; but, I foresee I might use it for the follow-up expansions as ***illustrated by this clever developer***[23] or ***this outstanding example***[24] and ***this one***[25].

```
1     function create() {
2         var mummy = game.add.sprite(300, 200, 'mummy');
3
4         //Here we add a new animation called 'walk'
5         //Because no other parameters were given, it
6         //  makes an animation from available frames .
7         //  in the 'mummy' sprite sheet
8         var walk = mummy.animations.add('walk');
9
10        //And this starts the animation by using
11        //  its keyword ("walk"),
```

[23] http://mark-rolich.github.io/MemoryGame.js/
[24] http://jppresents.net/games/memory/
[25] http://igorminar.github.io/Memory-Game/app/index.html

```
12          //  at 30 frames rate per second (30fps)
13          //  true == it will loop when it ends
14          mummy.animations.play('walk', 30, true);
15      }
```

One of my technology students had bought a new ultra-computer. He was so proud of it; "... because I can get 600 frames per second!", he boasted. I, of course, was stunned.

In the US, all electronic devices derive their power from the electrical grid and transformers. Most devices operate at 60Hz; outside the US, it is common to see electronics operate at 50Hz. The refresh rate of a digital display operates at the same frequency because of the power consumed. This is ***"HOW"** digital electronics work; if fact, the monitor is ***"THE SLOWEST"*** networked device; it is restricted by current electronics. It is the ***last device on a local area network (LAN)*** that the user sees. A game can only send so much data to its display device before the next (upcoming refresh screen) must be delivered. ***"WHAT"*** is displayed is a software concern, and software engineers have concocted some clever schemes to overcome the 60Hz (or 50Hz) restriction(s). The reason why I mention this is because, while the frame rate may be higher than the monitor's refresh rate for displaying images, ***the three variables are not dependent on each other. In other words, the number of times your update loop is being called per second is completely independent from the number of times you're refreshing the pixels on the screen, which is also independent from the refresh rate of the monitor itself.*** The only exception to this is the frame rate that you're using for images on the display screen can never be faster than the update loop is called per second (unless you keep them separate, which in my experience is never a good idea, at least with web games).

The reason why GPU developers and game developers are so obsessed with the monitor's frame rate is that it mandates two extremely important ideas. The first one is the ***processing budget*** that you need to work within, and the second is the ***accuracy and precision of certain calculations.*** Those clever schemes are the topics that follow.

The ***processing budget*** mandates how much time is available until the next frame is sent. Consider this, if your game is running at 60 fps, you only have 16.67 ms to handle the all following tasks:

* Pixels being painted on the screen
* Calling the next tick on the physics engine
* Updating variables, game state, and other objects

If for some reason your monitor is unable to process all those tasks in that amount of time, you will skip a frame, which results in jerky and animation stuttering. For this reason, some game developers choose to display graphics at 30 fps. So, instead of

having just 16.67 ms, they have 33.3 ms to process everything, leaving some room to handle more demanding scenes or calculations. Others might choose to call the update loop at 120 fps to have increased precision in physics calculations on the local device, but only display graphics at a constant 30 fps.

## Deeper Dive: History of Animation

Let's take a brief history lesson from Walt Disney –another one of my heroes. Film (aka Movies) technology began in the 1890s; the concept was to simply "flash" multiple pictures — each of which was slightly different. The viewer was fooled into "thinking" they saw a "moving picture". Walt Disney discovered that flipping 24 picture frames per second were the optimal rate; he was awarded an Oscar for Snow White and the Seven Dwarves. The cartoon animation Snow White was 119,550 frames (pictures) in length. Divided by 24 frames per second, and 60 seconds per minute, that comes to around 83 minutes. Research any movie website and discover that 83 minutes is the duration of the Snow White movie. Another little-known fact is how his animators drew this film.

> ***Snow White and the Seven Dwarfs — in Blu-ray***[a]
> The famous Disney "everything in the frame is moving at the same time" isn't there. While the central focus of the frame has movement ***(2 frames per move, i.e. 12 different frames per second)*** the backgrounds and those elements at the sides, stay frozen for all time. The new multi-plane camera is used to beautiful effect.
>
> [a] http://www.hometheaterforum.com/community/threads/a-few-words-about-%E2%84%A2-snow-white-and-the-seven-dwarfs-in-blu-ray.287313/

Disney set the industry standards for "frame rate" — how many pictures flipped per second. The formal definition of frame rate, (also known as frame frequency), is the frequency (rate) at which an imaging device is capable of displaying consecutive pictures called frames. This definition applies equally to film technology, video cameras, computer graphics, and motion capture systems.

> Walt Disney, who had previously been in the short cartoon business, stepped into feature films with the first English-speaking animated feature Snow White and the Seven Dwarfs; released by RKO Pictures in 1937. 1939, a major year for American cinema, brought such films as The Wizard of Oz and Gone With The Wind.
>
> 1982 also saw the release of Disney's Tron which was one of the first films from a major studio to use computer graphics extensively.

> During 1995, the first feature-length computer-animated feature, Toy Story, was produced by Pixar Animation Studios and released by Disney. After the success of Toy Story, computer animation would grow to become the dominant technique for feature-length animation, which would allow competing for film companies such as DreamWorks Animation and 20th Century Fox to effectively compete with Disney with successful films of their own. During the late 1990s, another cinematic transition began, from physical film stock to digital cinema technology. Meanwhile, DVDs became the new standard for consumer video, replacing VHS tapes. ***History of film***[a]
>
> [a] https://en.wikipedia.org/wiki/History_of_film

## Animation Today

When a ***movie***[26] is displayed, each film frame is flashed on a screen for a short time ***(nowadays, usually 1/24, 1/25, or 1/30 of a second; translated as 41.6 ms, 40 ms, and 33.3 ms respectively)***[27] and then immediately replaced by the next one. ***Persistence of vision***[28] blends those frames together, ***producing the illusion***[29] of a moving image. ***There is a point at which a human's visual pathways are fully saturated; it is somewhere around 70 (14.28 ms) — 80 (12.5) fps.*** This makes perfect sense when compared to the fastest human reaction is the ***"blink of an eye" at 13 ms***[30], and the speed of the human nervous system is 4 ms.

> The frame is also sometimes used as a unit of time so that a momentary event might be said to last six frames, the actual duration of which depends on the frame rate of the system[a], which varies according to the video or film standard in use. In North America and Japan, 30 frames per second (fps) is the broadcast standard, with 24 fps now common in production for high-definition video shot to look like film. In much of the rest of the world, 25 fps is standard.
>
> In systems historically based on National Television System Committee (NTSC)[b] standards, for reasons originally related to the Chrominance subcarrier in analog NTSC TV systems[c], the exact frame rate is actually (3579545 / 227.5) / 525 = 29.97002616 fps. [See Information note below] This leads to many synchronization problems which are unknown outside the NTSC world, and also brings about hacks such as drop-frame timecode[d] [***ed. NOTE: a software solution found in computer animation today.***]

[26] https://simple.wikipedia.org/wiki/Movie
[27] http://www.vhsdvdfilmtransfer.com/blog/frame-rates-simplified.php
[28] https://en.wikipedia.org/wiki/Persistence_of_vision
[29] https://en.wikipedia.org/wiki/Optical_illusion
[30] http://news.mit.edu/2014/in-the-blink-of-an-eye-0116

In film projection, 24 fps is the norm, except in some special venue systems, such as IMAX digital[e], Showscan (60 frames per second – 2.5 times the standard speed of movie film)[f] and Iwerks 70 (in which 30, 48 or even 60 frame/s have been used). Silent films and 8 mm amateur movies used 16 or 18 frame/s. Flash animations and games range from a minimum of 12 up to 60 frames/s — 15 and 32 are most common.

[a] https://en.wikipedia.org/wiki/Frame_rate
[b] https://en.wikipedia.org/wiki/NTSC
[c] https://en.wikipedia.org/wiki/Chrominance_subcarrier
[d] https://en.wikipedia.org/wiki/SMPTE_timecode
[e] https://en.wikipedia.org/wiki/IMAX
[f] https://en.wikipedia.org/wiki/Showscan

***NOTE:*** In actual practice, the master oscillator is 14.31818 MHz, which is divided by 4 to give the 3.579545 MHz color "burst" frequency, which is further divided by 455 to give the 31,468.5275 kHz "equalizing pulse" frequency, this is further divided by 2 to give the 15,734.2637 Hz "horizontal drive" frequency (also the horizontal line rate), the "equalizing pulse" frequency is divided by 525 to give the 59.9401 Hz "vertical drive" frequency, and this is further divided by 2 to give the 29.9700 vertical frame rate. "Equalizing pulses" perform two essential functions: 1) their use during the vertical retrace interval allows for the vertical synch to be more effectively separated from the horizontal synch, as these, along with the video itself, are an example of "in-band" signaling, and 2) by alternately including or excluding one "equalizing pulse", the required half-line offset necessary for interlaced video may be accommodated.

## Animation Recommendations

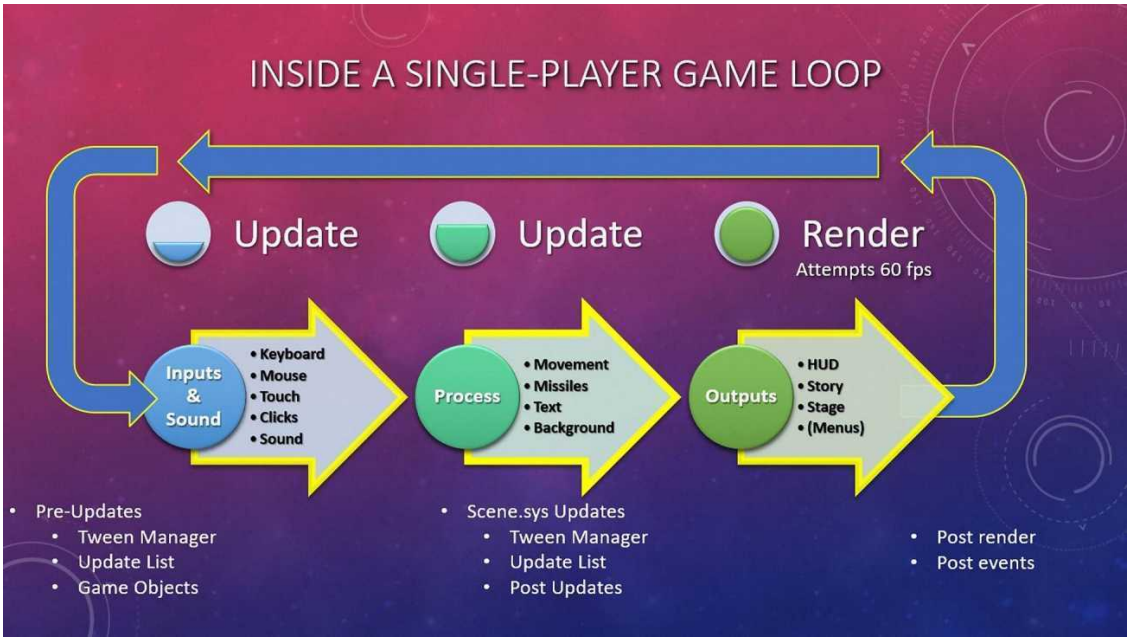
Let's say you have 300 fps, what that means is the Graphics Processing Unit (GPU) is rendering 300 frames per second. However, it's not sending all of those frames to the display monitor. The GPU sends frames, that are partially overwritten, without any form of sync. So, depending on the frame rate:

- If your frame rate is higher than the monitor, parts of the rendered frames will never be sent — sometimes entire frames are lost, or never sent. Generally, this is great for performance — it tends to reduce input and display lag, but it makes your GPU work at nearly 100% with no graphical benefit to speak when compared to other options. If your GPU output is 300 fps but only 144 of them are displayed that translates into 52% of the graphical workload was lost. You're almost guaranteed that some frames will be completely dropped (about 1/3). There are more efficient ways to reduce input lag.

- If your frame rate is exactly equal to your monitor rate (which is not quite possible without sync), all frames sent to the display are new, and all rendered frames are sent — although they are sent in separately displayed frames, leading to tearing, probably the worst tearing you could achieve since it would theoretically always cut the screen at the same height, and thus very visible.
- If your frame rate is lower than the monitor's rate, those frames contain parts that were already sent to the display on the last refresh cycle.

***Exercise:*** Test your browser here https://www.testufo.com/gsync
***Exercise:*** Read more here[31]

## Frame Rates Recommendations




***Frame rate and human vision***[a]

The temporal sensitivity and resolution of human vision varies depending on the type and characteristics of visual stimulus; and, it differs between individuals. (***Is the dress White or Blue***)[b] The human optical system can ***theoretically*** process 1,000 separate images per second; but, is not noticeable to the untrained eye ***after about 150 and up to around 240*** where motion looks realistic. [Chapter 5 footnote #2]

Modulated light, such as a computer display, is perceived as stable by the majority

[31] https://www.blurbusters.com/gsync/gsync101-input-lag-tests-and-settings/

of participants in studies when the rate ***is higher than 50 Hz through 90 Hz***. This perception of modulated light "as steady" is known as the flicker fusion threshold[c]. However, when the modulated light is non-uniform and contains an image, the flicker fusion threshold can be much higher[d]. [Chapter footnote 3]

With regard to image recognition, people have been found to recognize a specific image, in an unbroken series of different images, each of which lasts as little as 13 milliseconds[e]. Persistence of vision[f] sometimes accounts for very short single-millisecond visual stimulus having a perceived duration of between 100 ms and 400 ms. Multiple stimuli, that are very short, are sometimes perceived as a single stimulus, such as a 10 ms green flash of light immediately followed by a 10 ms red flash of light perceived as a single yellow flash of light[g].

[a] https://en.wikipedia.org/wiki/Frame_rate
[b] https://en.wikipedia.org/wiki/The_dress
[c] https://en.wikipedia.org/wiki/Flicker_fusion_threshold
[d] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4314649/
[e] http://link.springer.com/article/10.3758%2Fs13414-013-0605-z
[f] https://en.wikipedia.org/wiki/Persistence_of_vision
[g] http://link.springer.com/article/10.3758%2FBF03211193

## Tweens

Earlier, I have presented the secrets of my hero Richard Williams. His secret is 2 movements during 12 frames!. Negotiate with your artists or draft your animations with this secret that I learned from ***Richard Williams***[32] — ***The Animator's Survival Kit Expanded edition (25 Sept. 2012)***[33]

Computer Animation[a] tricks the eye and the brain into thinking they are seeing a smoothly moving object, the pictures should be drawn at around 12 frames per second or faster[b]. With rates above 75-120 frames per second, no improvement in realism or smoothness is perceivable due to the way the eye and the brain both process images. At rates below 12 frames per second, most people can detect jerkiness associated with the drawing of new images that detract from the illusion of realistic movement[c]. Conventional hand-drawn cartoon animation often uses 15 frames per second in order to save on the number of drawings needed, but this is usually accepted because of the stylized nature of cartoons. To produce more realistic imagery, computer animation demands higher frame rates.

Films seen in theaters in the United States run at 24 frames per second, which is

[32] https://www.youtube.com/watch?v=Abkz-oJ3HSs
[33] http://amzn.to/2dSSZ59

sufficient to create the illusion of continuous movement. VCR display at ***29.967fps (Frames Per Second) for NTSC and 352x288 at 25fps for PAL-M***[d]. Peter Jackson's Lord of the Rings series displays at ***48 fps***[e]. The HTC Vive and ***Oculus Rift*** are virtual reality headsets that ***refresh at 90 Hz***[f]. YouTube allowed users to upload videos at 60fps in June 2014. PC gaming monitors can display ***144 Hz through 240 Hz***[g]. 240fps is near the limits of perceivable smoothness. Interpolated 300 FPS along with other high frame rates have been tested by ***BBC Research for use in sports broadcasts***[h]. 300 FPS can be converted to both 50 and 60 FPS transmission formats without major issues. ***300fps is also the maximum frame rate for the HEVC format.***

[a] https://en.wikipedia.org/wiki/Computer_animation
[b] http://amzn.to/2ecIHrO
[c] http://amzn.to/2elnVqz
[d] http://www.divx-digest.com/articles/vhs_capture.html
[e] https://en.wikipedia.org/wiki/Computer_animation
[f] https://en.wikipedia.org/wiki/Computer_animation
[g] http://120hzmonitors.com/monitor-list-120hz-144hz-165hz-200hz-240hz/
[h] http://downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP169.pdf

***Hint:*** Should I tell my high-school student? or let his parent pay the monthly credit card payment PLUS interest???

***Exercise:*** Study this ***game developer's use of tweens in Phaser***[34]. This guy is a genius!

## 6.6 Camera & Viewports

***Exercise:*** Study this implementation of ***Camera and Viewports***[35] in a multi-player environment. His source code is available from ***GitHub***[36]

A Camera is your ***"`viewport`"*** into the game world. It has a position and size properties and renders only those visual objects within the ***"`viewport`".*** The game will create

[34] http://jppresents.net/games/memory/
[35] https://phaser-multiplayer-game.herokuapp.com/
[36] https://github.com/xicombd/phaser-multiplayer-game

automatically a single (Stage sized) camera upon boot-up. Use ***"Phaser.Camera.x"*** or ***"Phaser.Camera.y"*** to Move the camera ***"viewport"*** around the world.

See how I use Phaser III cameras and viewport in Jigsaw Puzzles as game mechanisms in ***"Making Puzzle Browser Games"***[37]

```
1     new Camera(
2         game,   //reference to the current game
3         id,     //Not supported; but, will have more cameras
4         x, y,   //position of the camera on the grid
5         width,  //same as the Game size
6                 //   and should not be adjusted for now
7         height  //same as the Game size
8                 //   and should not be adjusted for now
9     )
```

The Phaser v2.6.2 Camera supports ***currently supports 32 properties and 15 special effects***[38] through the Game instance global references. You can control the camera via ***"this.camera"*** from any scene, or more specifically via the ***"game.camera"*** if the game has been globally defined — as we already have done on our index page. The Phaser III can support "multiple cameras".

As any movie director knows, camera shots enhance the emotional value of a film and provide "suspended disbelief" — both of which are crucial in game development. This could be the topic of an entirely different book; so, I refer you to these resources to maintain our focus on Phaser development.

- ***"bounds"*** (a ***"Phaser.Rectangle"***) The Camera is bound to this rectangle and cannot move beyond it. It is enabled and initially set to the world's size by default. This viewport rectangle can expose any surface of the world. The values can be anything and are in World coordinates, with 0,0 being the top-left of the world. If you wish to disable the Camera then set it to "null".
- ***"fx"*** a Graphics object used to handle such view effects as fade and flash.
- ***"lerp"*** (a ***"Phaser.Point"***) This is a linear interpolation value and used to follow a target. The default value is 1 which means that the camera will instantly snap to the set target coordinates. A lower value, for example, 0.1, means that the camera will track slowly toward a target resulting in a smooth transition. You can set either the horizontal and/or vertical values independently. You may also adjust these values dynamically in real-time during your game.

[37] http://leanpub.com/mbg-puzzle
[38] https://phaser.io/docs/2.6.2/Phaser.GameObjectFactory.html

- ***"target"*** (a ***"Phaser.Sprite"***) tells the camera to track the designated sprite; otherwise, the target is set to "null".
- ***"view"*** (a ***"Phaser.Rectangle"*** ) This is the viewport into the world; by default, it is the game dimensions. The "x" and "y" values are in world coordinates, ***not screen coordinates***; the "width" and "height" is the dimension and quantity of pixels to render. If ***"Sprite.autoCull"*** is set to "true", sprites are not rendered if positioned outside of this viewport.
- ***"fade(color, duration, force)"*** This creates a camera fade effect. It works by filling the viewport with a color specified, over a set duration in seconds, and ending with a solid fill of the color specified initially. The game will remain filled at the end of this effect. To reset this operation, you can call ***"Camera.resetFX"*** to clear the fade effect. Or you could call ***"Camera.flash"*** with the same specified color as the fade operation; doing so will reverse the process (fading the color's "alpha to 0"), bringing the game back into view again. When the "fade" effect ends the signal ***"Camera.onFadeComplete"*** is dispatched.
- ***"follow(target, style, lerpX, lerpY)"*** Tells the camera which target sprite to follow. If a slight "jitter" effect is observed when following the target, it is probably the results of sub-pixel rendering concerning the sprite's position. This can be disabled by setting the following code snippet to force full pixel rendering. Set `unfollow()` to stop following the targeted sprite.

```
game.renderer.renderSession.roundPixels = true
```

***NOTE:*** Using any of the camera's "shake" features consumes battery power significantly on mobile devices.

## 6.7 Summary

Another intensive chapter! Are you getting your money worth? Good!
Here's what we covered in this section:

* Dove into the Phaser Library and discover various items in a "Bottom-Up development" approach.
* Studied the Input Manager
* Research 26 `game.add` objects
* Deploy sprites and images in the typical game states.
* Deployed sprites and sprite sheets
* Studied the game.make
* Research 40 artwork resources
* Create control mechanisms for keyboard, touch, tap, customized button, and device buttons.
* Studied deprecated keyboard issue.
* Created pointers (mouse, touch, and tap.)
* Learned how to adjust control mechanism timing.
* Created, text, debug feedback, and Heads Up Displays.
* Created Tilemaps.
* Studied several innovative game developments.
* Discovered resources to build unique game worlds.
* Researched building a Game dynamic World editor.
* Created audio for a game.
* Distinguished various audio formats and which to use.
* The discovered mobile issue with the audio drivers.
* Determined whether customized font was essential.
* Learned secrets of computer animations
* Studied hardware capabilities about frame per second.
* Discovered limitations of human optics
* Developed motion cameras
* Discovered visual camera effects similar to movies

## 6.8 Chapter Footnotes:

1. Masson, Terrence (1999). CG 101: A Computer Graphics Industry Reference. page 148, Digital Fauxtography Inc. ISBN 0-7357-0046-X.
2. Paul; Meyer, Mark-Paul; Gamma Group (2000). Restoration of motion picture film. Conservation and Museology. Butterworth-Heinemann. pp. 24–26. ISBN 0-7506-2793-X.

3. FREE BOOK[39]: James Davis (1986), Humans perceive flicker artifacts at 500 Hz, Wiley, PMC 4314649

[39] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4314649/

# 7. Whazzz-sUP! .... HUD Development

***HUD Topics:***

- ***Review Phaser III 68 Plugins***[1]
- Character Inventory & Development Scene (ARRAv15)
- Forum discussions ***here***[2]
- Best, in my opinion, supporting library for ***HUD Development https://www.zebkit.com/***

Variety of **FREE** online tutorial from ***Game Dev Academy:***

- ***How to Create a Game HUD Plugin in Phaser.***[3]
- ***Create a Game UI with the HTML5 CANVAS***[4]

The “heads-up display” is a critical part of your game flow in the “Play Phase”. It is the one item that provides feedback ***on how well a gamer is playing.*** Sure, animations and blood gushing everywhere can be entertaining(?), but the HUD tells the player ***whose blood it is!*** It is the one item in your game that encourages your “customer” to continue “spending their time” in your game; the better your customer feels about their experience the more they spend. Now tell me, have you ever returned to a restaurant that gave you bad service, poor quality, and awful food? Do you ***NOT see a relationship*** between what you’re serving up as a ***“tasty”*** game and other entertainment services? The HUD should encourage, entice, taunt and ***titilize***[5], in “Phaser Render phase”, for more of what’s to come from your game!

[1] https://rexrainbow.github.io/phaser3-rex-notes/docs/site/#list-of-my-plugins
[2] http://www.html5gamedevs.com/topic/15822-building-in-game-ui/
[3] https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47
[4] https://gamedevacademy.org/create-a-game-ui-with-the-html5-canvas/?a=47
[5] http://www.urbandictionary.com/define.php?term=titilizing


**HUD information The Rogue Prince Main Menu**

> Richard Davey said this concerning HUDs, “No it’s too game-specific, anything we provide would only cater for a small set of games. I’d suggest ***you just create a Group and put all your HUD related items in it*** then just keep that Group on the top of your game.” Quote from Phaser.io Forum[a]
>
> [a] http://www.html5gamedevs.com/topic/1924-hud-how-to-implement-it/

***Hint:*** I recommend that this HUD Group float, in a separate layer, above the Tilemap inside the canvas or use the “DOM” feature external to the canvas.

## 7.1 HUD Housing Development

You can have multiple “Phaser III Scenes” all running in parallel. This is similar in my mind as Flash MovieClips running on their main time-line. Each section of your overall stage could be divided into several “Scenes” views. The HUD leads players — with the

information the HUD provides — into decisions about what to do next in the game. HUD placement should enhance gameplay; it should readily display the pertinent information a gamer needs for their avatar's actions, and current gameplay. Here are some suggestions from this excellent ***Game Dev Academy tutorial***[6] showing the "border layout" positioning style.

**Phaser v3.17** has the features I've long waited for — DOM, CSS, load.css, and Load.html. The new DOM elements can appear either above or below the game canvas. The new CSS works to modify the DOM elements. Both act as a typical game object. So, that means external html panels might be a thing of the past, and internally controlled Phaser III panels are the "go-to method". Refer to the following links to discover what you can do!

- About the new ***DOM elements***[7]
- Examples in the ***Phaser III labs.***[8]

**Activate the new Phaser v3.17+ DOM elements**

```
    // Add to the config object
    dom {
      createContainer: true
    }
    /*
    When this is added, Phaser will automatically create a DOM Container
    div that is positioned over the top of the game canvas. This div is
    sized to match the canvas, and if the canvas size changes, as a
    result of settings within the Scale Manager, the dom container is
    resized accordingly.
    */

    // ... . . then inside a scene create
    /*
    You can create a DOM Element by either passing in DOMStrings,
    or by passing in a reference to an existing Element that you wish
    to be placed under the control of Phaser.
    */
    var hud = this.add.dom(x,y,'div',
        // 4th parameter sets the CSS for the DOM element
        'background-color': black;
```

[6] https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47
[7] http://phaser.io/news/2019/05/phaser-3170-released
[8] http://labs.phaser.io/index.html?dir=game%20objects/dom%20element/&q=

```
22          width: 220px; height: game.height;
23          font: 48px Arial',
24          // last parameter is the key name of html DOM
25          'Phaser');
```

## Quote Newsletter 146

"You should ... always, without exception, use explicitly sized HTML Elements, in order to fully control alignment and positioning of the elements next to regular game content.

Rather than specify the CSS and HTML directly you can use the ***"load.html"*** File Loader to load it into the cache and then use the ***"createFromCache"*** method instead. You can also use ***"createFromHTML"*** and various other methods available in this class to help construct your elements.

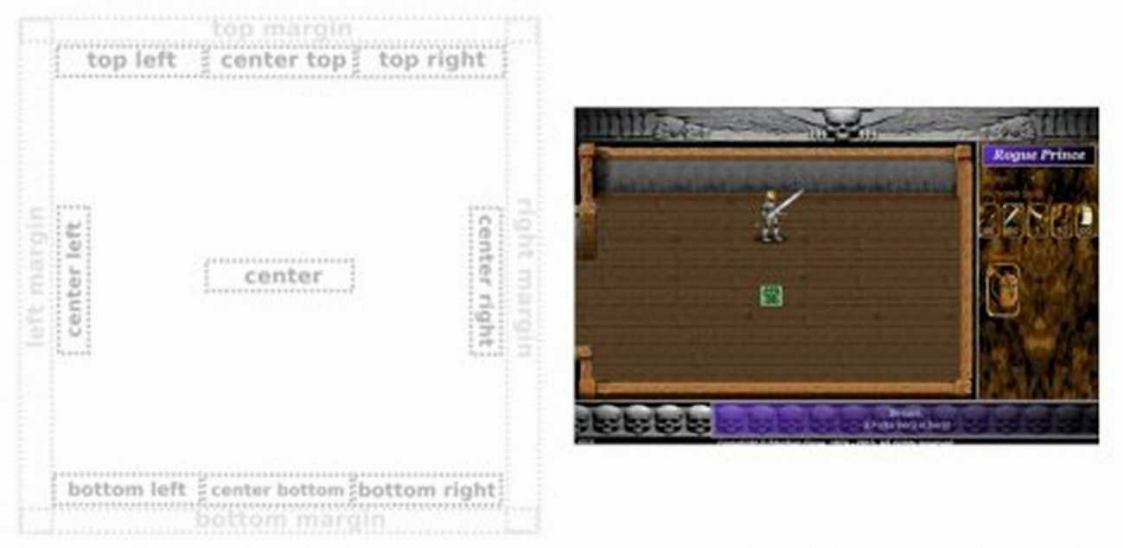
Once the element has been created you can then control **it like you would any other Game Object.** You can **set its position, scale, rotation, alpha, and other properties.** It will move as the main Scene Camera moves and be clipped at the edge of the canvas. It's important to remember some limitations of DOM Elements: The obvious one is that they appear above or below your game canvas. You cannot blend them into the display list, meaning you cannot have a DOM Element, then a Sprite, then another DOM Element behind it."

You can find lots of examples on using DOM Elements in the ***Phaser 3 Examples here.***[a]

[a] http://labs.phaser.io/index.html?dir=game%20objects/dom%20element/&q=

***Exercise: Learn more about the border layout style***[9] from Oracle and Java implementation.

[9] http://www.java2s.com/Tutorial/Java/0240__Swing/WhatistheBorderLayout.htm




**HUD information prior to combat**

The HUD is composed of various items such as text, images, sprites, animations. These should be contained within the new Phaser III DOM element or use a separate Phaser III Scene.

**Sample 8.1: Prototyping a HUD**

```
    //Load images during `preload`
    this.load.image('background', 'assets/images/menubkgrnd.jpg');

    //Assign variable and Add text parameters.
    var playtxt = this.add.text(0, 0, "Play" , style);  // "Play" text

    //Assign text styles and placements with new Phaser III.17 CSS.
    var style = {
        font: "32px Monospace",
        fill: "#C60",
        align: "center"
    }

    //Assign variable and Add text parameters.
    this.scoreText = this.add.text(5, 5,
            "Score (hints off): " + (score * 2), style);

    if(hints){
        //Assign variable and Add text parameters.
        this.scoreText = this.add.text(5, 5,
            "Score (hints on): " + score, style);
    }
```

```
    //Assign variable and Add text parameters.
    this.timeText = this.add.text(5, game.height - 5,
            "Time left: " + timeLeft, style);
            this.timeText.setOrigin(0, 1);
```

## 7.2 HUD as Panels

The illustrations, thus far, have shown HUDs in fixed positions. The HUD could be created into a "group collection or container" as suggested earlier by Richard Davey. A group is a "collection bucket" for any display objects. Groups and containers are treated as sprites with physics and movement. For example, all the children, inside a group collection, are also ***"moved", "rotated", "scaled"*** when its containing parent group is ***"moved", "rotated", "scaled".*** This allows the group to act a sliding panel onto and from the game area using Phaser. Groups are also displayed objects; this means that groups could nest children within larger parent groups. Lastly, groups utilize fast pooling and object recycling.

```
new Group(game, parent, name, addToStage, enableBody, physicsBodyType)
```

Remember the "Dynamic Combat Menu" mentioned earlier? Instead of merely moving buttons in and out of the viewport, we could move an entire Phaser v3.17 DOM element instead of a "group" with its buttons, graphics, and text information as a single DOM element. We can create standard html pages and load them into the game cache, and then use them, either above or below. These new DOM elements are managed as we would any typical game object.


**Clicking the Menu buttons reveals HUD Panels as Phaser III Scenes**

Play the ***Rogue Prince***™ ***demonstration here***[10]

## 7.3 HUD Panels outside the Canvas?!?

Phaser helps display an ***"`html5 canvas`"*** element; in Phaser v3.17, we can use the new load.html for DOM elements and load.css into the Phaser cache. Why should we limit ourselves only to HUDs only inside canvas elements? It's a simple matter to leverage our front-end development skills and let Phaser III manage our DOM elements. If you have studied mobile web design, as suggested by ***Josh Morony,***[11] it becomes an innovation to have HUD Panels controlled by the Browser, jQuery, and CSS. However, with the Phaser v3.17 features, we can control those ***same HUD Panels*** from our Phaser framework. It becomes a simple matter to have a sliding HUD information panel on one or both sides of the canvas game.

***Exercise:*** Review examples at ***w3Schools.com:***[12]

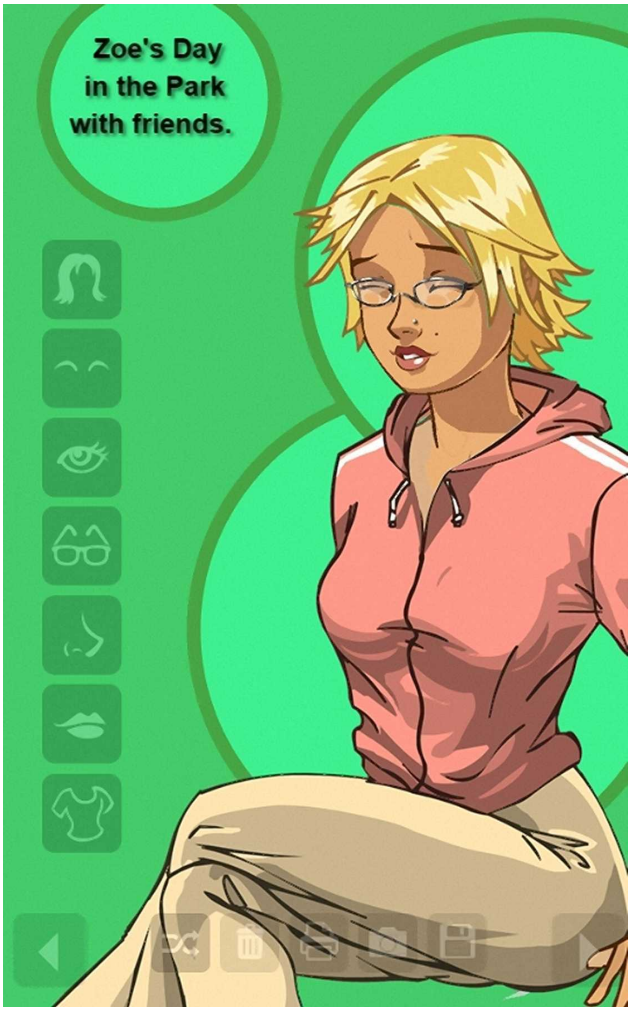
[10] https://makingbrowsergames.com/starterkits/rpg/_arrp-phaser/p3/
[11] https://www.joshmorony.com/mobile-development-for-web-developers/getting-started-with-phaser.html
[12] https://www.w3schools.com/jquerymobile/jquerymobile_panels.asp

- ***Open Panel***[13]
- ***Overlay, Reveal, and Push Panels***[14]
- ***Right-side***[15]
- ***Create a Dialog Modal Plugin in Phaser 3***[16]
- https://jqueryui.com/
- https://codyhouse.co/gem/css-slide-in-panel
- https://codepen.io/jasesmith/pen/raqBpm
- http://wowslider.com/html5-slider-sunny-fade-demo.html
- https://davidwalsh.name/css-slide

Demonstration of ***external panels and PIXI game canvas here.***[17]

[13] https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_basic
[14] https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_display
[15] https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_rightpos
[16] https://gamedevacademy.org/create-a-dialog-modal-plugin-in-phaser-3-part-1/?a=47&campaign=Phaser3GamePrototyping
[17] https://www.merixstudio.com/skytte/

## 7.4 HUD Demos



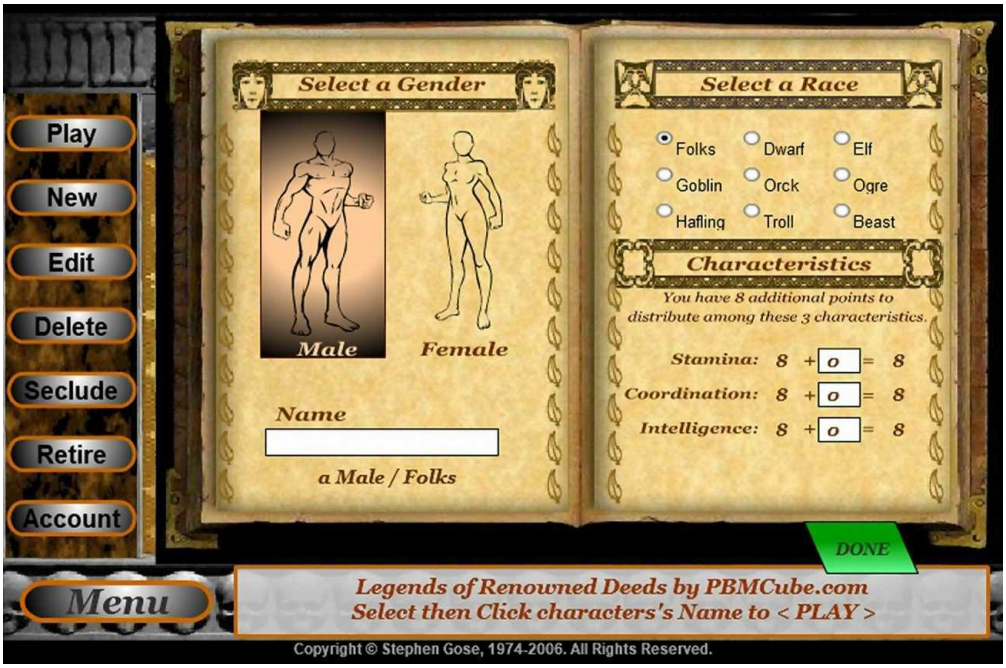

***Zoe Dress-Up (v3.16+)***

**Example 8.3: HUD Menu Grouping**

```
    function HUD(game, parent, name, useStage) {
        var hudPanel = this.physics.add.staticGroup();

        //Your elements here text information displaye.
        //Animated components (i.e., health bars)
    };

    HUD.prototype = Object.create( Phaser.Group.prototype);
    HUD.prototype.constructor = HUD;
```




**HUD as a game editor**

HUD displays provide users feedback, collect menu choices, and modify abstract data structures and properties.


***Troops HUD details in Rulers of Renown™ (RRTE) MMoG***

***Info:*** Several Phaser plugins are available to enhance HUDs. Research ***http://zebkit.org/light/about.html***

## 7.5 Summary

### *Step-by-step guide: Create a Game HUD Plugin in Phaser*[18]

Here's an inventory of what we've learned thus far about HUDs.

- Importance of player feedback in Heads Up Displays.
- Where to place HUD: in separate Groups, on avatars, and map layers.
- What to place in a HUD with examples.
- Code Snippet to generate a HUD.
- HUDs are NOT REQUIRED in the canvas tag only; they could be placed in the DOM and manipulated with standard JQuery, CSS, and JS libraries.
- Learned from other authors using HUD outside the canvas.
- Played a HUD Demonstration.
- Reviewed 5 various HUD Panels and the sample code.
- 23% of Users Abandon an App after 1 use ... and why!
- Methods to increase customer retention and loyalty.
- Best locations to place HUDs
- Found and studied a Phaser HUD plugin.

## 7.6 Footnotes

- Phaser.io Forum[19]
- HUD Manager via plugins.[20]
- Game Dev Academy tutorial[21]

[18] https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47&campaign=Phaser3GamePrototyping

[19] http://www.html5gamedevs.com/topic/1924-hud-how-to-implement-it/

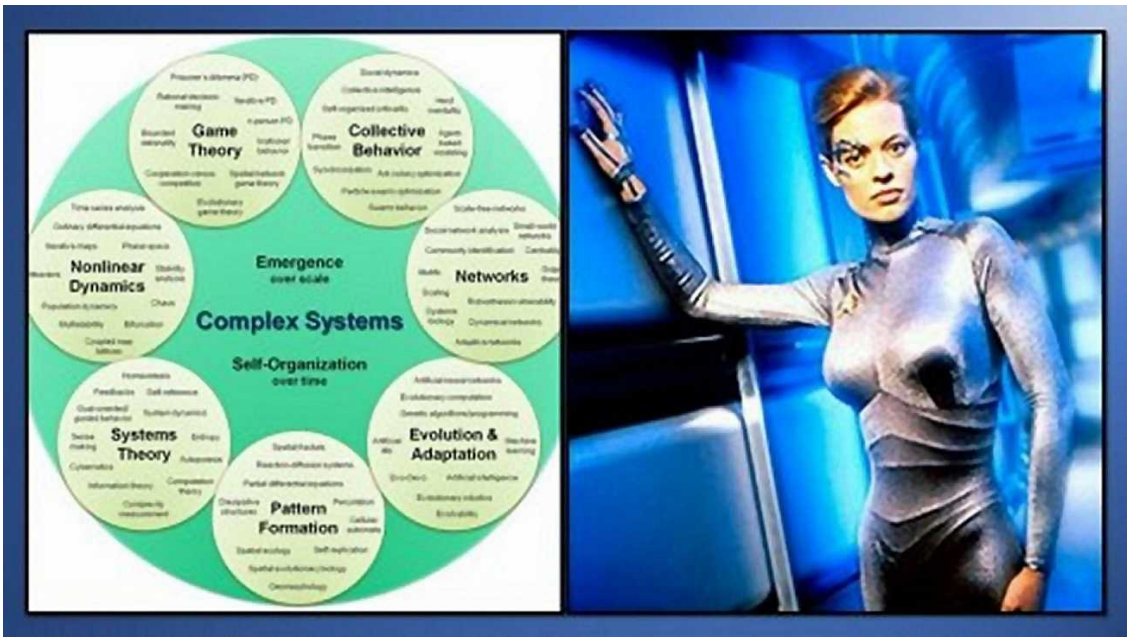
[20] http://phaser.io/docs/2.6.2/Phaser.PluginManager.html

[21] https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47&campaign=Phaser3GamePrototyping

# 8. Don’t make me think or “Artificial Intelligence for Dummies”

So far in our game prototype, “monsters” have just stood there and have taken our punishment bravely. Now, comes the rise of the down-trodden; they’ve gotten mad, saying, ***“I’m a monster, Gaul Dawn it! My life has value. …”;*** they’re “… madder than hell and they’re not going to take it anymore” (movie: 1:40 minutes)[1], a paraphrased quote (with generous liberties) from the movie ***“Network”***[2].

## 8.1 The “6 of 9”




***Not to be confused with 7 of 9***

These are only 6 artificial intelligence (AI) routines[3] we’ll review of the nine (9) in Gaming Theory.

- Chasing: the relentless hunting of a player’s avatar.

[1] https://www.youtube.com/watch?v=rGIY5Vyj4YM
[2] http://www.tcm.com/tcmdb/title/342/Network/
[3] https://en.wikipedia.org/wiki/Artificial_intelligence

- Evading: the "shy retiring" or "withdrawal from" a player's avatar.
- Patterns: are detailed choreographed patterns of movements (aka Kata) practiced either solo or in parts of the whole.
- Fuzzy logic: the random selection of inconsequential decisions.
- Recursive: environmental feedback.
- ***Finite State Machines (FSM).***[4]

## 8.2 Chasing

Is a simple intelligence routine; it involves comparing the player's avatar x- and y-coordinates to the antagonist's and moving closer.

**Example 9.1: Combat Pseudo Code**

```
//Let ex, ey be the enemy x- and y- grid positions.
//Let px, py be the avatar's x- and y- grid positions.
//During the updates, enemy moves to avatar.
if (ex < px){ex += 1;}
if (ex > px){ex -= 1;}
if (ey < py){ey += 1;}
if (ey > py){ey -= 1;}
```

## 8.3 Evading

Is a simple intelligence routine; it involves comparing the player's avatar x- and y-coordinates to the antagonist's and moving farther away.

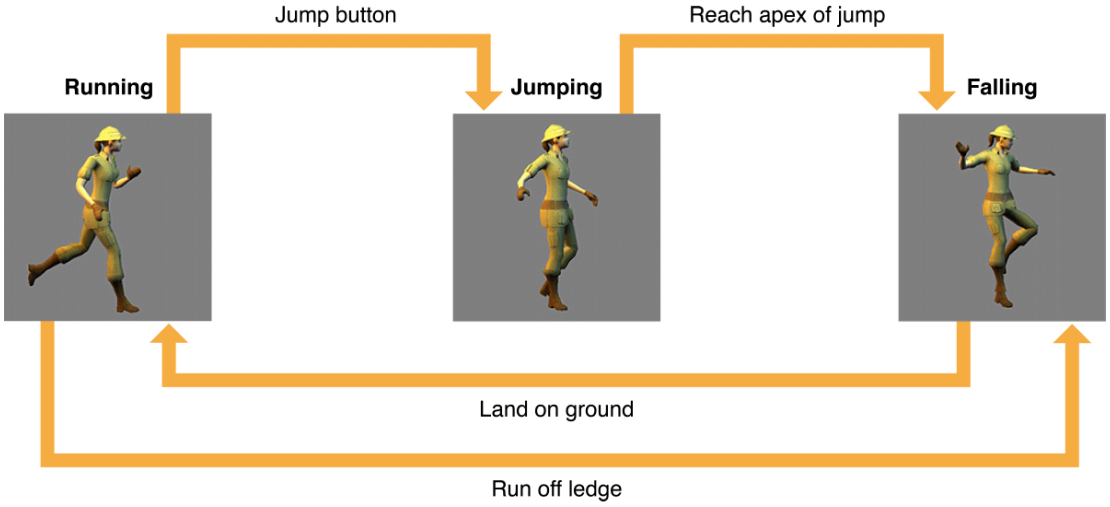
**Example 9.2: Combat Pseudo Code**

```
//Let ex, ey be the enemy  x- and y- grid positions.
//Let px, py be the avatar's  x- and y- grid positions.
//During the updates, enemy runs from avatar.
if (ex > px){ex += 1;}
if (ey > py){ey += 1;}
if (ex < px){ex -= 1;}
if (ey > py){ey -= 1;}
```

[4] https://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf

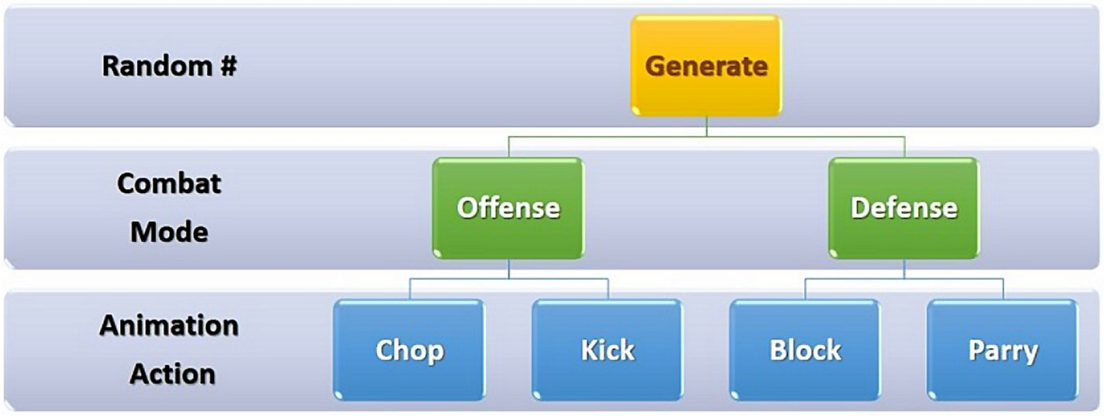
## 8.4 Patterns

There are dozens of Phaser tutorials about artificial intelligence patterns. And if you squint really hard[5], you might see that these AI patterns are similar to "Key Frame Animation" and their associated atlas.[6]




**Apple Gameplay Kit: Animation state machine**

For a combat example, consider this pattern:




**Random numbers to select a patterned response**

- "Patrolling" monsters[7] is a simple AI pattern discussed in this tutorial. You might want to compare another method to implement patrolling here.[8]

[5] https://en.wikipedia.org/wiki/Squint
[6] https://www.joshmorony.com/how-to-create-animations-in-phaser-with-a-texture-atlas/
[7] https://phaser.io/news/2016/04/patrolling-enemy-ai
[8] http://www.emanueleferonato.com/2015/03/05/create-an-html5-game-like-drop-wizard-with-phaser-patrolling-enemies/

- "Flocking" is another AI pattern discussed here with Phaser examples.[9]
- "A*" (aka A-star) is a method to determine movement along paths to any destination.
- http://www.easystarjs.com/ is the ***most popular Phaser plugin*** to use for path finding or
- perhaps you might find this tutorial[10] to your liking. Their ***Phaser Plugin***[11] is open-source.
- GameDevAcademy has my favorite tutorial on Path-finding here.[12]
- My favorite author wrote an interesting article[13] on the use of "antenna feelers" for a top-down racing game.
- Here's my low IQ routine[14] used in Ruins of Able-Wyvern™.

**Example 9.3: Enemy mirrored movement**

```
    //frame refresh and display updates
    var speed = 250;
    this.player.body.velocity.x = 0;
    this.player.body.velocity.y = 0;
    this.enemy.body.velocity.x = 0;
    this.enemy.body.velocity.y = 0;
    //monitor player's movement input

    //Example 5.3 Enemy AI mirrored movement
    if (this.cursor.up.isDown){
        this.player.body.velocity.y -= speed;
        this.enemy.body.velocity.y += speed;
    }
    if (this.cursor.down.isDown){
        this.player.body.velocity.y += speed;
        this.enemy.body.velocity.y -= speed;
    }
    if (this.cursor.right.isDown){
        this.player.body.velocity.x += speed;
        this.enemy.body.velocity.x -= speed;
    }
    if (this.cursor.left.isDown){
        this.player.body.velocity.x -= speed;
```

[9] https://processing.org/examples/flocking.html
[10] https://phaser.io/news/2016/02/how-to-use-pathfinding-in-phaser
[11] https://github.com/appsbu-de/phaser_plugin_pathfinding
[12] https://gamedevacademy.org/how-to-use-pathfinding-in-phaser/?a=47
[13] http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/
[14] https://www.verywell.com/what-is-considered-a-low-iq-2795282

```
        this.enemy.body.velocity.x += speed;
    }
    //Applying fuzzy logic ...
```

***Play AI Combat demo***[15]

***Exercise:*** Find the "Street Fighter" game[16] and determine how your opponent selects its tactics in a single-player game. Does it use a pattern of animation?

## 8.5 Fuzzy logic

What I mean by "fuzzy logic" is doing something randomly, just as we did in some of the previous "Patterns". We can extend this idea, not only to selecting responses but becoming a response itself. There are no reasons why we can't add one more type of behavior to our "enemy's mirrored movement" patterns — namely random motion.

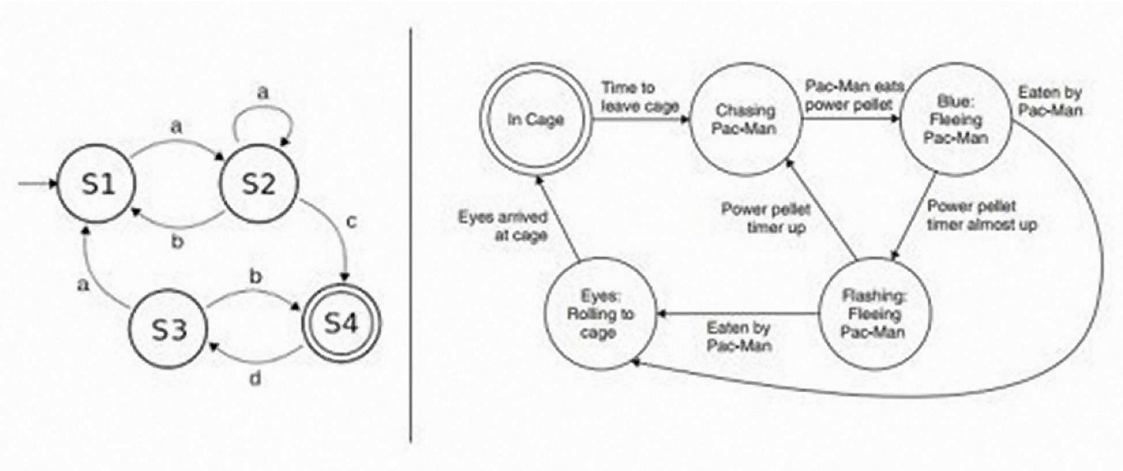
The random movement is used in many games today as one of the many responses an antagonist could take. The "fuzzy logic" comes into play when an opponent selects an action without any or all of the information available — in other words, ***"without perfect knowledge". There'll be no cheating here!*** For example, an opponent is chasing a player's avatar across a room. The avatar steps behind a protective table and stands still, all the while, firing missiles at the antagonist. ***Should the monster just stands there and takes it?! I don't think so.*** Now, if we gave our monster some "fuzzy logic" and some "secret sauce" (which we'll discuss in the combat chapter); the monster may make a "fuzzy" (***aka hair-brain; Sorry, just couldn't pass that one up!***) decision to step around the table and continue chasing the avatar. The direction the monster takes around the table is inconsequential; it makes no difference going around it to the right or left.

## 8.6 Finite State Machines (FSM)

The true power of AI is reached when ***finite state machines (FSM)***[17] are used. We will study more about "how to use FSM" in an opponent's combat actions and decisions later in this chapter and the next chapter on Ruins of Able-Wyvern™.

[15] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/index.html
[16] https://en.wikipedia.org/wiki/Street_Fighter
[17] https://en.wikipedia.org/wiki/Finite-state_machine

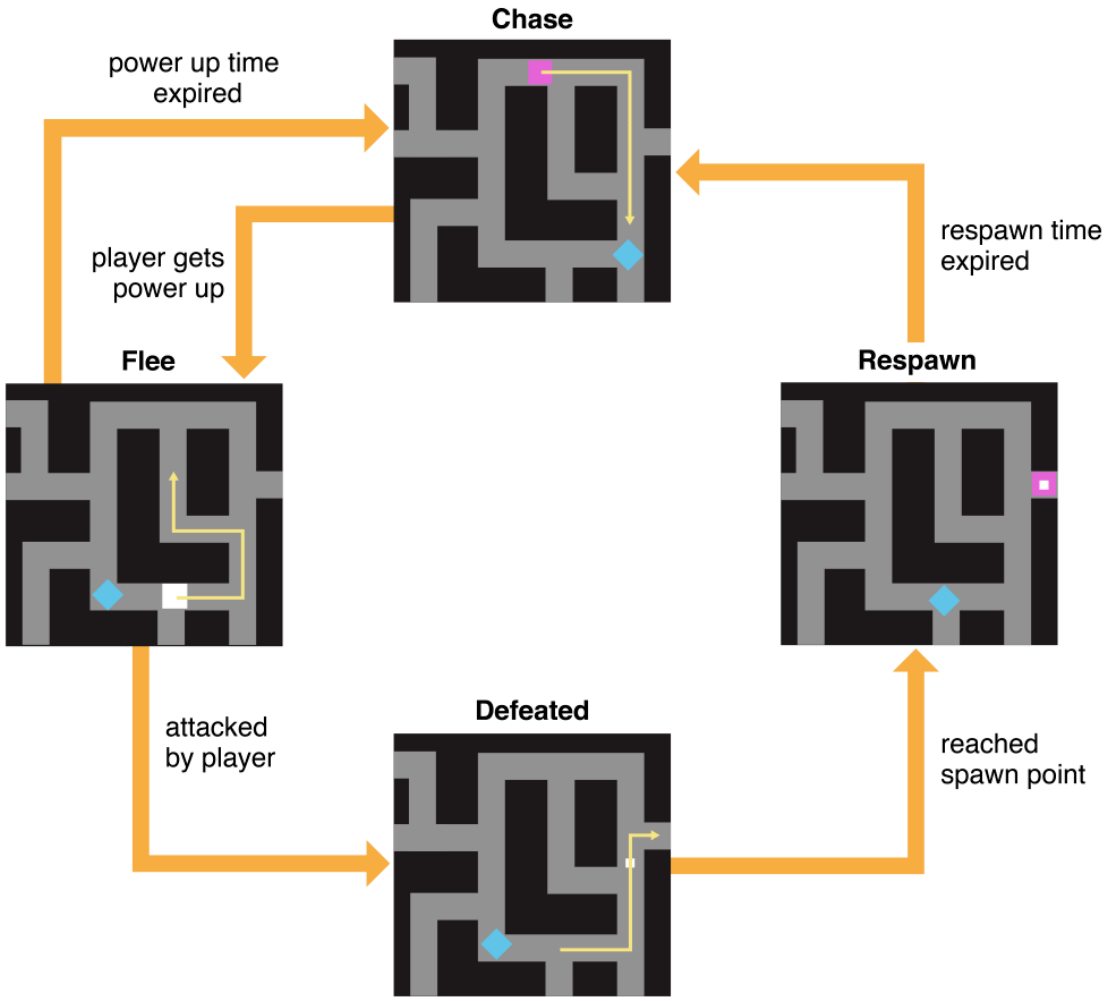
***A finite state machine***[18] is a ***"State and Transition Diagram"*** expressed visually. It is used to show all the states, inputs and outputs, and the event relationships that cause a move into a new state. Transition events are labeled with an input event that triggers the transition and possibly an output that results from that trigger's activation. A "double-circled state" shows the "final acceptance" or "resting" state". We've used FSM since chapter 1; now, we will apply this same idea to opponents' behaviors and reactions.




**Sample FSM and PacMan FSM**

The example above shows an Opponent behavior FSM.

[18] https://en.wikibooks.org/wiki/A-level_Computing_2009/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Finite_state_machines




**Apple Gameplay Kit: Enemy Behavior State Machine**

***Note:*** Here is a tutorial on “How To Design A Finite State Machine”[19], worked out from start to finish.

Machina.js[20] “ … is a ***JavaScript framework*** you could adopt for ***highly customizable finite state machines (FSMs).*** Many of the ideas for Machina.js have been loosely inspired by the Erlang/OTP FSM behaviors.”

## FSM Resolving Combat Outcomes

[19] http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf
[20] http://machina-js.org/

**Example 9.4: Combat Pseudo Code**

```
    //How to find the raw skill
    //(refer to Bonus Content rulebooks for further details):
    //Parrying ability or Thrown Weapon's attack.
    defensive = (ModifiedCoordination) + WeaponSkill

    //Missile & Melee Weapon's attack
    offenseMissiles = (ModifiedCoordination) + BallisticsSkill
    offenseMelee = Stamina + WeaponsSkill
```

How to determine whether a combat action was successful: calculate your character's ability to cause or prevent damage (i.e. parry), do the following:

Hit Percentage = (Characteristic * 2) + (SpecificSkill * 5)

Generate a random number from 0 to 100; if the results are less than or equal to the character's chance to "Hit", the attack was successful. Assign a 1 for success or a zero (0) for a miss. If the results are 10% of what was needed or less (Hit Percentage/10) then the attack was ***unusually powerful and produced "critical" damage.*** Double all damage for these critical hits and assign a 10. Do the same for the defender and cross-reference the attacker's results to the defender's result in the FSM Combat chart. The chart provides the combat outcome. Negative numbers favor the defender in some fashion; positive outcomes favor the attacker. For example, a combat outcome of "9" would mean a powerful "critical hit" attack possibly damaging the defender's armor or weapon that successfully shielded or block the strike. A "-9" would have the same effects described but switching the roles ***"vice versa"***[21] of attacker and defender.

0 - Missed
1 - Success
10 - Critical

| Combat Matrix | | Defender | | |
|---|---|---|---|---|
| | | 0 | 1 | 10 |
| Attacker | 0 | 0 | -1 | -10 |
| | 1 | 1 | 0 | -9 |
| | 10 | 10 | 9 | 0 |

**FSM for Combat Outcome resolution**

Your character's Hit Percentage might exceed 100%. This shows a high level of combat mastery, and the potential to engage multiple adversaries at once. If your character

[21] http://www.lukemastin.com/testing/phrases/cgi-bin/database.cgi?database2=phrases&action=view_product2&productID=288&category2=Latin

has a 40% higher Hit Percentage than his opponent, he may engage an additional opponent in the same ***mega-square.*** Provided that the sum of (each Opponent's Hit Percentage + 40%) is not greater than your character's total Hit Percentage.

There is always a 5% chance that any attack or defense may fail. There is also a chance that your attack may cause unusually high or grotesque damage. As your character's combat skills increase, so will the chance to produce critical injuries. However, the 5% attack failure is fixed. No matter how good one gets at a skill; there is still a chance of failure.

## FSM Resolving AI behaviors

We could stretch FSM into how an antagonist responds to our avatar by using a Combat Tactics selection table. We could set "predetermined" prestigious and repulsion in various ethnic groups, so that, when they encounter each other within a "hostile" environment, a predictable behavior emerges.

```
1    Attitude FSM object for predispositions:
2    var attitude = {hate:-1, neutral:0, like:2}
```

| Pre-disposition | Columns Shifted |
|---|---|
| Hate | -1 |
| Neutral | 0 |
| Like | 2 |

| Actions / Attitudes | Hand to Hand (Grapple) | Fight | Idle | Flight | Talk |
|---|---|---|---|---|---|
| Charge(d) | X | Engage | * | Morale check | Morale check |
| Injure(d) | Morale check | Morale check | * | Morale check | Morale check |
| Winning | Continue | Continue | X | Fire | Continue |
| Losing | Disengage | Disengage | X | Fire | Morale check |
| Change(d) | Fight or Weapons | Move to H2H | Fire | Fire | Fire |

** - Start*
*X - N/A*

**Prejudice chart ties into Action/Attitudes Chart**

If we defined various behavioral actions a character could perform in combat as FSM events, and then compare those actions to combat resulting outcomes states, we have created the ***"Recursive World Feedback"*** described elsewhere. We can further use this FSM behaviors chart to dictate which "Keyed Animation Frame" to use from our avatar sprite sheets.

Let's present some examples:

1. An Elf — known for ancient feuds with Orcs— meets an Orc in combat. Both start in the "idle" FSM state.

- ***The first round of combat*** determines that we shift the encounter left (-1) for the elf; the elf prepares to fight and engage the Orc. The Orc senses a "charge" is imminent from the elf; if the Orc has a missile weapon already at hand, it will fire. If the Orc has a melee weapon ready, it decides to charge also.
- ***The second combat round*** begins with both antagonists engaged in melee combat, and both conduct successful attacks and different defensive moves — the Orc fails its defensive action; the Elf successfully blocked the Orc's attack. Orc is now injured and checks its morale to remain in combat; Elf senses it is winning.
- ***The third combat round begins.*** The Orc's "morale checked" failed and attempts to withdraw from combat; it forfeits its attack to "disengage"; but, was successful in its defense. The Elf senses the disengagement; it was able to land a "critical" attack but missed its defensive maneuver. The Elf's "critical attack" shatters (FSM 9) the Orc's shield into splinters — the shield is now useless, and the Orc receives the remaining bodily damage. The Orc is "losing" and "morale checks" now enforce "disengagement" or possibly "flight" from combat.
- ***Fourth combat round.*** Elf has a higher movement rate than the Orc and is becoming overconfident in the victory. The Elf engages in "hand to hand" combat and is successful! The two antagonists are now wrestling and grappling on the floor. The Orc — having superior strength and the advantage in "hand to hand" combat— achieves a successful attacking strangle-hold on the Elf. The Elf misses its "defensive wrestling move" and receives a "critical" and mortal choke from the Orc. The Elf is dead.

2. Another Elf meets a "dark" elf in combat. Being there is a kinship being the two, they begin a conversation and talk about their encounters thus far.

- ***Second combat round.*** The high elf makes a remark concerning the dark elf's "human" mother. This does not sit well with the dark elf who successfully fires an arrow that "hits" the high elf.
- ***Third combat round.*** High elf is wounded and "check morale" which failed. The high elf retreats from the combat encounter followed by a "hail of arrows".

**Example 9.5: New Combat States Module Added**

```
    //FSM Actions & Animations chart
    var AAstates = {
        'H2H':      Combat.H2H,
        'Fight':    Combat.Fight,
        'Idle':     Combat.Idle,
        'Flight':   Combat.Flight,
        'Talk':     Combat.Talk
        //etc .... add more? ....
        };
        for(var state in AAstates){
            //add combat FSM to Phaser
            game.scene.add(state, AAstates[state]);
        }
```

## 8.7 Recursive World Feedback

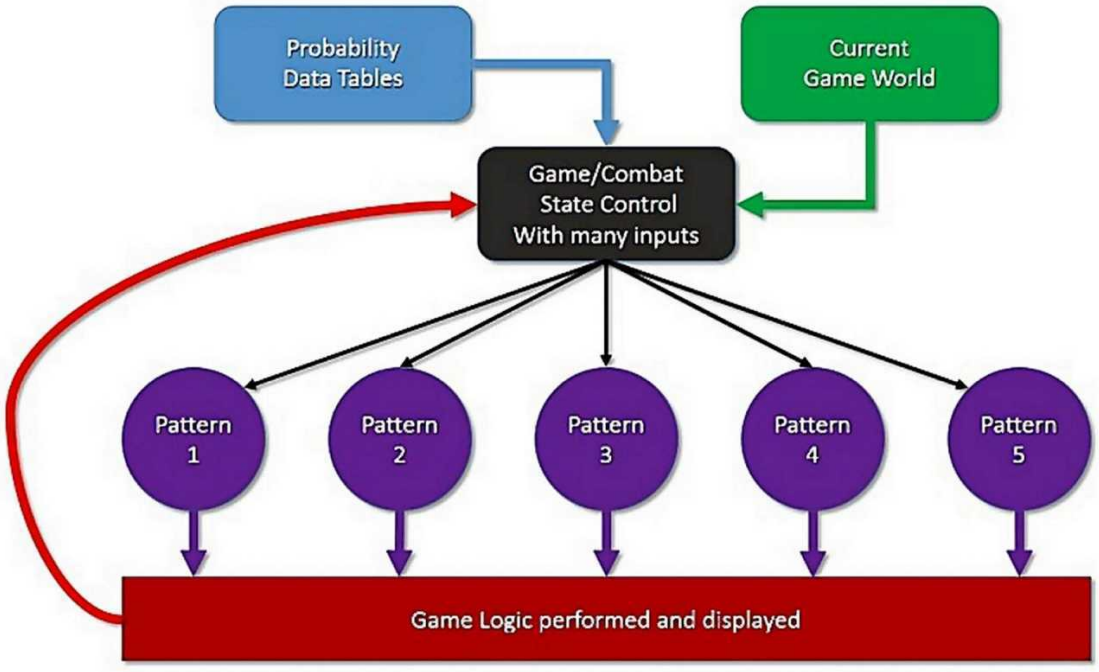
Let's get to the fun part of AI — ***the environment-driven Finite State Machine (FSM)!*** We could force an FSM to make state changes based on a set of parameters that have to do with the game world environment itself. In other words, we control the FSM ***with the very environment's current conditions;*** just as our own brains act and react to our surrounding environment. This would provide a challenging game **that is always different!**

Here's how it would work. We have several different AIs mentioned; we could instead use these and the "current combat conditions" as an input into our game's FSM. For example:

1. If the player's avatar is nearby, we could switch over to one of the random "Patterns AI" of patrolling.
2. If the player's avatar is ***NOT*** nearby, we could have the opponents hunt them down using the "Chase AI".
3. If the player's avatar was firing a hail of missiles at our poor little monster, we could switch to an "Evade AI" only if it is loosing "hit point". Otherwise, if nothing is "striking", our poor little monster turns ***MONSTROUS*** and charges!
4. Finally, if none of these premises are met, we could simply just switch over to a "Random AI" fuzzy state.

We could add one more consideration I call the "preemptive state control". This fancy terminology simply means changing states before all the involved actions are

completed. This is similar to the idea of the switch "Key Frame Animations" from walking to dying — one animation series is interrupted during its sequence and changed into a new series of action animations.




**Our recommended AI integration**

## Probability Data Tables

What's a "Probability Data Tables"? Let's start with an example using a deck of cards. In a deck of poker cards, you should find Ace to 10, and 3 royalty cards per each suite of which there are hearts, spades, clubs and diamonds, and a couple of "jokers" thrown in for fun. Now, what if you took out all the hearts suite the chances of drawing a card has increased because the overall population has decreased — that is probability tables.

```
1     //Probability Table with 50% of selecting "1"
2     var reaction = [1,1,1,1,1,2,2,3,3,4]
```

***Here's a well-kept secret about Phaser, you can download scripts!*** Think of this: you could download various probability tables — JSON scripts— and have a variety of environmental behaviors as dynamically loaded ***Recursive World Feedback!*** Just imagine what your game could do by dynamically downloading the appropriate `.js` script.

script(key, URL, callback, callbackContext)

The URL can be relative or absolute. If the URL is relative the `Loader.baseURL` and `Loader.path` values will be apprehended to it. If the URL isn't specified the Loader will take the key and create a file name from that. For example, if the `key` is "alien" and no URL is given, then the Loader will set the URL to be "alien.js". It will always add the `.js` as an extension. If you do not want this to happen then provide an URL. Upon a successful load, the JavaScript has turned automatically into a `script` tag and then executed, so be careful what you load! The callback, which will be invoked as the script tag was created, can also be specified. The callback must return relevant `data`. paraphrased from Phaser.io v2.6.2[a]

[a] https://phaser.io/docs/2.6.2/Phaser.Loader.html

## 8.8 Complete AI Prototypes

We have moved the Source Code Appendix onto a website and removed it from the book editions 1-6. This allows us to update code changes dynamically for Phaser v3.24 and the upcoming v3.5 as it nears completion.

## 8.9 Chapter Source Code

https://makingbrowsergames.com/book/index.html

## 8.10 Summary

Here's a review of how smart we've become:

- Gaming Theory has 9 various artificial intelligence method; we studied 6 of 9
- Chasing: the relentless stocking of a player's avatar with code samples.
- Evading: the "shy retiring" or "withdrawal from" a player's avatar with code samples.
- Patterns: are detailed choreographed patterns of movements (aka Kata) practiced either solo or in parts of the whole with several online tutorials.
- Fuzzy logic: the random selection of inconsequential decisions.

- Recursive: environmental feedback.
- Finite State Machines (FSM).
- Learned the relationship between AI Patterns and Keyed Animations.
- Discovered the most popular Phaser plugin for A* (A-star).
- Played an AI sample demonstration.
- Discovered the Recursive World Feedback.
- Discovered what Finite State Machines (FSM) are, and how to deploy it for combat and AI behaviors.
- Created Probability Tables.
- ***Learned about the best-kept secret about Phaser — downloading scripts!***
- Studied how to design an FSM.
- Review sample FSM combat turns.
- Learned how to create probability data tables for gambling games.

## 8.11 Footnotes

- Machina.js[22]
- "How To Design A Finite State Machine"[23]
- "Patrolling" monsters[24]
- Implement patrolling here.[25]
- "Flocking" is another AI pattern discussed here with Phaser examples.[26]
- "A*" (aka A-star) is a method to determine movement along paths to any destination.
  - http://www.easystarjs.com/ is the ***most popular Phaser plugin*** to use for path finding or
  - perhaps you might find this tutorial[27] to your liking. Their Phaser Plugin[28] is open source.
  - GameDevAcademy has my favorite tutorial on Path finding here.[29]
- My favorite author wrote an interesting article[30] on the use of "antenna feelers" for a top-down racing game.
- My favorite author wrote an interesting article[31]

[22] http://machina-js.org/
[23] http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf
[24] https://phaser.io/news/2016/04/patrolling-enemy-ai
[25] http://www.emanueleferonato.com/2015/03/05/create-an-html5-game-like-drop-wizard-with-phaser-patrolling-enemies/
[26] https://processing.org/examples/flocking.html
[27] https://phaser.io/news/2016/02/how-to-use-pathfinding-in-phaser
[28] https://github.com/appsbu-de/phaser_plugin_pathfinding
[29] https://gamedevacademy.org/how-to-use-pathfinding-in-phaser/?a=47
[30] http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/
[31] http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/

# Part III: "Walk-thru" Tutorials & Resources

***Part III*** implements all our Phaser v2.x.x and v3.24+ Game Prototype Libraries in these "Walk-Through Tutorials". These "Step-by-Step" tutorials show how to create ***Game Recipes™*** either manually or with the help of our ***online automation tools***[32].

In ***Part III,*** we'll take our various elements from Parts I & II and combine them in the "Walk-Thru Tutorials". We'll construct:

- RPG maze games,
- code 6 different combat systems,
- heads-up displays (HUD) both internal to and outside of the ***HTML5 <canvas> tag***[33],
- apply 6 different Artificial Intelligence (AI) systems,
- build grids and tiled-maps in Phaser III,
- create other actions with these popular game perspective of 2D, 2.5D, and 3D.

You'll discover how to develop multiple game levels, isometric games using newest features available in Phaser III. All of these techniques and supporting source code are explained in an easy-to-understand manner for game designers to scaffold new skills into JavaScript Game Development.

We have moved all the Source Code onto our book's website and removed it from within book editions 1 to 6. This allows us to update code changes dynamically, reduces the book's retail price, and eliminates re-publishing expenses.

[32] https://makingbrowsergames.com/gameDesigner/
[33] https://www.w3schools.com/graphics/game_canvas.asp

# 9. Game Prototype Libraries

- **Book's website: https://makingbrowsergames.com/p3gp-book/**
- **Phaser Game Prototype Libraries:**
  - ***Phaser v3.16+ Game Prototype Library***[1]
  - ***Phaser v3.24+ Game Consolidated Examples***[2]
  - ***Phaser v2.x.x Game Prototype Library***[3]
  - ***Phaser v2.x.x Game Consolidated Examples***[4]
  - ***Phaser v2.x.x Combat Systems Game Prototype Library***[5]

## 9.1 Walk-through Tutorial Series

***Are you "new" to Phaser JavaScript Gaming Frameworks?***

Start with these introduction tutorials. Then tackle the intermediate ***Game Recipes™.*** This section includes proprietary Game Rule Books and Gaming License ***EULA***[6] included as a part of your book purchase. See the ***"terms of service" here***[7]. ***Your license number is your book invoice number.*** You can learn about further benefits by registering your book's invoice number as your license at ***Renown Games Developers***[8]

The "Walk-through Tutorials" Series (WTTS) are listed below by complexity — "1" = easiest (introduction skills) to "4" = most complex (advanced skills required across several ***"Full-stack" technology***[9] disciplines).

### Introductory (Difficulty Rating #1)

***Official*** Phaser Tutorials.

[1] https://makingbrowsergames.com/book/index10.html
[2] https://makingbrowsergames.com/p3gp-book/_p3-demos/index.html
[3] https://makingbrowsergames.com/book/index9.html
[4] makingbrowsergames.com/book/index12.html
[5] https://makingbrowsergames.com/p3gp-book/index3.html
[6] https://www.renown-games.com/LICENSE.pdf
[7] https://www.renown-games.com/shop/index.php?id=tos
[8] https://www.renown-games.com/shop/index.php?id=prod-developers
[9] https://www.w3schools.com/whatis/default.asp

- ***Making your first platform game (Phaser v2.x.x)***[10]
- ***Making your first platform game (Phaser III)***[11]

***Free Game Developer Courses:***

- ***Phaser III Game Design Course***[12] ***FREE!*** as a bonus with this book's purchase; (valued at $19.99) — join the 100s who've already earned their ***Phaser III Game Developer Certifications.*** Free license included in your Bonus Content.
- ***Phaser v2.x.x Game Design Course***[13] ***FREE!*** as a bonus with this book's purchase; (valued at $19.99) — join the 100s who've already earned their ***Phaser v2.x.x Game Developer Certifications.*** Free license included in your Bonus Content.

## Intermediate (Difficulty Rating #2 to #3)

- ***Kiko Escapes™***[14] — a "Clue Mystery" (aka "Cludo" in the UK) Game **Engine in "pure" JavaScript**. Find and rescue the Fairies' Queen.
- ***Blood Pit™ (Walk-thru tutorial)***[15] and game EULA license ***(valued at $48)***[16] included with your book's purchase — ***a significant 80% discount!*** Blood Pit™ is a "hero vs hero" in-game module (IGM) written in Phaser III as an expansion for the ***Legends of Renown Deeds™***[17] Gaming System. The ***FREE game rule book***[18] will help you ***"deconstruct"*** this game's mechanics. — also sold separately on Amazon.com; search for **"Stephen Gose"**[19].

## Advanced — "The Full Monty!" (Difficulty Rating #4)

***The Ruins of Able-Wyvern™***[20] (ARRA), rv_8 1994-2008 Flash release, is the foundational project for the upcoming ***Rogue Prince Quests™ (IGM)***[21]. We are building an updated Phaser III CMS standalone version.

[10] https://phaser.io/tutorials/making-your-first-phaser-2-game
[11] https://phaser.io/tutorials/making-your-first-phaser-3-game
[12] https://leanpub.com/c/p3gdc/c/Tx4iHQ6m64c5
[13] https://leanpub.com/c/phasergamedesignworkshop/c/3IWDBydPFVj1
[14] https://makingbrowsergames.com/p3gp-book/index-bonus-clue.html
[15] https://leanpub.com/c/bloodpit-wtts/c/ZlApCvzPlvz2
[16] https://www.renown-games.com/shop/index.php?id=arbp
[17] http://legends-of-renown-deeds.com/
[18] https://leanpub.com/tigs/c/f3d6wFdxNdHk
[19] https://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Daps&field-keywords=stephen+gose
[20] https://www.renown-games.com/shop/index.php?id=arra
[21] https://leanpub.com/c/arrp

**The Ruins of Able-Wyvern™*** is an introductory game of man-to-Hero tactical combat with arcane weapons in the ***Adventurers of Renown™***[22] gaming series. ***This game has been online continuously since 1994 with 18+ million total games plays, hosted on 1,174 websites.*** You can play the original ***flash versions here***[23] or the Phaser v2.x.x ***in-game-module (IGM) here***[24]. The Town of Lake-Shore provides the best place to begin your first-few campaigns. This game is statistically balanced and allows your newest characters to grow in status, skills, and renown before launching into its parent game — the ***Legends of Renown Deeds™: The Hero's Quest (LoRD).***[25] (... also continuously online since 1994!)


**The Ruins of Able-Wyvern™ (ARRA), © 1994 — 2007, original Flash plugin CMS**

***Download and Review*** the source code ***from this website.***[26]

- ***Phaser v2.x.x Lifetime license***[27] (a $48 value included with this book's purchase!)
- JS Modules Listed alphabetically.

[22] https://renown-games.com
[23] https://www.adventurers-of-renown.com/quests/arra.php/
[24] https://makingbrowsergames.com/book/GAMEAPP_rv_8/index.html
[25] http://www.legends-of-renown-deeds.com/
[26] https://makingbrowsergames.com/p3gp-book/
[27] https://makingbrowsergames.com/book/index11.html

- ***Refer to your download Bonus Content Files***
- We use the ***Phaser Editor***[28] to develop the room sub-states.


**Adventurers of Renown: Ruins of Able-Wyvern™ (2010 release)**

- ***Phaser v3.24+ Lifetime license***[29] and a ***Walk-thru tutorial***[30] included with your book's purchase ***at a significant 80% discount!***

## 9.2 References:

- ***Creating Adventure Games On Your Computer***[31] by Tim Hartnell
- ***Modular Game Worlds in Phaser 3***[32]
- ***How to Make a Roguelike Game***[33]
- ***RogueLike Engines***[34]

[28] http://phasereditor.boniatillo.com/
[29] https://makingbrowsergames.com/p3gp-book/_p3-arrav15/
[30] https://leanpub.com/c/bloodpit-wtts/c/ZlApCvzPlvz2
[31] https://amzn.to/2LAkJb5
[32] https://itnext.io/modular-game-worlds-in-phaser-3-tilemaps-3-procedural-dungeon-3bc19b841cd
[33] http://www.gamasutra.com/blogs/JoshGe/20181029/329512/How_to_Make_a_Roguelike.php
[34] http://www.roguebasin.com/index.php?title=RogueLike_Engines

# 10. What's next?

## 10.1 Game Distribution & Marketing

Short Excerpt from the ***Phaser Game Design Workbook***[1].

> **"How to publish a game on the web??"**
>
> ***Quoted from the Unity forum***
> Hello, I have a little problem with the publishing thing. I've created a little "game", which has only one scene and exported it as a web game. OK, now I have an HTML and a Unity 3d files. But, the problem is, **I don't know anything about creating websites, or uploading files to servers.** I know that there are several questions about this, but I just can't understand what to do. I would really appreciate it if someone could explain to me **how to publish my "game" on the web step by step.** By the way, I've created a ***WIX site,*** but I'm not sure if I can put a Unity 3d game in there. ***Read answers here***[a]
>
> [a] https://answers.unity.com/questions/59535/how-to-publish-a-game-on-the-web.html

I forbid my students using ***WIX*** when attempting to "showcase" their Web Developer or Gaming Programming Skills in their portfolios.

### Introduction: 8-Step Deployment Method.

1. Research game publishers. Learn who they are, what games they favor, and who their target audience is. Be careful when analyzing Return On Investments (ROI). ***This article***[2] gives a proper perspective.

[1] https://leanpub.com/LoRD
[2] https://medium.com/@sm_app_intel/a-bunch-of-average-app-revenue-data-and-why-you-should-ignore-it-2bea283d37fc

2. Contact those publishers, discover their submission policies and requirements for Software Development Kit (SDK) usage. Read carefully about surrendering your rights. Learn what game genre peek their interests. However, be careful not to lock yourself in and become entirely dependent on a single company' SDK for your entire marketing strategy. I learned my lesson from the MochiMedia collapse. While there's ***no doubt you should integrate*** a tracking SDK to register conversions, I recommend that you keep tracking and usage analytics technically distinct from your advertising collection management, to remain flexible and be able to switch partners if you are not satisfied with the results.
3. Create your game .... ***(duh!) You can't sell "blue-sky" ideas.***
4. Create ***a domain name and game website.***[3] (**NOTE:** Some ISPs ***include a 1-year FREE domain name***[4] with their web hosting packages.) Demonstrate your game prototype(s) to their buyers in a protected section of your website — as an example ***click here.***[5]
5. Refine your game mechanics. Get ***strangers (non-developers with their "gut reactions")***[6] and other ***indie developers***[7] to play it. An excellent place to find fellow developers is in the new ***Phaser Forum***[8]. They'll find problems you may have overlooked. Naturally, you'll want to fix those errors they find.
6. Deploy the latest obfuscated/compacted game version on your public website.
7. Wait ... wait ... read their feedback and if necessary return to step #1. Otherwise, continue to step #8.
8. Negotiate a contract ***wisely.*** Your new publisher might require the use of their Software Development Kit (SDK).
9. Start your next game project(s). ... return to Step #1

### *Shareably (SBLY)* looking to rent your Phaser games

***SBLY***[a] is looking for new game titles to add to their platform through rental sponsorships ($50 - $100 per month) and possibly non-exclusive or exclusive licensing deals.

SBLY is fairly new to the web game space, but by no means new to publishing. They started in 2015 and have now amassed over 50 million monthly readers across their publishing networks. Finding early success in this space they now believe that bringing on new titles from other developers is a worthwhile investment. Your game will be on their publishing site https://shareably.net although this will transition to a new domain in the coming months.

[3] http://gose-internet-services.net/openvz-vps-2/
[4] http://gose-internet-services.net/domain-names-hosting/
[5] https://makingbrowsergames.com/book/ch2/index.html
[6] https://www.newgrounds.com/games/under_judgment
[7] https://phaser.discourse.group/c/showcase
[8] https://phaser.discourse.group/c/showcase

SBLY's product lead engineer, John Lee, explains: "We like the rental sponsorships because we can quickly see if your game will succeed with low commitment early on and of course, developers are free to do this with many other publishers!

As for games, I'm particularly looking for familiar games (Minesweeper, FreeCell, spider solitaire, matching games, etc) right now, but in the future, we'll be open to all categories so please don't hesitate to reach out so we can get the conversations rolling."

If you're interested, you can reach John at john@sbly.com, or on the Phaser Discord (@jawnwee)

[a] https://shareably.net/about/

## 10.2 Book Review Protocol

***Did you "like" or "learn" anything from this Book?***

If you liked this book, it would mean the world to me if you took just a moment to leave an **honest positive review** on your "book distributor's review page".

**Remember, you can earn commissions by recommending this book through your affiliate links!**

If **your opinion is less than 3 stars out of 5,** please allow me to make amends before you "publically crucify me" and tell me what you believe is missing ***in a personal one-on-one email.*** I've learned that "poor reviews" from "angry readers" were **never revisited and never updated by them** after I've made those corrections in the following book's editions. Those belligerent comments only make the reviewer appear vindictive when the newest edition has those corrections.

E-Mail me at ( **https://leanpub.com/phaser3gameprototyping/email_author/new** ) with the "Subject: Book concerns – (the book title)". Please provide, in your email:

- the page number, and
- supporting evidence from ***primary technical references*** that support your opinion(s). (Sorry, I don't consider "rumors", "crowd following", "comments from forums" and ***any unsubstantiated opinions — even if you do hold a Ph.D. or have "a gazillion years" of experience)***

- allow me ***sufficient time to respond with an updated book edition — which I will email directly to you as a DRM .pdf format.***
- THEN, provide your rating (or update your rating if you didn't follow my request?!) to 3+.

## 10.3 Tell the world about *your* game!

Excellent! You completed this workbook and constructed your game. Do you want to brag about your additional features or unique modifications? Do you have additional features or game "tweaks" you want to show to the world? Then, let me use your creation in up-coming articles ***and book edition updates! Earn the popularity you deserve!***

Use this contact email:

- E-Mail me at (**https://leanpub.com/phaser3gameprototyping/email_author/new**) with the "Subject: Game Show Case - (your game's title)". Please provide, in your email:
- the book you read to develop your bespoke game edition. ***(hint: use your affiliate link and earn commissions!)***
- a list of your game's unique and / or innovations.
- a website URL to play your demo or licensed versions.

# Appendix

Excellent! You completed this workbook and constructed your game. Still hungry for more? Take some time and review the following resources.

# More Resources

Still hungry for more? Take some time and review the following resources.

## JavaScript Garden

***JavaScript Garden***[9] is a **growing collection of documentation about the most quirky parts of the JavaScript programming language. It advises to avoid common mistakes and subtle bugs, as well as performance issues and bad practices, that non-expert JavaScript programmers may encounter on their endeavors into the depths of the language.**

JavaScript Garden does not aim to teach you JavaScript. Former knowledge of the language is strongly recommended to understand the topics covered in this guide. To learn the basics of the language, please head over to the excellent guide on the Mozilla Developer Network.

## Additional Appendices

Available Bonus Content has 76-pages of **external downloads.**

- ***Appendix: Building HTML5 Web Page (3-pages)***[10]
- ***Appendix: Conversion Cheat-sheet (9-pages)***[11] — ***into and from*** Phaser v2.x.x and Phaser
- ***Appendix: Distribution Channels (2-pages)***[12]
- ***Appendix: Game Design Considerations (1-page)***[13]
- ***Appendix: Game Design Overview (3-pages)***[14]
- ***Appendix: Game Resources and References (17-pages)***[15]
- ***Appendix: JS Coding Styles (7-pages)***[16]
- ***Appendix: Networking (9-pages)***[17]

[9] http://bonsaiden.github.io/JavaScript-Garden/#intro
[10] https://makingbrowsergames.com/book/Appendix-buildHTML5webPage.pdf
[11] https://makingbrowsergames.com/book/Appendix-CC.pdf
[12] https://makingbrowsergames.com/book/Appendix-Channels.pdf
[13] https://makingbrowsergames.com/book/Appendix-GD.pdf
[14] https://makingbrowsergames.com/book/Appendix-GameDesignOverview.pdf
[15] https://makingbrowsergames.com/book/Appendix-Game.pdf
[16] https://makingbrowsergames.com/book/Appendix-CodingStyles.pdf
[17] https://makingbrowsergames.com/book/Appendix-Networking.pdf

- ***Appendix: Phaser 3 Resources (2-pages)***[18]
- ***Appendix: Project Management Analysis (13-pages)***[19]
- ***Appendix: Security (5-pages)***[20]
- ***Appendix: US Business Start-ups (4-pages)***[21]
- **FREE** Phaser III ***Game Developer Course***[22]
- ***Facebook Game AI & WebXR Developers***[23]
- ***Facebook Game Developers***[24]

## Other resources:

- ***Phaser FAQ***[25] ***OR Phaser Discord***[26]
- ***Phaser Plugins (.com)***[27] 121+ Phaser Plugins (as of 20191001) for either v2.x.x or III.
- 9-slice: **https://github.com/jdotrjs/phaser3-nineslice**
- ***Pathbuilder***[28] — A tool to build paths for Pathfollowers and path tweens. Draw and edit Lines, Bezier Curves, Splines, and Ellipses during runtime and export them to Phaser.
- ***Weapons Plugin***[29] — A Phaser v3 compatible port of the Weapon Plugin shipped with Phaser v2.x.x. The author considers this plugin is stable and mostly feature complete. Most bugs should be fixed, but if any do occur please help us by reporting them.
- **Phaser 3 Ninepatch Plugin**[30] — Phaser3 Nine Patch plugin adds 9-slice scaling support to Phaser 3 by KoreezGames
- **Generic Platformer and Phaser Bootstrap Project**[31] — Generic platformer and Phaser 3 bootstrap project

[18] https://makingbrowsergames.com/book/Appendix-Phaser3.pdf
[19] https://makingbrowsergames.com/book/Appendix-PM.pdf
[20] https://makingbrowsergames.com/book/Appendix-Security.pdf
[21] https://makingbrowsergames.com/book/Appendix-USBusiness.pdf
[22] https://leanpub.com/c/p3gdc/c/Tx4iHQ6m64c5
[23] https://developers.facebook.com/products/#filter-id=gaming
[24] https://developers.facebook.com/search/?q=html5%20build%20games&notfound=1
[25] https://github.com/samme/phaser3-faq/wiki
[26] https://github.com/phaser-discord/community/blob/master/FAQ.md
[27] https://phaserplugins.com/
[28] https://github.com/samid737/phaser3-plugin-pathbuilder
[29] https://github.com/16patsle/phaser3-weapon-plugin
[30] https://github.com/koreezgames/phaser3-ninepatch-plugin
[31] https://github.com/nkholski/phaser3-es6-webpack

## Selling your Game Assets

This is similar to ***selling a car in parts***[32] from a "hack-shop". See how ***AppGameKit***[33] manages their ***"piecemeal"***[34] sales.

- Sell your knowledge, discoveries, concepts, and designs as ***tutorials, ebooks, guides, and courses (examples here)***[35].
- Sell your ***3D Models here***[36] or from my favorite 3D publisher ***DAZ 3D***[37] download their ***FREE DAZ 3D Studio Manager***[38].
- ***As a Commercial Game Starter Kits***[39] (examples)
- ***Creative Market***[40] — is the world's marketplace for design. Bring your creative projects to life with ready-to-use design assets from independent creators around the world. ***Sell your game assets.***
- ***Envato Elements***[41] — Sell your graphics, audio, images, plugins, themes, and videos.
- ***Game Developer's Market***[42] — GameDev Market (GDM) is a community-driven marketplace based in the UK that connects indie game developers with talented asset creators. There are asset stores specific to certain game engines, huge generic stock websites with an overwhelming catalog of assets of varying quality, and smaller indie stores with small selections of very niche assets.
- ***HumbleBundle for Developers***[43] — Humble Games wants to make it easier for indie developers to succeed. Publishing is their way to give back to the indie community by helping great ideas to become successful games. They want to share their Humble Bundle's experience and international marketing scale to help every developer make more games and reach more customers. You can create your own webpage to showcase and sell your game. ***Hosting is free, and you keep 95% of the proceeds after payment processing and taxes.*** You can edit your page whenever you want, and you also have access to our customer support team. Get many of the features and benefits of their "Humble Gamepage" in a handy widget that ***you can add directly to your site to sell your game.*** Edit and

[32] https://www.cashcarsbuyer.com/how-to-sell-a-car-for-parts/
[33] https://www.appgamekit.com/order
[34] https://www.dictionary.com/browse/piecemeal
[35] https://gamedevelopment.tutsplus.com/tutorials?ref=PBMCube
[36] https://help.sketchfab.com/hc/en-us/articles/115004259063-Selling-your-3D-Models?utm_source=website&utm_campaign=footer
[37] https://www.daz3d.com/community/community-publishing
[38] https://www.daz3d.com/get_studio/
[39] https://codecanyon.net/search?utf8=%E2%9C%93&term=html5+mobile+fighting+game&ref=PBMCube&referrer=search&view=grid
[40] https://creativemarket.com/?u=PBMCube
[41] https://community.envato.com/become-an-elements-author/?ref=PBMCube
[42] https://www.gamedevmarket.net/?ally=GVgAVsoJ
[43] https://www.humblebundle.com/subscription?refc=vu8nCv

customize it at any time, and ***keep 95% of proceeds*** after payment processing and taxes.

# Appendix: Online Game Development

- ***Modd.io***[44] — Make a game in **3 days,** not in 3 months. Modd.io is running on a Multi-Player game engine with Box2d physics built-in. Many features have been put in place over the years such as client-side prediction and bandwidth optimization.

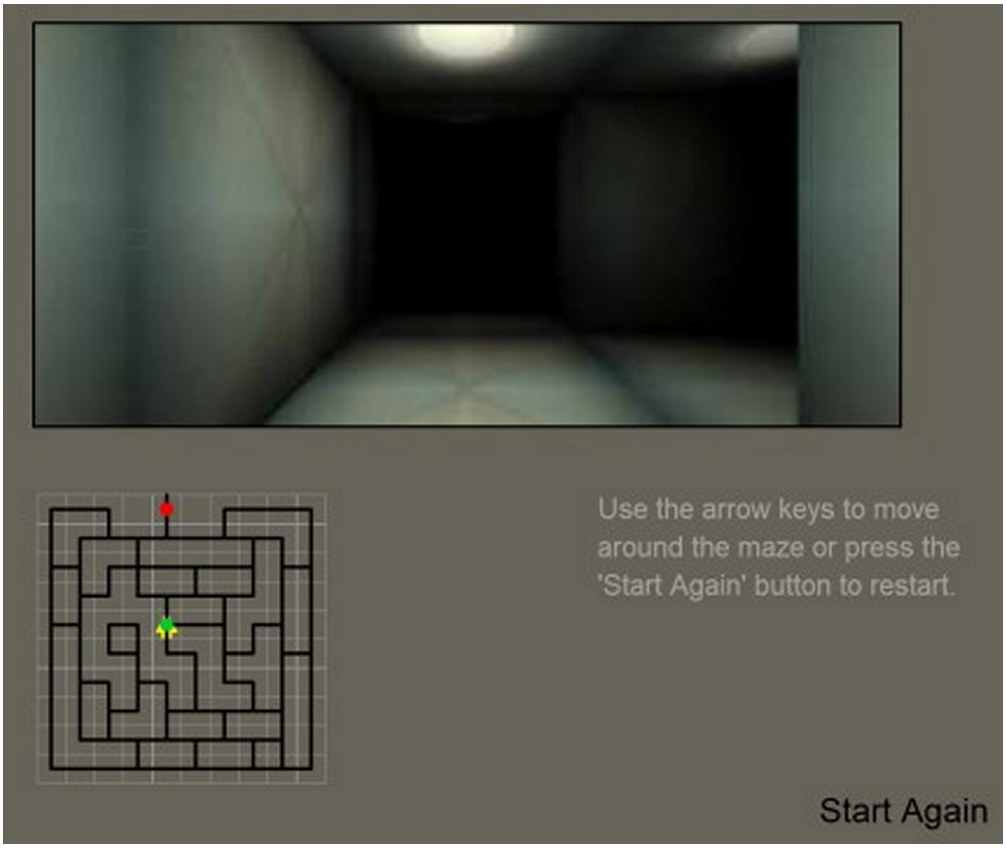
[44] https://www.modd.io/

# Appendix: Making WebXR Games!

Refer to the following pioneers who are building 3D Phaser Games.

- ***Headless Game Design***[45]
- ***CubeVille***™[46] — an Online Life-Simulation Game (and social science research). Alpha concept demonstrations for ***Phaser III here***[47] and ***Phaser v2.x.x here***[48].
- ***WebXR Device API Explained***[49]
- ***Making DOOM 3D in Phaser***[50]
- ***Enabled 3D for Web, Mobile, and PC***[51]
- ***Phaser III 3D Camera Plugin***[52]
- ***Phaser v3.5 Extern Code for Three.js***[53]
- ***34 3d examples in Phaser III***[54]
- ***Build fake 3D HTML5 games with Phaser, Arcade physics, three.js, and Phaser 3D library***[55] advice from Emanuele Feronato
- ***Super-Powers***[56] — **2D and 3D game making for indies; Free and open source.** Superpowers is powered by three.js, Socket.IO, TypeScript, Electron, Node.js, and many other lovely Open Source projects.

[45] http://leanpub.com/hgd
[46] https://pbmcube.com/cubeville
[47] https://daan93.github.io/phaser-isometric-demo/
[48] https://dozenschuiven.daniel-dewit.nl/
[49] https://github.com/immersive-web/webxr/blob/master/explainer.md
[50] https://phaser.io/news/2018/01/making-doom-3d-in-phaser
[51] https://github.com/yandeu/enable3d#readme
[52] https://github.com/photonstorm/phaser/tree/master/plugins/camera3d
[53] https://labs.phaser.io/edit.html?src=src%5Cbugs%5Cextern.js
[54] https://phaser.io/news/2019/04/phaser-backer-examples-april
[55] https://www.emanueleferonato.com/2019/11/27/build-fake-3d-html5-games-with-phaser-arcade-physics-three-js-and-phaser-3d-library/
[56] http://superpowers-html5.com/index.en.html




**First-person view of the 3D environment**

See more about using Phaser III in 3D projects:

- ***Phaser v2.x.x (GitHub)***[57] and ***Phaser III (GitHub)***[58] **Isometric Plugins.**
- ***Phaser III Dev Log #98***[59].
- ***Phaser III 3D examples***[60]
- ***Phaser III / Enable.io integration***[61]

See items that ***"won't be fixed"***[62] concerning ***3D Sprites***[63] — as of 27 APR 2020 — and design your 3D projects accordingly.

[57] http://rotates.org/phaser/iso/
[58] https://github.com/sebashwa/phaser3-plugin-isometric
[59] https://phaser.io/phaser3/devlog/98
[60] https://phaser.io/news/2019/04/phaser-backer-examples-april
[61] https://enable3d.io/
[62] https://github.com/photonstorm/phaser/issues/5091
[63] https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Sprite3D.html

# Appendix: Phaser III Plugins

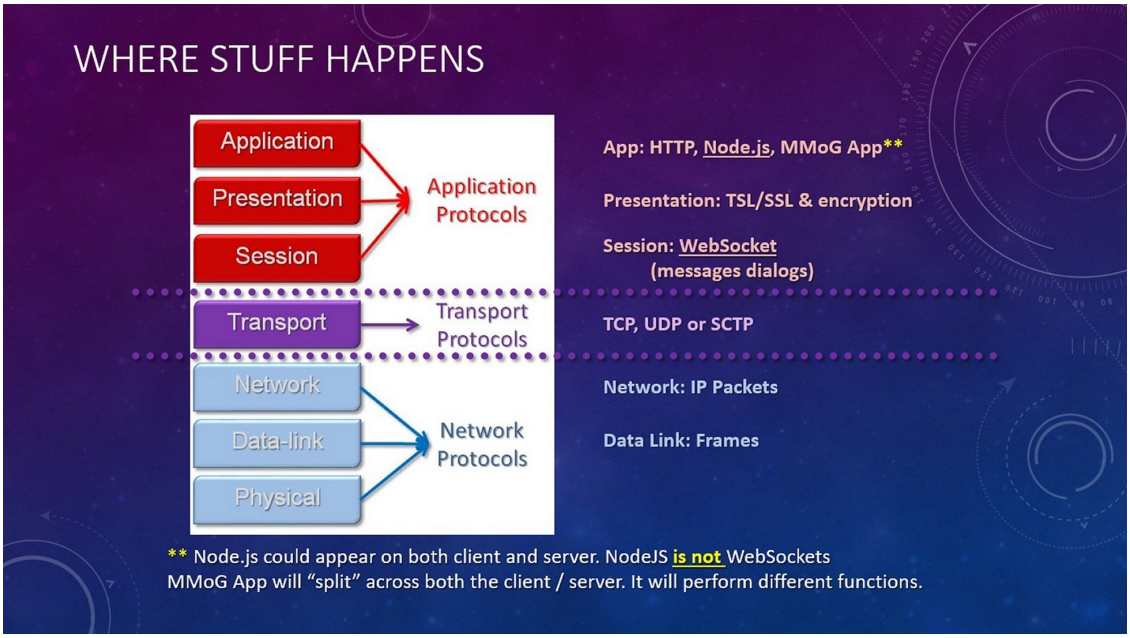
I highly recommend ***phaser3-rex-notes on GitHub***[64] and ***Awesome-phaser***[65] as your first couple of stops for any Phaser III plugins. **Also, there is superior documentation located at https://rexrainbow.github.io/phaser3-rex-notes/docs/site/ Current, as of the last visit, there are the following Phaser III available plugins: 20181211**

- actions (Hexagon and Quad grid)
- audio (fade and midiplayer)
- behaviors (boid, bullet, 8-directions, fade, flash, interception, moveto, pathfollower, rotateto, scale, ship,textpage, texttyping)
- board (board, chess, grid, hexagonmap, match, miniboard, monopoly, moveto, pathfinder, shape utils)
- data (bank, csvtoarray, csvtohashtable, pool, restorabledata)
- gameobjects (bbocdtext, canvas, containerlite, gridtable, shape, tagtext)
- geom (hexagon, quad, rhombus, utils)
- input (button, cursoratbound, drag, dragscale, dragspeed, mousewheeltoupdonw, scroller, slider, touchsursor, touchstate, virtualjoystick)
- loader (awaitloader, webfontloader)
- logic (achievements, conditionstable, fsm, runcommands, waitevents)
- math/gashapon (Gashapon.js)
- shaders (pixelation, swirl)
- string (lzstring, xor)
- time (clock, lifetime)
- utils (... 25 total! in this directory)

[64] https://github.com/rexrainbow/phaser3-rex-notes/tree/master/plugins
[65] https://github.com/Raiper34/awesome-phaser

# Appendix: Network Concepts

This chapter is a sample excerpt from ***Multi-player Gaming Systems***[66]




**Game Scenes**

I debated about this section of the workbook whether to include how networks impact the delivery of games and the reason for implementing these next three sections: Initialize, Boot and Preload (traditional method). There are technical reasons that support this laborious and seemingly redundant part of your game's design. Many Wide-Area Networks (WAN) use the MPLS (Multi-protocol Label Switching) protocol[67]. Briefly, it forces routers to collapse data streams to switching only, thus reducing network delay by 30% to 40%.

***NOTE:*** Want to learn more? I elected to offer those that are interested a coupon to access this content as a ***FREE TCP/IP Networking tutorial***[68]. Access to this course does not have an expiration date, but it is a ***one-time access only*** access. You will need your purchase invoice number as the second part of your coupon's code.

[66] https://leanpub.com/rrgamingsystem
[67] http://www.protocols.com/pbook/mpls/
[68] http://www.tbcube.com/courses/

# Security Concerns

In this section, I will develop more methods to foil piracy attempts on your game.

## Protecting Game Assets

You will spend many hours developing your game; and, as you learned, copyrights only protect the tangible expression of your idea — not the idea itself! Yahoo User Interface team provides a hint in the note below entitled ***Minification vs Obfuscation***[69]. (See Chapter Notes for more references.)

> JavaScript is a load-and-go language. Programs are delivered to the execution site ***as text (not as executable or semi-compiled class files)*** where it is compiled and executed. JavaScript works well with the Web, which is a text delivery system because it is delivered as text.

There are two downsides of textual delivery of programs. The first is code size. The source can contain material (such as white-space and comments) which aids in the human interpretability of the program, but which is not needed for its execution. Transmitting superfluous material can significantly delay the download process, which keeps people waiting. If we could first strip out the white-space and comments, our pages would load faster.

The second downside is code privacy. There might be a concern that someone could read the program and learn our techniques, and worse, compromise our security by learning the secrets that are embedded in the code.

There are two classes of tools which deal with these problems: minifiers and obfuscators.

[69] http://yuiblog.com/blog/2006/03/06/minification-v-obfuscation/

> … Then finally, there is the question of code privacy. ***This is a lost cause. There is no transformation that will keep a determined hacker from understanding your program.*** This turns out to be true for all programs in all languages, it is just more obviously true with JavaScript ***because it is delivered in source form.*** The privacy benefit provided by obfuscation is an illusion. If you don't want people to see your programs, unplug your server.

## Use of <iframe>

The reason behind using an ***<iframe>*** (short for the inline frame) is "… the ultimate modularization tool, allowing you to break up content, seamlessly display content from other sources, and better manage downloading. It's a must-have for any web designer's arsenal." (See footnote 1)

Other possible reasons developers deploy iframes (instead of AJAX) could be:

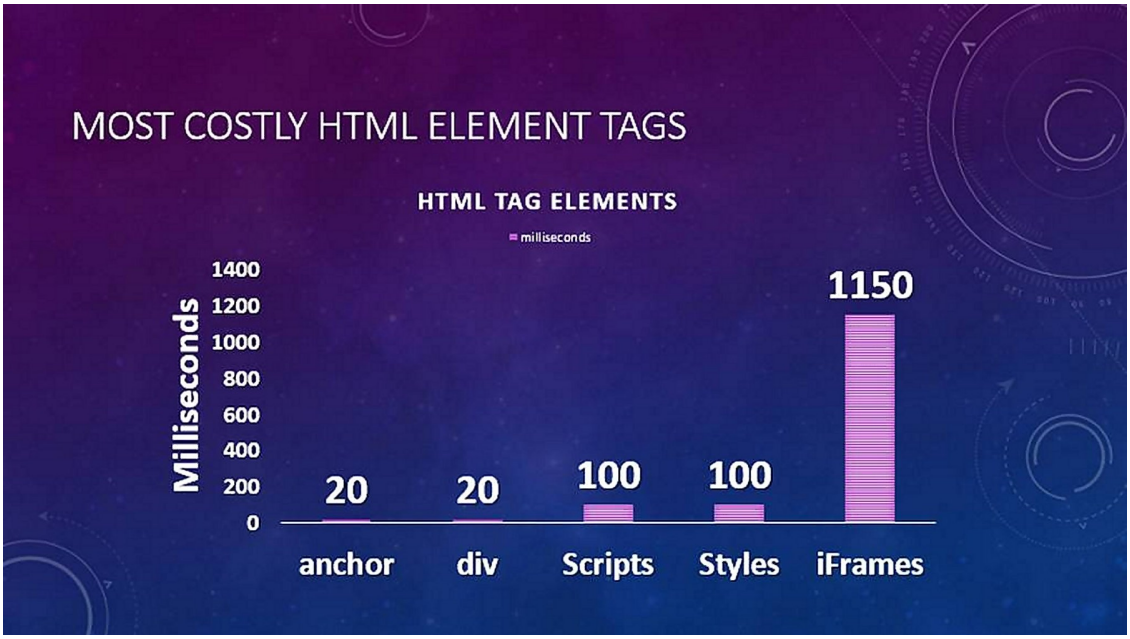
1. Iframes circumvent the cross-domain origin policy; whereas images, scripts, and styles do not. This trick is beneficial for pulling content relatively safely from other domains. The results are presenting foreign content without clobbering one's page content and styling — something JSONP would be able to do.
2. Iframes are not restricted to a single resource mime-type. Google Gmail is a perfect example because it is a set of inline frames.
3. Poor architecture design or resorting to a simple fix.

However, the problem with an ***<iframe>*** is that whenever you navigate to an HTML5 game inside a browser, the game's ***<iframe>*** does not automatically receive "focus". This means the gamer must click on the game's displayed ***<iframe>*** before the game can accept any keyboard input. Annoying? Sure is!

Other web developers point to a number of reasons to further avoid iframe usage[70]. But as scary as they might wail, several game distribution channels insist on using iframes to preserve their website integrity through game content isolation.

Here is a suggested workaround, but remember what we discussed in previous chapters about the window.onload??? And do NOT FORGET that using iframes is the most costly method of securing your content.

[70] http://stackoverflow.com/questions/23178505/good-reasons-why-not-to-use-iframes-in-page-content




**Mostly Costly Web Page elements**

***Read - Using Iframes Sparingly***[71]

```
<script>
    window.onload = function() {
    document.getElementById('frame').focus();
    };
</script>
<iframe id="frame" width="768" height="480" src="yourDomain.com">
```

## Bad Bot!

No this is not about the movie studio[72]. Bots, spiders, spam bots, crawlers search websites for content — your game content you spent hours creating. There are a few minimal steps that should be implemented.

- A “silence is golden” index.htm(l) in every directory you require privacy.
- .htaccess for Linux servers. Here’s the best book, in my opinion,[73] on .htaccess.
- logging

[71] https://www.stevesouders.com/blog/2009/06/03/using-iframes-sparingly/
[72] https://en.wikipedia.org/wiki/Bad_Robot_Productions
[73] http://htaccessbook.com/?ap_id=PBMCube

- Black hole bot trap[74] "for those stubborn, hard to get rid of" bots.

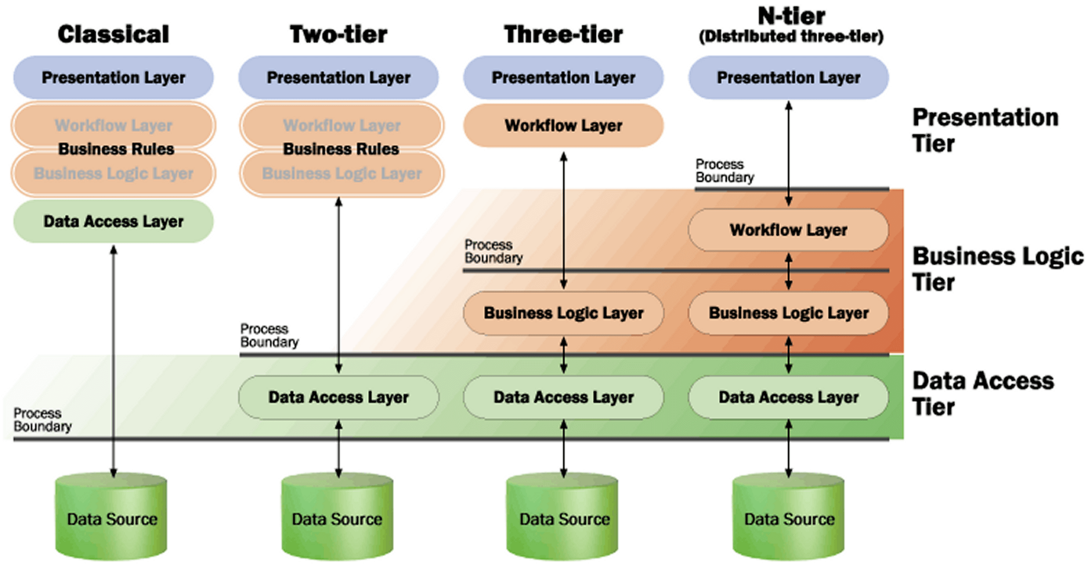
***Exercise:*** Refer the to index.php file and compare it to both the `MMM-js-SPWA/index-mobile-mmm.html` and `MMM-js-SPWA/index-mobile-mmmSPWA.html` (Single Web Page Application). The index.php launches the CodeIgniter application, controllers, system, and views. It is the "mmm" directory/views that have all the page components to dynamically constructor the content delivered to the end-user.

***NOTE:*** how the entire game's JavaScript in the `MMM-js-SPWA/index-mobile-mmmSPWA.html` page is exposed to the end-user! Deploying a game in this manner does not protect my tangible copyrighted idea; I might as well have released it as a public domain or copyleft distribution. In a few days after releasing it in the "wild", thousands of similar "copy-cat" versions — with different graphics ***or not*** — would appear on the Internet.

***Exercise:*** Open and edit the index.html file in `MMM-js0CI3.1.2-v0001`. This would be an excellent place for a "bot trapper" or a "hacker logger". Research Blackhole bot trap version 1[75] and the newest version for WordPress as of Feb 2016[76]. Determine if you need such a system for your game.

[74] https://perishablepress.com/blackhole-bad-bots/
[75] https://perishablepress.com/blackhole-bad-bots/
[76] https://perishablepress.com/new-plugin-blackhole-bad-bots/

## Other Considerations




**N-Tier Cloud Architecture**

The topic concerning network and application security is huge! I recommend the following sites for more details on security.

- 64 Network DO's and DON'Ts for Multi-Player Game Developers[77]
- Storage Area Network Security * sans.org[78] you will find 1,000 of free references, avenues to earn a Cyber Security Master Degree and certifications.

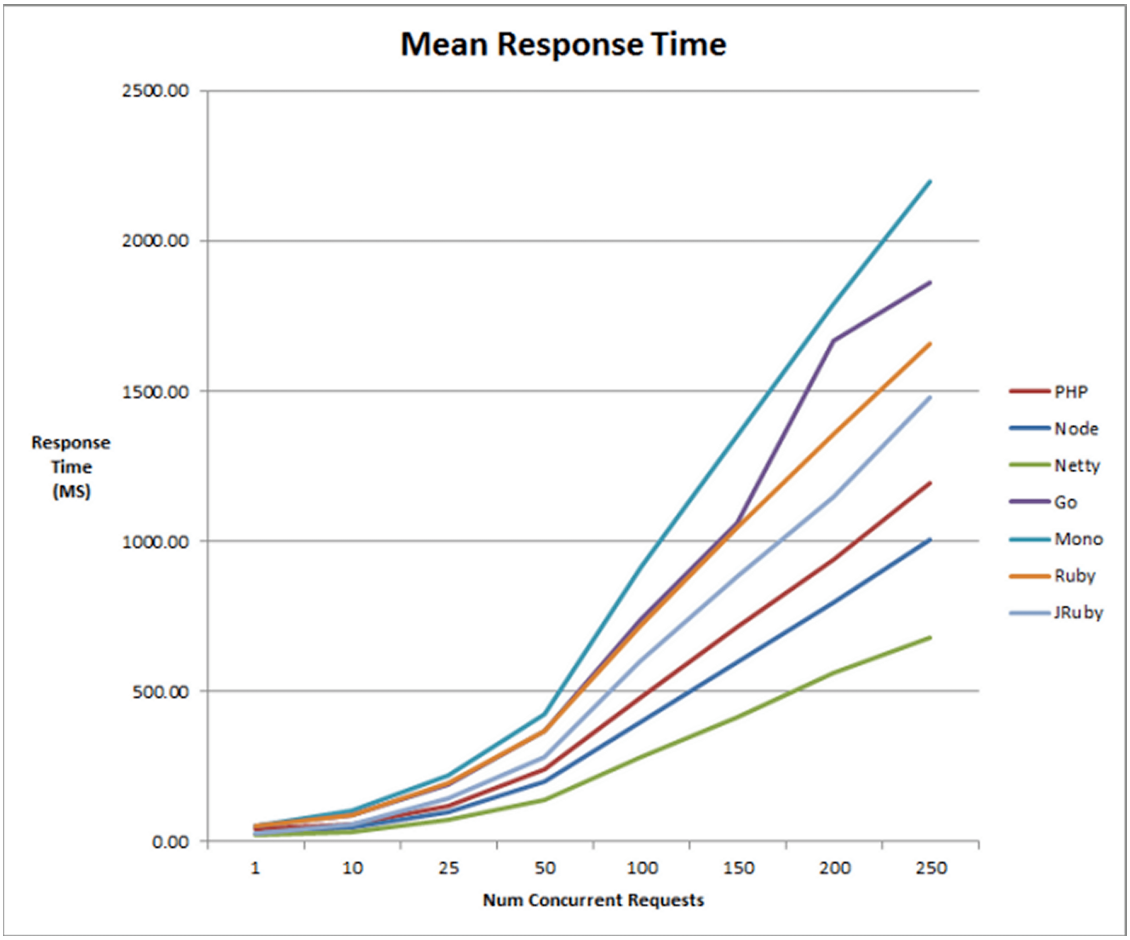
# Game Services (Back-end)

PHP is a popular server-side middleware; there are others available, but I will focus on my own servers and implementations that support Mozart's Music Match. So instead of blindly following my recommendation, let's review several web middleware frameworks.

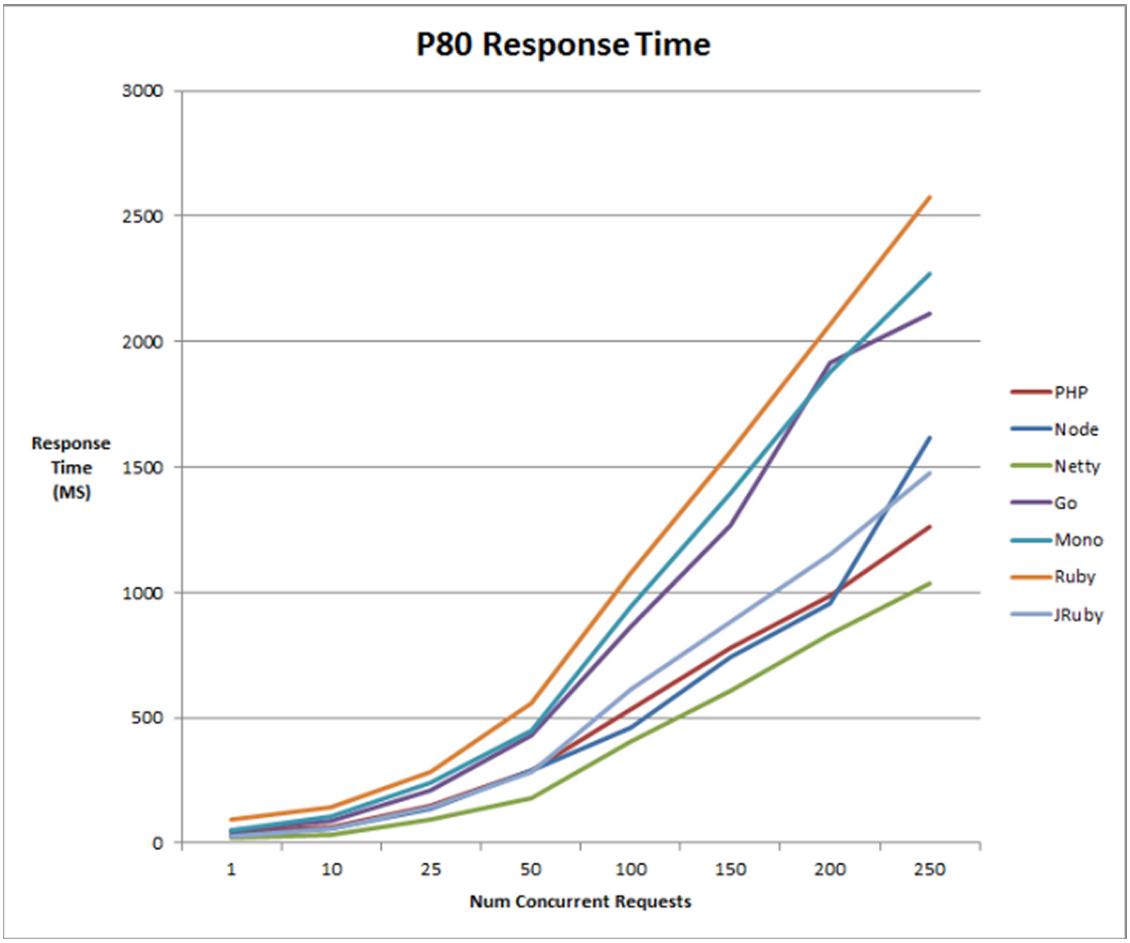
Among the hundreds of technical choices made available, the choice of a web framework is probably one of the most important — behind that of a client-side development framework (such as Phaser).

This article details[79] the various stress tests performed on seven different middleware sources.

[77] http://ithare.com/64-network-dos-and-donts-for-multi-player-game-developers-part-viib-security-concluded/
[78] http://www.sans.org/
[79] https://juiceboxgaming.wordpress.com/2012/11/20/benchmarking-web-frameworks-for-games/




**Mean Response Time (Smaller is Better)**




**80th Percentile Response Time (Smaller is Better)**

As demonstrated, node.js, as a single-threaded process, has limitations; whereas PHP barely shows any "sweat"! Looking further into this stress-test "response time" were analyzed. The article concludes:

> So a few conclusions
>
> - There's a marked difference in RPS and response time for new async frameworks and sync frameworks – this becomes especially obvious as the number of concurrent requests exceeds the number of physical cores on the box. Async frameworks achieve a peak RPS at above concurrent workers
> - Workloads demonstrating a richer I/O workload (as a % of all time) get larger benefits from using an async framework. Workloads with a pure CPU workload should demonstrate near identical performance on both sync and async frameworks.
> - Go achieves very terrible scores on workload 1 and really excellent scores on workload 2 – I think this is pointing to particularly crappy JSON and ZLib implementations present in Go.

- Mono 3.0.1's implementation of .NET 4.5 is super disappointing and really not ready for primetime (async keyword isn't implemented yet).
- It's a giant mistake to assume that system utility libraries demonstrate good/best performance. The best JSON lib can be 10x faster than the worst.
- There appears to be an emerging trend of decoupling a web framework from its server container (e.g. having a local nginx proxy). This introduces some new complexity in managing both processes and making sure that the guest web framework keeps its service and workers in a healthy state. (See footnote 2)

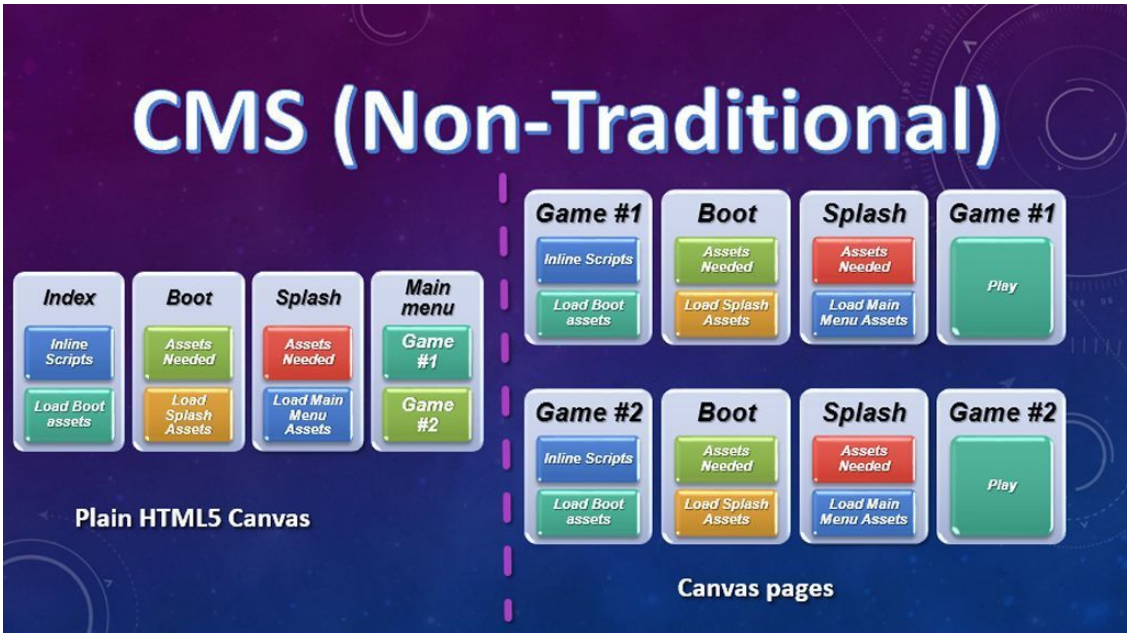
## CMS - Server-side Frameworks

Content Management Systems (CMS) is the topic of this section. We have focused on Phaser as a single web page, but what if we created a game (Part III — Mozart Music Match) that was a "game-shell" and packaged multiple individual games inside it? How would we manage handing-off the gamer into a subordinate-games and then return them back to the game-shell canvas? If you are familiar with Bulletin Board Systems (BBS) "Door Games", then you quickly see my direction. For those who are not familiar, please indulge my following explanation.

Nearly every book on Phaser demonstrates how to create a "single page / web application" (SPWA) (I call it "the Phaser Traditional Method") canvas game using a linear sequence:

```
IndexPage > Boot/Preload/Splash > MainMenu > Play > Win/Lose > Again?
```

What if we broke that sequence into a "tree hierarchy"? Meaning that the Main Menu is an altogether separate canvas page (tree trunk) than its individual game components (tree branches)? We would need a Content Management System (CMS) for this type of game structure. Building our game in this manner also permits us to easily convert it into "Android Intents" and further extend it into multi-player mode.




**CMS Game**

## Index Page (Non-Traditional Method)

I prefer CodeIgniter as my CMS; others might like(??) to use WordPress. Consider that the Phaser canvas is a single web page, and the CMS is the over-arching umbrella that permits access. You can review samples on how I use CodeIgniter here[80] or on my mobile game site.

- Blood Pit[81]
- Rogue Prince[82]
- Ruins of Able-Wyvern[83]
- Rulers of Renown[84]
- Soldiers Test[85]

Notice that the "indexes" are ".php" middleware pages, and are dynamically executed from the server-side — ***we are sooooo close to a multi-player construction. Can ya smell it??*** The advantage provided by CMS is a common head and footer template per dynamically generated body content.

[80] https://renown-quests.com/
[81] https://www.adventurers-of-renown.com/quests/arbp.php
[82] https://www.adventurers-of-renown.com/quests/arrp.php
[83] https://www.adventurers-of-renown.com/quests/arra.php
[84] https://www.adventurers-of-renown.com/quests/rrte.php
[85] https://www.adventurers-of-renown.com/quests/arst.php

***Exercise:*** Study the links above and discover how I can direct players to specifically tailored pages for the devices and tailor access game versions.

***NOTE:*** Refer to your ***Bonus Content:*** *`/back-endServices/samplePHPIndex.php`*[86]

## High Scores Services

You have several options when storing game information as a ***"non-invasive cross-cut concern";***[87] you may storage game scores and progress locally on the gamer's device or remotely on your database server/service. Below is some generic client-side script you will need for you "High Score" Menu.

```
1     //create a data structure to hold high scores
2     var highScores = [];
3     //create a data structure to track ranking
4     var hsPosition =[10,9,8,7,6,5,4,3,2,1];
5     //create a data structure to remote/local storage
6     var dbURL;
7
8     //create a high score function
9     BasicGame.HighScores = function(){};
10    BasicGame.HighScores.prototype = {
11    preload: function(){},
12    create: function(){
13        //link to your storage
14        //dbURL = new ("<where is your database?>");
15
16        for(var i = 1; 1< 11; i++){
17        hsPosition = game.add.text(
18                    x, y + (i + 50),
19                    i+" . "+ hsPosition[i],
20                    {fonSize:"40ps"}.anchor.serTo(1,0);
21        }
22
23        //dbURL api to pull remote/local data records
```

[86] https://makingbrowsergames.com/design/bonusDownLoads/back-endServices.zip
[87] https://software.intel.com/en-us/html5/hub/blogs/oop-is-not-your-hammer/

```
24      }
25      //CRUD
26      updateHSPosition: function(){
27          for(var i = 1; 1< 11; i++){
28              hsPosition[i].text = hsPosition[i];
29          }
30      }
31
32      }
```

***Exercise:*** Study the High Score PHP page in your ***Bonus Content: /back-endServices/samplePHPIndex.php***[88]. Determine what information you might collect on your game.
***Exercise:*** Where would you permanently store gamers' score? on the server in a database? or local on their device using storage such as SQLite3 [89], ForeRunnerDB[90] or perhaps PouchDB[91]?

***NOTE:*** Refer to your Bonus Content: `/back-endServices/HighScores.php`

***Hint:*** CodeIgniter support all the databases above.

## Membership Login

This is a tricky topic. Some distribution channels forbid this, others allow it ***only IF*** you use their software development kits (SDK).

***Exercise:***If you were to provide privileged access, where would you store the information? locally on their device or in a server-side storage system?
***Exercise:***What security would you provide for privileged user passwords?

[88] https://makingbrowsergames.com/design/bonusDownLoads/back-endServices.zip
[89] https://www.sqlite.org/version3.html
[90] https://github.com/Irrelon/ForerunnerDB
[91] https://github.com/pouchdb/pouchdb

Collecting credit card information is restricted by merchant account policies and qualifications. Collecting biological information is restricted by US Federal Laws. You will need a competent lawyer to guide you in these matters.

## Production release version.

My desire is to have my games load as quickly as possible. I covet those game producers' abilities to load and launch their games "in a blink of the eye."; don't you??!! How do they achieve such lighting fast game downloads? We've covered some of those topics in Chapter 3 and 4. This becomes the focus of our efforts in this section.

We plan to reduce our web server's workload across the web for our "production released" index file. Rather than loading each individual JavaScript script tags listed in the head, I will load the minimum scripts possible and apply a little-known feature in Phaser — `game.add.script`. So, my final result becomes just the phaser.js and a single game logic script file (called as you please, "game.js"? "app.js"? Just keep those hackers guessing).

Now I hear you saying, "Why are we putting all those scripts into one file? Won't a monolithic file take the same amount of time to download as several smaller files?" Wow! you are learning a trick or two, but hear me out. Don't worry, we're ***not going to put everything in one monolithic file!*** I plan to "bootstrap my game's loading" using a daisy-chain/tree hierarchy discussed earlier in the CMS section.

***Hint:***Remember Phaser paints the game onto an HTML5 canvas. We're using Phaser as a "thick client". The JavaScript files are easy to capture and read unless some security precautions are followed, and even then a determined "hacker/cracker" will reverse engineer your code. Much of your game logic resides on your server when using a CMS framework.

# CodeIgniter & Phaser Integrated CMS

**PHP development frameworks**

***NOTE:*** More detailed comparison information found here[92]

CodeIgniter[a] is a powerful open-source PHP framework with a very small footprint, built for developers who need a simple and elegant toolkit to create full-featured web

[92] https://www.valuecoders.com/blog/technology-and-apps/top-popular-php-frameworks-web-dev/

applications.

[a] https://github.com/bcit-ci/CodeIgniter

We are using a recent CodeIgniter stable release v3.1.2 (28 Oct 2016), since CodeIgniter 2.x version tree was discontinued 31 October 2015. CodeIgniter project is supported by Yeoman which facilitates rapid project creation or you could take the more traditional approach[93]. If this is your first exposure to CodeIgniter, you might like to review "the broad concepts behind CodeIgniter"[94] or create a "practice project"[95].

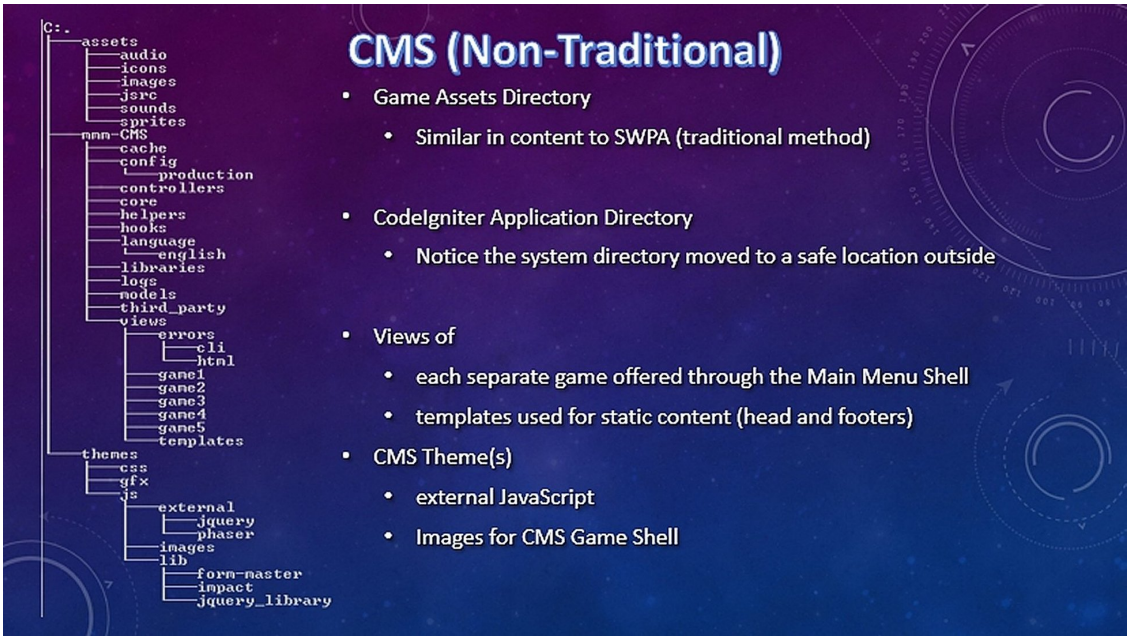
We will scaffold our CodeIgniter (CI) as a simple download of version 3.1.2. If you prefer, you could use Yeoman to build this CI project for you. I recommend either:

- RobonKey[96] offers several favors of CMS (***Express for NodeJS***[97], ***WordPress***, Drupal, ***CodeIgniter***, ***Laravel*** sub-generators).
- "CI3 Fire Starter"[98] — a CodeIgniter3 skeleton application that includes jQuery and Twitter Bootstrap. It is intended to be lightweight, minimalistic and not get in our way. Also for your convenience, it was created for developers new to the CodeIgniter (CI) framework who want a simple, easy platform for learning the CI framework. My mind "gets stoked" when I see excellent work as his.

***Exercise:*** Review both recommendations above. List several ways you might use the features presented in your game development.

***NOTE:*** If you are a WordPress developer or fanatic user, RobonKey would be your "Yeoman". Everything we are studying about CodeIgniter in this chapter could be tweaked to WordPress Posts or Pages. Express JS Framework is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It is the de facto standard server framework for Node.js.[99]

[93] http://www.codeigniter.com/user_guide/installation/index.html
[94] http://www.codeigniter.com/user_guide/overview/index.html
[95] http://www.codeigniter.com/user_guide/tutorial/index.html
[96] https://www.npmjs.com/package/generator-robonkey
[97] http://expressjs.com/
[98] https://github.com/JasonBaier/ci3-fire-starter
[99] https://en.wikipedia.org/wiki/Express.js




**CMS File Structure**

***Exercise:*** Review the CMS in your ***Bonus Content/MMM-js-CI3.1.2-v0001***[100] directory. I have changed the "application" directory name to "mmm"; all CodeIgniter (CI)applications have … well … an application directory. Changing its name is a simple security step. All the Content Management System is in the application directory.

Next, we need a way of running our CI CMS application. We can't simply open the index.html file in a web browser because our CI CMS create pages dynamically with the help of PHP. There are two files that launch our CodeIgniter CMS: mmm.php and index.php. Both of these files are exactly alike. Web servers always search and serve-up an index.* file by default. I use the mmm.php for internal CMS launches to distinguish between externally and internally CMS activation; this is logged for my security analysis.

Our migration plan is to let CodeIgniter handle all the navigation to and between (aka routing) for the various Phaser-content pages while letting Phaser concentrate its efforts on the current single-canvas display as a "thick client" and use the Phaser internal function `game.add.script`. Why? Because the content delivered to the gamer does NOT reveal any specific state changes; those state changes remain on the server and invisible inside of CodeIgniter. Furthermore, we can dynamically generate pages and select which Phaser modules to send; and, the delivered JavaScript files through `game.add.script` are dynamically generated only for that current gaming session's page. In short, a drastic increase in asset protection and overall game security.

[100] https://makingbrowsergames.com/design/bonusDownLoads/MMM-js-CI3.1.2-v0001.zip

```
    script(
        key,    //Unique name for this script file.
        url,    //file URL. If undefined or null
                // the url will be set to <key>.js,
                // i.e. if key was "alien" then
                // the URL will be "alien.js".
        callback,   //called after the script is loaded
        callbackContext // loader context
    )
```

***Warning:*** Using `game.add.script` means that we are taking the browser's load management into our own hands. Remember what Ilya Grigorik stated in chapter 4?

> "Unfortunately, these optimizations do not apply to resources that are scheduled via JavaScript; the preload scanner cannot speculatively execute scripts. As a result, moving resource scheduling logic into scripts may offer the benefit of more granular control to the application, but in doing so, it will hide the resource from the preload scanner, a trade-off that warrants close examination."

## CodeIgniter Prep Step-by-Step

***Step 1)*** Review your newly loaded CodeIgniter Directory; my application sub-directory is labeled "mmm-CMS". I have moved the CodeIgniter Systems directory to a "safe directory" outside of my root web server. Why?

- I have multiple games that use this CMS concept, and all of which use a common system.
- It is typically "beyond the reach" of a website hacker. If they were accessed, *** IT'S TIME TO FIND A BETTER Internet Service Provider (ISP)[101]!!***

***Step 2)*** Dive into the (root CI directory)/index.php. Review my "index.php" file. There are a few entries to observe, and the comments will help you to create your index.php.

- Lines 40 to 91 separate your development from your production environment automatically.

[101] http://gose-internet-services.net/

- Lines 92 to 102 must be changed to reflect your web-server path to the CodeIgniter Systems directory. I have kept the standard `$system_path = 'system';` and you see I have "commented out" mine.
- Line 103 to 119 must be changed to reflect your web-server path to the CodeIgniter application directory. Remember, I renamed my "application" directory to "mmm-CMS".
- Lines 120 to 134 if you would like to move your "view" content outside the web server, you would modify that path here.
- Lines 135 to 164 concerns "routing". I left this section alone and unchanged. However, this provides an additional and powerful security feature you might consider. I recommend going to another of my heroes and reading his tutorials on "CodeIgniter CMS"[102]. More tutorial references are listed in the Appendix.
- Lines 164 to 185 concludes the recommended configuration changes.

***Step 3)*** Dive into the (application directory)/config/ and review my "Config" file. CodeIgniter is very flexible, and with the flexibility comes responsible configuration settings. We have two mandatory files to configure — config.php and database.php and two very handy files to consider — autoload.php and constants.php. Let's start with the config.php:

***Step 3A)*** Config.php

- Lines 1 to 14 is my own concoction for Affiliate sales. I have provided my own code. If you are interested in becoming an Affiliated Sales agent, please read the Affiliate Guide in your Bonus Content or click this link[103], more information about this ***FREE*** program is here[104].
- Lines 15 to 40 tells CI where your project is located on your development server. You can see my project directory is active and my "live production" path was commented out.
- Lines 40 to 52 have a number of articles how to make the "index" file invisible. My development server uses the mmm.php as my internal access (mentioned above).
- Lines 53 to 69 are set to the default settings. You might research the other settings and determine if they are appropriate for your project.
- Lines 70 to 81 allow you to set your default page extensions. The default is " (nothing); I prefer ".html". You could tailor this to .jsp, .xhtm, .shtml depending on how you choose to deploy your general released game.
- Lines 82 * 93 are initial language settings.
- Lines 94 to 106 are character set settings.

[102] http://avenir.ro/en/
[103] http://www.stephen-gose.com/online-docs/AffiliateGuide/index.html
[104] http://www.stephen-gose.com/products/affiliates-program/

- Lines 107 to 117 allows for CI extensions (aka hooks)
- Lines 118 to 131 set apart your controllers that override core class.
- Lines 132 to 153 active Composer — a Dependency Manager for PHP[105].
- Lines 154 to 175 are provided for input security and the permitted characters in a URL.
- Lines 176 to 206 are the default settings for query strings.
- Lines 208 to 230 are your debugging logs. The default is "0"; you should turn this "on" while developing your CMS, and set back to "0" in the production environment.
- Lines 231 to 278 where would you like to find the logs? What log extensions would you prefer? Who has access to those logs? and date format for log files.
- Lines 279 to 289 sets your error on your "views" directory. The default is " (nothing).
- Lines 290 to 318 sets your cache path and query strings.
- ***Lines 320 to 331 is critical!*** This is your encryption key settings.
- Lines 332 to 411 sets your sessions and cookies information.
- ***Lines 446 to 460 set your cross-site scripting and global XSS filters. These are critical as you deploy your game across various distribution channels.***
- Lines 461 to 482 enable output compression. The default is off. Turning it "on" puts additional work-load on your server.
- Lines 483 to 495 set your time references.
- Lines 510 to 527 are a concern if you are using a Content Delivery Network (CDN such as cloudflare).

***Step 3B)*** database.php
The default setting is for 'mysqli'. I prefer to use 'pdo', and my reference notes are included in the file. More details are provided in the User Guide[106].

An often overlooked feature of CI is the database forge[107] that will help you manage your database and migrations. The Database Utility Class[108] contains methods that help maintain and backup your databases automatically.

***Step 3 Optional)*** autoload.php and constants.php. I have provided the default versions of my project.

## Game Shell (click dummy)

At this point in our game development, we are taking our ideas and affixing them to tangible media from all those grocery lists we have created. I would like to demonstrate how easy it is to build our CMS with Phaser game pages.

[105] https://getcomposer.org/doc/00-intro.md
[106] https://www.codeigniter.com/user_guide/database/configuration.html
[107] https://www.codeigniter.com/user_guide/database/forge.html
[108] https://www.codeigniter.com/user_guide/database/utilities.html

All of our pages have 3 standard components: head, body and Copyright footer information. Review my `MMM-js-CI3.1.2-v0001/mmm-CMS/views` directory. This holds everything "seen" by the gamer. Then review my `MMM-js-CI3.1.2-v0001/mmm-CMS/controller` directory, the controllers listen for input from the gamer(s), dissects it, pulls information from the database, if required, and merrily returns the resulting view back to the gamer(s). Yes, this is a standard `MVC` (Model, View, controller) client-server arrangement. You will find relief that CI also support jQuery/AJAX as the default. Here's is the code (jQuery Form Plugin)[109] I use in CI for that.

I take all the HTML head information and create a single template file called "head.php" and save it to my `MMM-js-CI3.1.2-v0001/mmm-CMS/views/templates` directory. I do the same for the Copyright footer; calling it "footer.php".

***Exercise:*** Review the content of my `MMM-js-CI3.1.2-v0001/mmm-CMS/views/templates` directory.

What remains for me to create in a "click dummy" game shell is the body of the "welcome page" — this combines the Phaser modules of Boot, Preload, Splash were avoided. CI did all this work for us and presented the gamer a simple "welcome/introduction page". CI dynamically generated:

- anchors and links in the head,
- page titles,
- JavaScript for this single page,
- the body,
- updated internal PHP variables,
- game state status,
- the footer, and generated
- live statistics

***Exercise:*** Study my online "click dummy" game shell here[110].

PHP anchor references in the index.php such as `<?php echo site_url('welcome/lobby')?>` permit a dynamically created `href`. If I choose to move directories, server names, or even ISPs, these references will always create the correct URL. This powerful feature

[109] http://jquery.malsup.com/form/
[110] https://www.renown-games.com/MozartMusicMatch/mmm.php

allows you to create generic games and sell your source code on marketplaces such as itch.io[111] or ***CodeCanyon***[112]. Clicking the anchor takes the gamer to the "Game Lobby".

***Exercise:*** Study my controller file located in `MMM-js-CI3.1.2-v0001/mmm-CMS/controllers/` are the `Welcome.php` and the `Area.php` files.

CI creates the default index page inside the `welcome` controller. Each HTML page is a function. The function builds or redirects (routes) gamers to their destinations. The gamer arrives in the "Game Lobby" where they are offered several decisions. Gamers may select how they want the game displayed by choosing a device and then choosing which music game to play. Many of the input controls are disabled to simplify the `area` controller.

The "Game" Lobby function inside the `welcome` controller redirects the gamer to a selected game. The information collected is sorted by the `area` controller and builds that page for the gamer. The page sent is a Phaser SWPA — the traditional method found on phaser.io[113] and many other tutorials. In my example, I am sending either the gamer to `index-mobile-mmm.html` or `index-mobile-mmmSWPA.html` (pages that we have constructed earlier in the workbook).

[111] https://itch.io/
[112] https://codecanyon.net/become-an-author?ref=PBMCube
[113] http://phaser.io/

## Summary

This chapter had numerous reference to external resources and examples. Here's what you accomplished:

- Learned about minimization and obfuscation methods.
- Correctly use inline frames (iframes) when deploying finalized game releases.
- Discover 64 "do" and "do not do"
- Found 1,000 of free security references, certifications, and a college degree in Cyber Security.
- Analyzed, compared and found shocking information on various middleware capabilities.
- Discovered most Phaser deployments are single-page games.
- Created a new method to deploy Phaser games through a Content Management System.
- Studied various server back-end services and middleware.
- Analyzed a CMS index.php.
- Learned the flexibility provided by using CMS hierarchy and "Non-Traditional" index page.
- Reviewed to potential local storage methods for JavaScript.
- Became aware of legal requirements for privileged user access.

## Chapter Footnotes

1. Rohler, N. (n.d.). The Magical Tag: An Introduction[114]. Retrieved November 2016.
2. Benchmarking Web Frameworks for Games.[115] (2012). Retrieved November 01, 2016.

[114] http://www.dwuser.com/education/content/the-magical-iframe-tag-an-introduction/
[115] https://juiceboxgaming.wordpress.com/2012/11/20/benchmarking-web-frameworks-for-games/

# Appendix: "How to Start a WebSocket"

***Excerpt from Chapter 6***
**"Phaser Making Massive Multi-Player Online Games"**

With the solid foundation from Parts I & II, your research, and answers to the various exercises, we are — at last! — ready to begin our source coding sessions. We will start on the client-side and migrate across the MMoG Architecture we discussed in Part II. Our goal, by the end of this chapter, is to have:

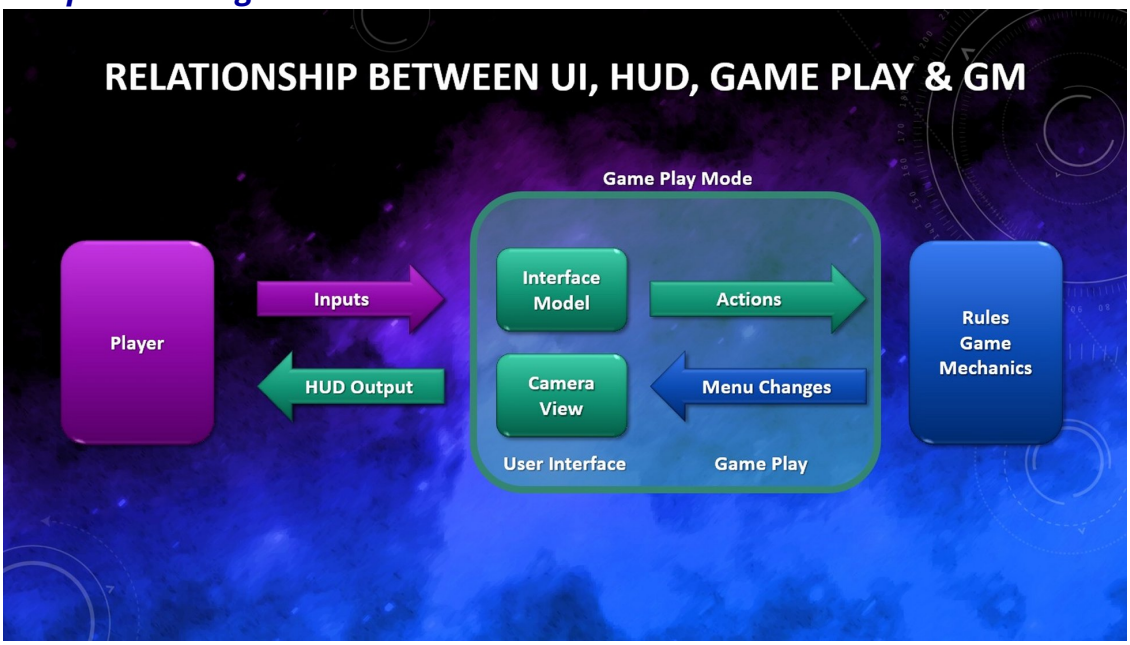
- a functional WebSocket conduit (aka "telecommunications channel") starting from client-side proxy to server API with data formatted as either an RPC or MOM. It becomes our MMoG prototype to expand into more specific game messages in later chapters.
- a guiding regimen showing us where to place our source code across our MMoG systems.

To use WebSockets, you need a browser, a web server, and a socket app server that all support ***the same WebSocket protocol.*** (You learned that in Part I, and ***all the reasons*** to avoid using Socket.IO and its incompatibilities). Your best friend will become ***"Can I Use" website***[116]; it provides current information about the WebSocket technology, browsers to avoid, known issues, and resources. You were given two test sites in Part II; it's time to create your own testing server — this is not the same as `Web Server`. You will need a traditional web server (such as Apache or IIS) on your local development workstation to serve your game's ***dynamic content.*** (You'll recall from Part I that Node.js and Express only deliver static content? Right?) You have a "Project Game Starter Kit" in your Bonus Content you need to copy this into your workstations web server root directory and make sure it's running in the background of your development workstation. You'll need to make sure not to close the command prompt window!

***Exercise:*** Download the "Project-GameStarterKit": ***https:***

[116] https://caniuse.com/#search=websocket

***//makingbrowsergames.com/mmog/bonusContent/WebSockets.zip*** or any client SDK from ***https://kaazing.com/download/***




**New MMoG Information Flow - Single Player to MMoG Server**

The Client-side has several important responsibilities while processing the game logic so that it works well with the remote server (or local proxy server) and provides a quality game-play experience:

- ***Rendering*** — the Client-side is responsible for rendering the game on the player's display, and may be responsible for non-important physics simulations, such as cloth simulations or particle effect — such special effects (sfx) should ***NOT*** come from a remote server, ***RIGHT?*** If a client-side animation doesn't have gameplay relevance but is merely "frosting or eye-candy" to make the game "taste" better, it shouldn't be executed on the `MMoG App server` at all. ***RIGHT?*** Why should we add network lag and delay to sfx that do ***NOT*** impact other players?
- ***Sound*** — The Client-side plays all audio sound effects and/or music tracks. It is ***preposterous***[117] to think that the server would stream audio playback.
- ***Input*** — The Client-side collects a player's input, packages it, and then sends that information to the server in a negotiated format. We'll study RPC or MOM **JSON declarative** formats in the upcoming Server and MMoG App chapters.
- ***User Preferences*** — Many games store user preferences on the local machine where the game runs. However, that also depends on the type of data. Obviously, a gamer's subscription account balance would be safely stored in the back-end business logic and storage. Safe data might be the gamer's native language, volume controls, and any keyboard adjustments.

[117] https://www.dictionary.com/browse/preposterous

- ***Prediction*** — The Client-side could predict what might happen to game objects in the short term while it awaits for the centralized server to synchronize and send "confirmations". I'll show you how this works later in this chapter using a client-side proxy server.
- ***Interpolation*** — As part of the Client-side prediction process, the Client can calculate where a game object needs to be, and where it thought it should be, and where the authoritative server confirms where it is, and shall go. Interpolation is important in "Real-time" games — when multiple players are simultaneously modifying the game state. Many MMoG tutorials don't tell you about "Separation of Concerns". The ***"interpolation and extrapolation"***[118] calculations are perfect candidates for a client-side proxy server.

***Chapter 5 Exercise revisited:*** You'll recall that Colt McAnlis from Google Chrome Games (https://www.youtube.com/watch?v=Prkyd5n0P7k video 51:40; see 21:23 to 21:27) stated (quote), " . . . we're **going to ignore client-side prediction, is hands down, is the perceptive correct solution from all the game players.** If you're writing games, make sure you do this way."

The whole idea is amazing, that JavaScript can do all this locally. Then add onto this idea that JavaScript is based on a **single-threaded environment** that increases the "awesome factor". Web Workers remove (partially) this single-threaded limitation as you've read in the previous MMoG Application Architecture chapter.

## Testing Your Browser

As of 20190327, we know that over 95% of all browsers ***already*** support WebSockets ***natively!*** We should check if our browsers support WebSockets before we go any further writing our MMoG.

**JS check for client-side WebSocket.**

```
1    // Make sure the browser supports WebSocket
2    if (!window.WebSocket) {
3        displayMessage("Your browser does not support WebSockets");
4        return;
5    }
```

[118] https://gamedev.stackexchange.com/questions/118006/when-should-i-extrapolate-and-when-should-i-interpolate

In your "Project Game Starter Kit", you downloaded earlier, search for the directory "WebSocket" and inside the "WebSocketTest.html" file and open it with your favorite browser. Click the link — "Run WebSocket" link. Testing completed. Did it worked? Right? If not, then research https://canluse.com[119] at the bottom of the page are tabs leading to "Notes", "Known Issues", and current "Issues". You also might like to import their usage data and discover how much of the world already supports WebSockets ***natively in browsers.*** You also noticed that we didn't need a webserver to run this simple test; but, we will need a "full-stack" web server installed on our development workstation eventually.

**NOTE:** What you just did above is how your gamers will also enter your MMoG. ***Study these files; it's just pure JavaScript*** and ***only your browser.*** There's only the native WebSocket protocol; Socket.IO is ***nowhere to be seen!*** All the current MMoG tutorials spend half their time just getting Socket.IO up and running on your local workstation? ***Why? Yeap,***[120] we've already covered all those reasons in Part I, and now I'm demonstrating what I've claimed from those previous chapters.

Open the `WebSocketTest.html` and study what we just did. You may also test your active browser against either ***my demo socket server***[121] or against ***websocket.org.***[122] Review chapter 5 for other details:

> **Test sites:**
>
> ***https://mmog.pbmcube.net/index.php***
> ***https://websocket.org/demos.html***
> ***Client Interface: http://www.abrandao.com/lab/websocket/client.html***

## WebSocket Protocol Handshake

To switch into an "upgraded WebSocket connection", the gamer's browser should send a ***"WebSocket handshake request",*** — which we did. The game testing servers will return a ***"WebSocket handshake response",*** — which it did; nothing new thus far . . . same old, same old that we studied in Parts I & II.

[119] https://caniuse.com/#search=Websockets
[120] https://www.urbandictionary.com/define.php?term=yeap
[121] https://mmog.pbmcube.net/index.php
[122] https://websocket.org/echo.html

## Deeper Dive: WebSocket API


**Upgrade to WebSockets**

**The WebSocket API - Read more ...**

| Event handler | *Event handler types*[123] |
|---|---|
| **onopen** | open |
| **onmessage** | message |
| **onerror** | error |
| **onclose** | close |

***Both server- and client-side WebSocket objects*** support the following API.

- ***on('open', function(event))*** fires when the socket connection is established. Event has no attributes. Ensure your data is sent only when a connection exists; we should wrap our `send()` method with onopen event.
- ***on('message', function(event))*** fires when the socket receives a message. The "Event" has one attribute, data, which is either a String (for text frames) or a Buffer (for binary frames). The default data format is "blob" which is particularly useful when sending and receiving files.
- ***on('error', function(event))*** fires when there is a protocol error due to bad data sent by the other peer. This event is purely informational, you ***do not need*** to implement error recovery because WebSockets rides on top of TCP/IP. ***"So how do errors still creep in?"*** Superior question; the answer is found in ***RFC 6455 Section 5.1 Overview***[124] which you've already read (?) from the Networking

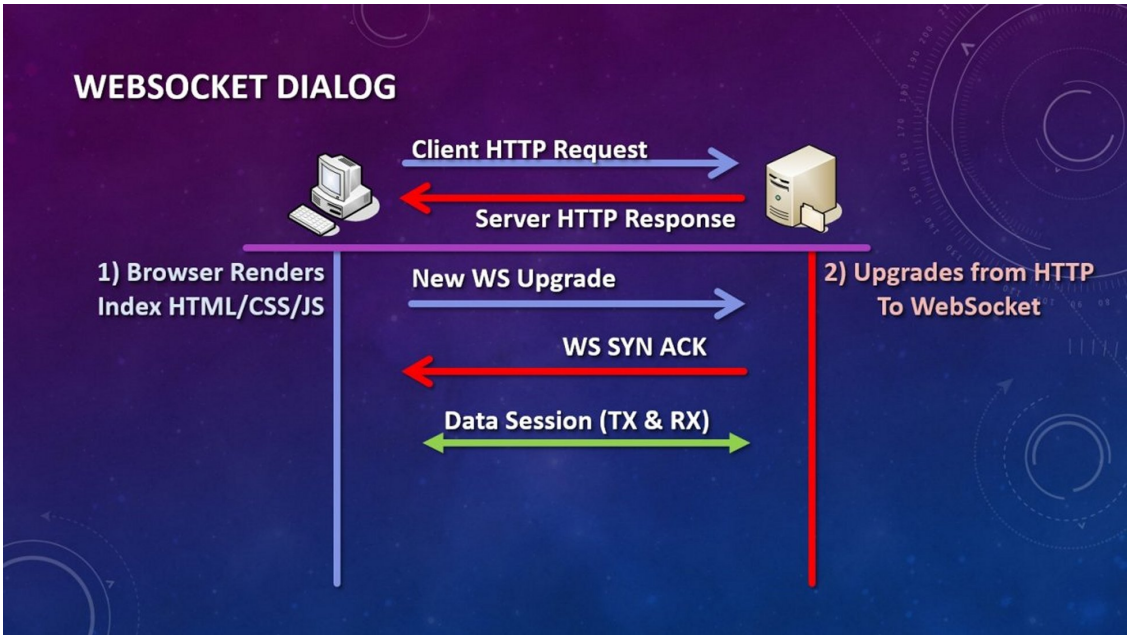
[124] https://tools.ietf.org/html/rfc6455#page-27

Chapter exercises.

- ***on('close', function(event))*** fires when either the client or the server closes the connection conduit. This event has two optional attributes — code and reason — that expose the status code and message sent by the peer that closed the connection. It also has three properties you can use for error handling and recovery: `wasClean`, `code`, and `error`.
- ***send(message)*** accepts either a String or a Buffer and sends a text or binary messages over the connection conduit to the other remote peer.
- ***ping(message, function())*** sends a "ping" frame with an optional message and fires the callback when a matching answer (aka "pong") is received. This is typically used to dynamically adjust the MMoG for network latency issues.
- ***close(code, reason)*** closes the connection conduit, sending the given status code and reason text, both of which are optional.
- ***version*** is a string containing the version of the WebSocket protocol the connection is using.
- ***protocol*** is a string (which may be empty) identifying the sub-protocol the socket is using. We discuss various "sub-protocols" in the upcoming Server chapters.

The illustration above demonstrates the responsibilities of the client and the remote MMoG server on each end of the WebSocket communications conduit.

```
This line in our testing script
var ws = new WebSocket('ws://mmog.pbmcube.net:30113/', 'mmog');
```




**WebSockets Upgrade Dialog**

| Property | Description |
|---|---|
| bufferedAmount | Returns the total number of bytes that were queued when the send() method was called `binaryType` Returns the binary data format we received when the onmessage event was raised |
| protocol | Returns the protocol used by the server |
| readyState | Reports the state of the connection and can take one of the following self-explanatory values:<br>- WebSocket.OPEN<br>- WebSocket.CLOSED<br>- WebSocket.CONNECTING<br>- WebSocket.CLOSING |
| url | Returns the URL of the WebSocket |

Gamer’s client requests above are similar to HTTP requests, each line ends with `\r\n` and there must be an extra blank line at the end. This is the standard format. The handshake resembles an HTTP request so that the game server can handle standard HTTP connections (port 80) as well as the new WebSocket connections across the same initial port 80 (or 443 for https). Once the new WebSocket request creates that connection, communications between the client and game server switches into a ***bidirectional binary protocol that does not conform*** to the former HTTP protocol. After a WebSocket connects, the gaming client and server can send WebSocket ***binary data or text frames*** in full-duplex mode — in either direction at the same time! This is a point-to-point connection as if the client and server were peers. The data is minimally framed (for us by the WebSocket Protocol) with just two bytes. (Review Chapter 3 Network Basics) There’s no need to write code to package these frames

as some tutorials lead us to believe. However, it is a “stream” of data; if your MMoG needs to understand discreet components, you might like to use a message format inside these WebSocket frames.

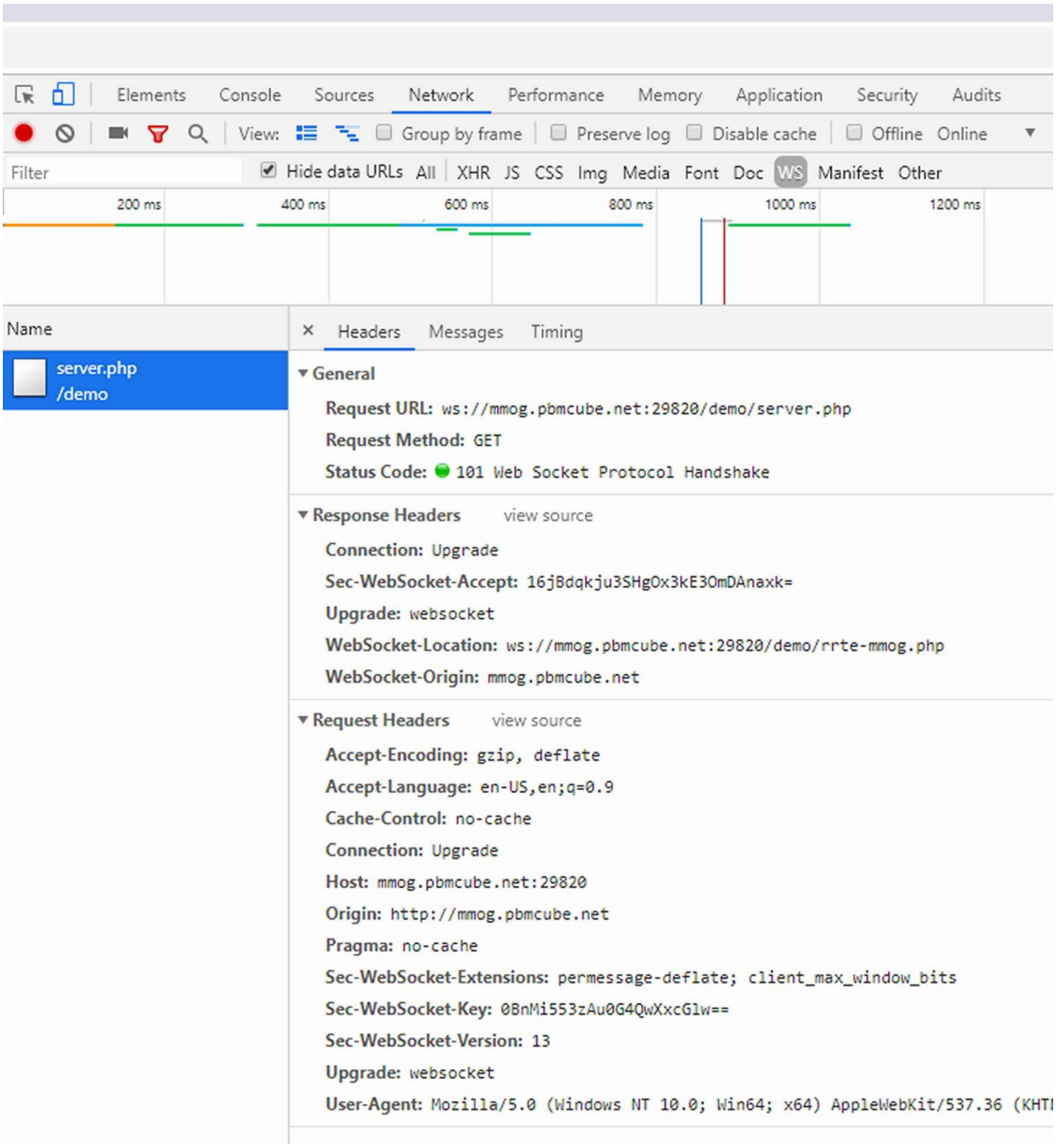
The WebSocket API allows for multiple types of data (`UTF8` text, `binary`, and `blob data`), and unlike the browser, these messages on the server are stored using different properties based upon those data types. In the case of text frames, each frame starts with a **`0x00 byte`**, ends with a **`0xFF byte`**, and contains **`UTF-8`** data in between. (Review Chapter 3 Network Basics page 62) This binary data stream is difficult for humans to read since it is a ***bidirectional binary protocol.*** WebSocket text frames use ***a terminator,***[125] while binary frames use a length prefix.

**A typical Server’s response header looks like this:**

```
1    HTTP/1.1 101 Switching Protocols
2    Upgrade: websocket
3    Connection: Upgrade
4    Sec-WebSocket-Accept: cGxheWVyMzIxfGdhbWUxMjN8QUNL
5    Sec-WebSocket-Protocol: mmog, soap, wamp, sip, amqp, mqtt, xmpp
```

In addition to Upgrade headers, the client sends a ***`Sec-WebSocket-Key`*** header containing **base64-encoded**[126] random bytes, and the server replies with a hash of the key in the ***`Sec-WebSocket-Accept`*** header. This is intended to prevent a caching proxy from re-sending any previous WebSocket dialog, (see footnote 1[127]) and ***does not provide any authentication, privacy or integrity.*** The hashing function appends the fixed string `258EAFA5-E914-47DA-95CA-C5AB0DC85B11` (a GUID) to the value from `Sec-WebSocket-Key` header ***(which is not decoded from base64),*** applies the SHA-1 hashing function, and encodes the result using base64. (see footnote 2[128])

[125] https://www.ietf.org/proceedings/76/slides/hybi-1.pdf
[126] https://www.base64decode.net/
[127] https://trac.ietf.org/trac/hybi/wiki/FAQ
[128] https://tools.ietf.org/html/rfc6455#section-1.3




**Observe WS Header Exchange in Developer Console**

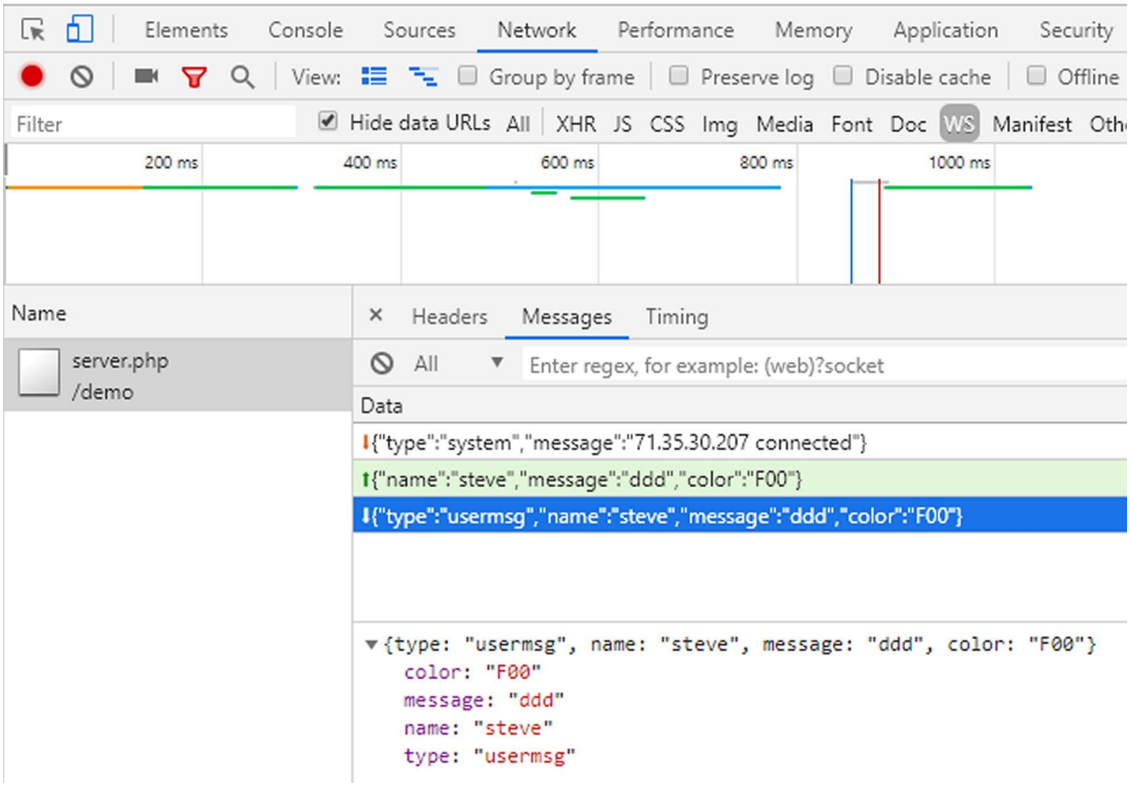
**Launched the following**

```
    Request Headers . . .
    GET ws://mmog.pbmcube.net:30113/demo/server.php HTTP/1.1
    Host: mmog.pbmcube.net:30113
    Connection: Upgrade
    Upgrade: websocket
    Origin: http://mmog.pbmcube.net
    Sec-WebSocket-Version: 13
    Sec-WebSocket-Key: zfG4XfYTSKp6NdIpCaYe+w==
```

```
9       Sec-WebSocket-Protocol: mmog, soap, wamp, sip, amqp, mqtt, xmpp
10      Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
```

***Exercise:*** WebSocket monitoring with ***Wire-Shark in 3 min***[129]

Observe these WebSocket frames with ***Wire-shark.***[130] (Refer to the Network Appendix on where to get **Wire-shark.**) Registered IANA Sec-WebSocket-Protocol is **referenced here.**[131] The server can’t send more than one ***`Sec-Websocket-Protocol`*** header. If the server doesn’t want to use any sub-protocol, it shouldn’t send any Sec-WebSocket-Protocol header. Sending a blank header is incorrect. The client may close the connection if it doesn’t get the sub-protocol it wants.




**Observe WS Message Content Exchange in Developer Console**

[129] https://youtu.be/5R1ugZ_jLDI
[130] https://wiki.wireshark.org/WebSocket
[131] http://www.iana.org/assignments/websocket/websocket.xml

**. . . and receives the following**

```
1    Response Headers . . .
2    HTTP/1.1 101 Web Socket Protocol Handshake
3    Upgrade: websocket
4    Connection: Upgrade
5    WebSocket-Origin: mmog.pbmcube.net
6    WebSocket-Location: ws://mmog.pbmcube.net:30113/demo/rrte-mmog.php
7    Sec-WebSocket-Accept:EZtDpyc+cXnNbqdzXW1QCmdm//Y=
```

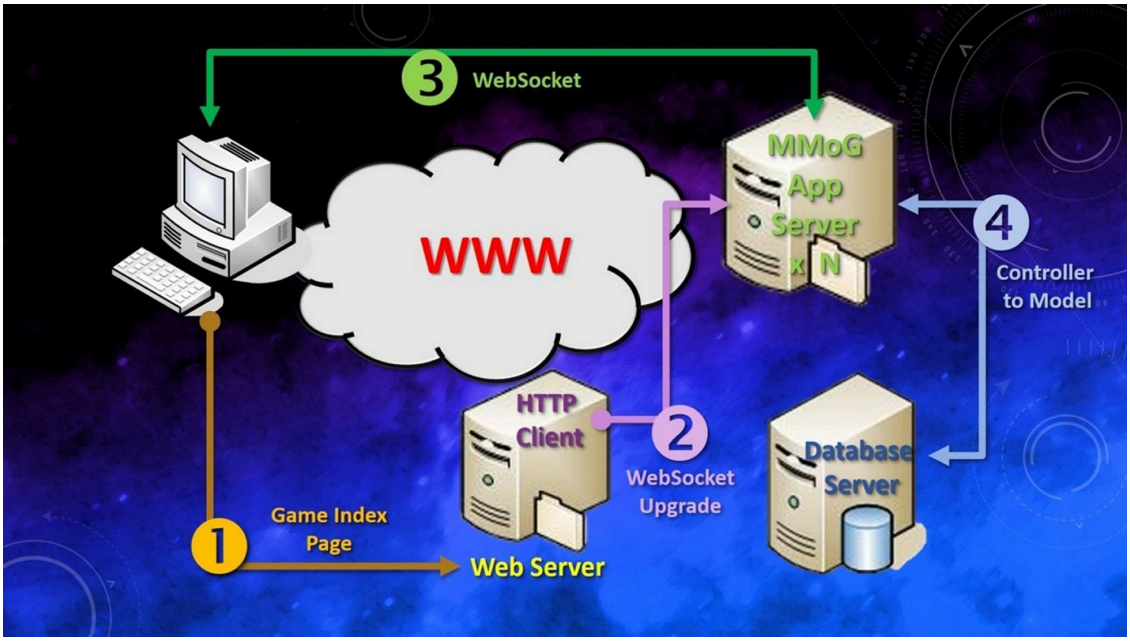
***Exercise:*** Go back and re-do the browser test to either `mmog.pbmcube.net` or `websocket.org/demos.html`. This time open your Develop Console and observe your information headers and responses.
***Exercise:*** Need help using the Developer Tools? With the Chrome Dev Tools,[132] you can now see the WebSocket traffic coming to and going from your browser ***without using tools like Wireshark.***[133] Review this article https://developers.google.com/web/tools/chrome-devtools/

***Note:*** Although the Web Sockets protocol is ready to support a diverse set of clients beyond the gaming community, ***it cannot deliver raw binary data to JavaScript, because JavaScript does not support a `byte` type.*** Therefore, binary data is ignored if the client is JavaScript — but raw binary data could be delivered to other clients that support it; ***the WebSocket protocol is platform-independent.***

## Sample Source Code: Client-side WebSocket

The client-side is easy; it's merely a JavaScript inside of an `index` page. Once we know our MMoG App server is running — the remote end of our WebSocket conduit — your gamers can connect to it through your "Web Server" and "subscribe to gaming messages" pushed from your MMoG App Server. The gamers will establish WebSocket connections through a process known as "WebSocket handshake". This "handshake" starts when the gamer visits a "Game's `index page`" from the Web Server to enter a specific game. The Web Server will then include an `Upgrade header` in this request that tells the "MMoG App Server" a gamer wants to establish a WebSocket connection. All of this was covered in Part I; this was simply a review.

[132] https://developers.google.com/web/tools/chrome-devtools/
[133] http://www.wireshark.org/




**To Summarize into our 4-Step development process (Chapter 5)**

1. ***Step #1:*** Inside the game's index page, create a JavaScript WebSocket object — it is the client-side conduit toward the MMoG server's URL.
2. ***Step #2:*** Generate code for the following events — ***"onopen", "onclose",*** and ***"on-error"*** as WebSocket event handlers. Construct the ***"onmessage"*** event handler (the client-side workhorse and potentially a proxy-server) to handle, deserialize, and read all incoming Game Turn Responses (GTR) coming from the `MMoG App server`.
3. ***Step #3:*** Sending Game Turn Orders (GTO) "messages" to the `MMoG App server` using the WebSocket `send()` method. This is the topic in later chapters and Part IV when we create Game Turn Orders and server's Game Turn Results.
4. ***Step #4:*** Closing the client-side WebSocket conduit connection to the `MMoG App server` using the WebSocket `close()` method and transition into a new game phase.

Using ***HiveMQ,*** you don't need a dedicated web-server in front of an MMoG App Server to forward the WebSocket connection. ***Read more about HiveMQ ...***[134]

[134] https://www.hivemq.com/blog/mqtt-over-websockets-with-hivemq/

> ***WARNING:***[a] User agents must not convey any failure information to scripts in a way that would allow a script to distinguish the following situations:
>
> - A server whose hostname could not be resolved.
> - A server to which packets could not successfully be routed.
> - A server that refused the connection on the specified port.
> - A server that failed to correctly perform a TLS handshake (e.g., the server certificate can’t be verified).
> - A server that did not complete the opening handshake (e.g. because it was not a WebSocket server).
> - A WebSocket server that sent a correct opening handshake, but that specified options that caused the client to drop the connection (e.g. the server specified a sub-protocol that the client did not offer).
> - A WebSocket server that abruptly closed the connection after successfully completing the opening handshake.
>
> [a] https://html.spec.whatwg.org/multipage/web-sockets.html#feedback-from-the-protocol

In all of these cases, ***the WebSocket connection close code***[135] would be 1006, as required by the WebSocket Protocol specification. **WSP**[136]

> Allowing a script to distinguish these cases would allow a script to probe the user’s local network in preparation for an attack. In particular, this means the `code 1015` is **not used** by the user agent (unless the server erroneously uses it in its close frame, of course).

## Step #1: Game `index` page

[135] https://html.spec.whatwg.org/multipage/infrastructure.html#concept-websocket-close-code
[136] https://html.spec.whatwg.org/multipage/references.html#refsWSP

***Typical `index.html` with WebSocket***

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <link rel="dns-prefetch" href="http://mmog.pbmcube.net/">
  <title>WebSocket MMoG Client Example</title>

  <!-- Typically Phaser Library is here with your gaming scripts -->

</head>
<body>
  <h1>MMoG Client-side via WebSockets</h1>
  <p>Open the JavaScript console to see what's up!
  Notice we're not using Node.js, Socket.IO; we're just
  using your vanilla browser for now.</p>
  <form>
    <label for="message">Send a Game Turn</label>
    <input id="message" name="message" type="text">
    <button id="send" name="send">Submit</button>
  </form>

<!-- The secret sauce is in this external file
    or simply insert the script directly. -->
  <script src="ws_client.js"></script>
</body>
</html>
```

This `index.html` file (above) is placed on the `WWW` server; it is sent to the gamer's browser. The client browser runs the JavaScript that will create the WebSocket conduit. This file is available in the "Project Game Starter Kit" at

- Download the ***WebSockets.zip***[137] ***OR***
- Download the ***MMoG server-side example***[138].

## Step #2: Generate Event handlers

Let's turn our attention to that JavaScript just above the closing `body` tag. Initializing a WebSocket is just as simple as the following lines of JQuery code. Create an external

[137] https://makingbrowsergames.com/mmog/bonusContent/WebSockets.zip
[138] https://makingbrowsergames.com/mmog/bonusContent/mmog-server-side.zip

file entitled `ws_client.js` or insert the following script directly into the `index.html` — search in your Bonus Content for the directory "/WebSockets/" and inside that for the ***`ws_client.js` (click to review)***[139] or ***test online here:***[140]

**Testing a WebSocket script (ws_client.js)**

```
(function () {
    // ---------------------------------
    // You can check the number of bytes that have been
    // queued but not yet transmitted to the server,
    // this is very useful if the client application transports
    // large amounts of data to the server.
    // ---------------------------------
    var MAX_BUFFER = 8192;

    // ---------------------------------
    // Warning! NEVER USE WEBSOCKET 12345!
    // port 12345 is a known virus;
    // Read in Phaser Multi-Player Gaming Systems
    // Chapter 6.2 WebSocket Protocol HandShake
    // ---------------------------------
    // new WebSocket (url, optional sub-protocol)
    var ws = new WebSocket('ws://localhost:12345', 'echo-protocol');

    //============================
    // Step 2: 4 WebSocket Protocol Msg
    //============================
    // WebSocket Events:
    // you simply add callback functions to the WebSocket object
    // or you can use the addEventListener() DOM method on your
    // WebSocket objects. It's important to implement these events
    //  before attempting to send any messages.
    // ---------------------------------
    // Step #4: Close client-side conduit
    // FIN finished and cleaning up connection.
    ws.onclose = function (event) {
        console.log('Connection closed.');
    }

    // Example of "close" addEventListner
    // Set handler for when the socket connection is closed
```

[139] https://makingbrowsergames.com/mmog/bonusContent/WebSockets/ws_client.js
[140] https://makingbrowsergames.com/mmog/bonusContent/WebSockets/WebSocketTest.html

```
36          // websocket.addEventListener("close", function() { ...}
37
38          // Oops! Discovered an error the TCP/IP
39          //  overlooked RTT or RTO problems
40          // See RFC6455 Section 5.1 for more potential errors.
41          ws.onerror = function (event) {
42              console.log('An error occurred. Sorry for that.');
43          }
44          // --------------------------------
45
46          // Conversations / dialogs messages to and from
47          //  We'll cover this topic in later Chapters & Part IV.
48          // alternate method:
49          //      ws.onmessage = incomingMsg(event);
50          // Example of "message" addEventListner
51          // Set handler for when the socket receives a message
52          // websocket.addEventListener("message", function() { ...}
53          ws.onmessage = function (event) {
54              console.log('Response from server: ' + event.data);
55          }
56
57          ws.onopen = function (event) {
58              console.log('Connection opened.');
59              // let's try to send every second?
60              setInterval( function() {
61                  if (ws.bufferedAmount < MAX_BUFFER) {
62                      ws.send(checkTheStatusAndSendData());
63                  }
64              }, 1000);
65          }
66
67          // Add our customized handling
68          // The readyState attribute may have one of the four values:
69          // - CONNECTING (The connection is not yet fully open)
70          // - OPEN (The connection is open and ready to communicate)
71          // - CLOSING (The connection is in the process of closing)
72          // - CLOSED (the connection closed or couldn't be opened)
73          // Knowing the current state can be very useful in
74          // troubleshooting your application.
75          ws.sendMessage = function (message) {
76              if(ws.readyState === ws.OPEN){
77                  this.send(message);
78                  console.log('Message sent: ' + message);
```

```
79              } else {
80                  console.log('Connection failed.')
81              }
82          }
83
84          // Attach created game turn orders into a message frame
85          document.getElementById('send').addEventListener('click',
86          function (event) {
87              event.preventDefault();
88              var message = document.getElementById('message').value;
89              ws.sendMessage(message);
90          });
91
92      })();
```

Review the sample code above. It contains the WebSocket API in a few events — `onclose`, `onerror`, `onmessage`, and `onopen`. ***That's it!*** Sending messages to the server uses the method `send()` and disconnecting from the server uses the method `close()`. ***Why does everyone try to make this so difficult? AND! it's supported in 95% of all browsers NATIVELY!*** We managed to flip from HTTP connection into a WebSocket connection using a simple `upgrade` command! Why was it so simple? Because both are "peer protocols" to each other riding above the TCP/IP transportation layer. Everything is explained in the ***official W3C WebSocket API.***[141] In Part I & II, we investigated that many other programming languages use this exact same WebSocket API specifications; without a doubt, WebSocket is ***ubiquitous.***[142]

In the code examples above, the ***`bufferedAmount`*** attribute is used to ensure that updates are sent at a rate of one update every 1000 ms (1 fps) if the network can handle that rate, or at whatever rate the network can handle, if that is too fast. The ***`bufferedAmount`*** attribute could saturate a network by sending data at a higher rate than the network can handle; this requires careful monitoring by using WebSocket "Pings" and "Pong" commands. We'll study and apply this information in upcoming server chapters.

***Study `ws_client.js`! There's no Socket.io, nor Node.js, nor Express.js!*** We are using the internal native WebSocket protocol ***already available inside our browsers. How easy was that? No extra code, no formatting the data stream, no encoding nor decoding, AND protocols are software language-agnostic!*** Everything was already performed ***for us by our browsers.*** Furthermore, ***Phaser — either version v2.x.x. or v3.16+ — had nothing to do with any of this;*** the WebSocket conduit simply opened. Phaser

[141] https://www.w3.org/TR/websockets/
[142] https://www.merriam-webster.com/dictionary/ubiquitous

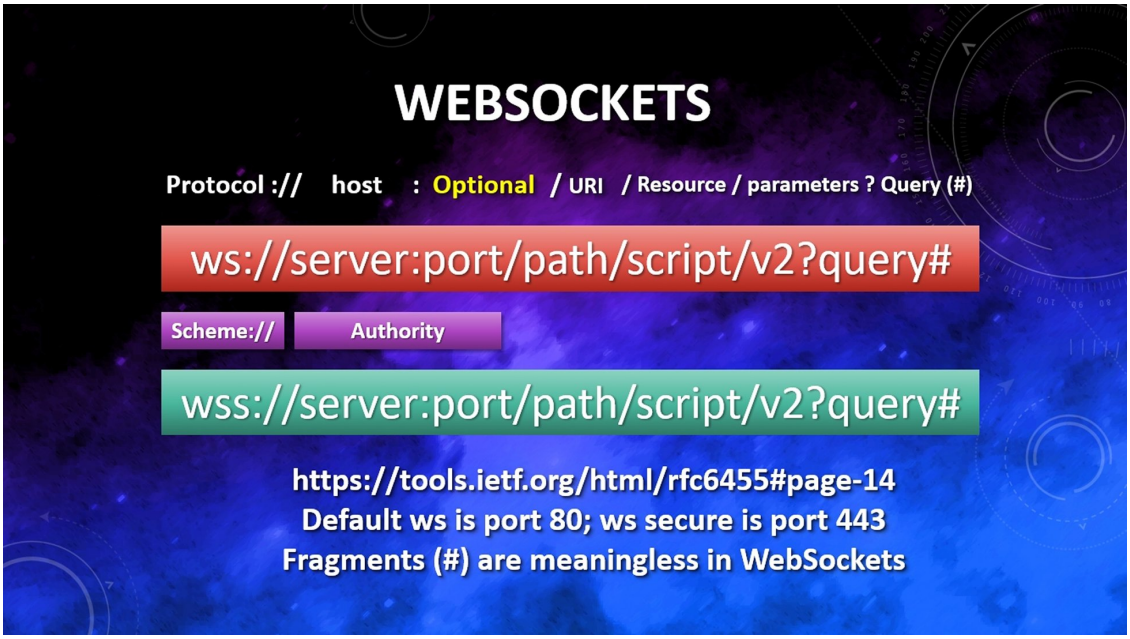
is inside an HTML5 canvas tag (i.e., a Layer 7 application ) — `Applications` receiving services from lower communication layers services (i.e., our WebSockets, and TCP/IP bindings) and let's keep it that way. This is the primary idea behind the concept of "Separation of Concerns" covered in both the ***Phaser Game Design Workbook,***[143] and ***Phaser Game Prototyping workbook, new 6th Edition***[144]

***WARNING: Do not use port 12345 as shown above***[145]; it appears as a Trojan Viruses: cron / crontab, Fat Bitch trojan, GabanBus, icmp_pipe.c, Mypic , NetBus, NetBus Toy, NetBus worm, Pie Bill Gates, Whack Job, X-bill. Click the link above to see all known port viruses. ***Port 54321***[146] in the example above is used by Back Orifice 2000 and School Bus viruses. ***Research your port number selections wisely!***

If your ISP permits WebSockets or Node.js, they will assign you a software port number to use. ***I'm fortunate to use an ISP***[147] ***that supports both Node.js and WebSockets.*** Yes, Node.js can have a role in MMoG development if applied correctly as we learned in Part II. We covered all those "ins and outs" of Node.js and all the "whens and how-tos" previously.

***Exercise:*** Research the other ***WebSocket API attributes and commands***[148] on Mozilla.
***Exercise:*** Study ***MDN: Writing WebSocket client applications***[149] for more details.

[143] https://leanpub.com/phaserjsgamedesignworkbook
[144] https://leanpub.com/LoRD
[145] https://isc.sans.edu/port.html?port=12345
[146] https://www.speedguide.net/port.php?port=54321
[147] http://gose-internet-services.net/the-hepsia-hosting-cp/advanced-features/
[148] https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
[149] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications




**Server-side target**

***Exercise:*** Research the following and select, not less than 5 port nominations for your MMoG server. This will be useful if you are using Dockers, Kubernetes, or Cloud-based virtual environments.

- Trojan Port Table[150]
- TCP & UDP ports for your online games[151]
- Special Application Port List[152]
- NMap Services[153]
- Internet Storm Center[154]
- Service Name and Transport Protocol Port Number Registry[155]

[150] http://www.chebucto.ns.ca/~rakerman/trojan-port-table.html
[151] http://www.gameconfig.co.uk/
[152] http://www.practicallynetworked.com/sharing/app_port_list.htm
[153] https://svn.nmap.org/nmap/nmap-services
[154] https://isc.sans.edu//
[155] http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

# Appendix: Project Mgmt Methods

Everyone has an opinion on "how to create game-design documentation" and "how to manage game product development". Formal Project Management suggests the ***"Systems development life cycle"***[156] for software game design — writing "big designs up-front" (BDUF) whose goal is to answer all questions about the game development process in a tome. The main problem with this formal process is the misconception of "perfect knowledge".

For small development studios such as ours, this formal process is expensive in time, man-hours, risk, and money. In reality, one can never truly know everything about a game initially. In the early stages of the game development process, one has the greatest range of "uncertainty".

One gains more knowledge as development proceeds through the Game Mechanics rules, data, and logic. The mistake of large development teams using formal project management is realized only too late in the postmortem follow-ups. The postmortem reveals:

- Creating knowledge has a high cost in man-hours,
- Knowledge is the greatest asset produced aside from the final game product itself. Is it a marvel that many small indie developers are moving to this new business model of ... "Actual Start The Damn Game"!?

Flash was acclaimed as a "rapid application development" (RAD) environment. In general, the RAD approach to software development puts less emphasis on planning tasks and more emphasis on the development and encoding process. The RAD approach emphasizes the necessity of adjusting requirements in reaction to current knowledge gained as the project progresses. This relies on the use of "throw-away prototypes" in addition to (or even sometimes in place of) design documentation. If RAD is adopted, and under closer scrutiny, indie flash game developers simply used ***"Cowboy Coding"*** — immediately producing source code. At some point, they would begin testing (often near the end of the project's production cycle), and the unavoidable bugs would then be fixed before distribution to "app stores". In its essence, it's programming **without a design;** it could be labeled "design on-the-fly" or "wouldn't that be cool, if ..." All too often "scope creeps" and excellent game ideas die

[156] http://en.wikipedia.org/wiki/Systems_development_life_cycle

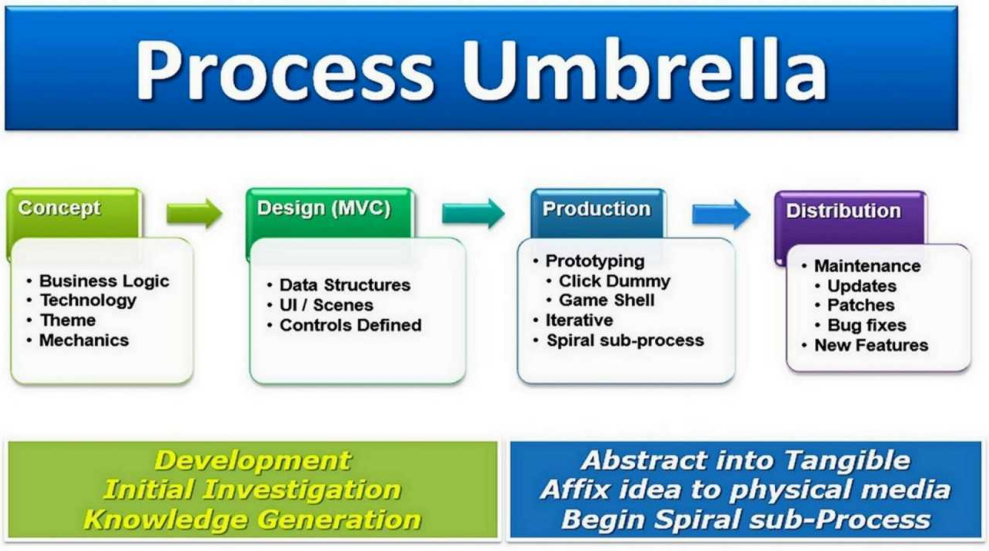
in mid-development. Yet, indie developers have created revenues from simply having a "game concept".

***Game Project Management (GPM)*** is becoming easier for the range of game studios — smaller game indie to larger development teams. It seems the new trend is to integrate GPM into their game design editors.[157] At Stephen Gose Game Studios (SGGS), we use a GPM method known as ***"Software Prototyping" with "Extreme Programming".*** On paper (aka "throw-away prototyping"), we identify the game's basic features and requirements in:

1. client and server technology,
2. business logic (i.e. Revenue generation),
3. environment themes, and
4. Gaming Mechanics — the mode, genre, rules, data, and logic.

We follow the "4-core umbrella" steps in our GPM:

1. Concept,
2. Design,
3. Production, and
4. Distribution.




**Part I Introduction to Game Design System™**

[157] See this article: ***"P4Connect Project Management Software into the Unity engine"***. P4Connect embeds the firm's P4D versioning engine into the Unity developer environment, allowing users to access "Perforce's features™" such as code and asset management, tracked change history, and automation" directly within Unity. Developers will not need a Unity Pro or Team Unity license, making P4Connect available to all indie developers that use Unity.

The first two steps are called the project's "Development Phase ", we are asking pre-production questions such as "what", "how", and "is it fun" questions. Some seasoned experts in the game industry would say we are following a "Waterfall method" or possibly "SCRUM" project management — if labels are important to you. We are cutting the fat off SCRUM and using a "leaner more agile" model for our small studio development. Since our team consists of one, two, or sometimes as many as three people, we already have tight integration within multiple disciplines.

Capturing the ideas and research are the major activities in the "Development Phase". We generate knowledge before we enter the "Production Phase". In this "development phase", we're starting with the game's metadata structures and building our data model's "ERDs" in StarUML. Next, we draft — on paper! — the game user interfaces (UI), menus, and navigation interactions. Afterwards, we move into a game prototype "click dummy" of the game shell in either the Flash IDE of Phaser Editor 2D. It worth mentioning at this stage, we are moving from paper into physical design calling this phase "pre-production".

Now the fun starts, source encoding! Iteration is paramount; development testing is usually done concurrently with, or at least in the same iteration as, code generation. As we write procedures, frames, and functions, we write "just barely good enough" documentation as comment annotations in the source code, so that we will understand what we're doing and its rational "why". These "notes" become the rudimentary outline for our "Making Browser Game" Series. Understand that we create games based on inspiration and that we may not return to a game idea for weeks, months or, — in some cases — years! We have adopted a philosophy that game ideas alone won't help us and won't get us to the market. So, we create a working **"game shell"** component with fully operational navigation among scenes for either ActionScript 2 and/or 3. ActionScript 2 is easily transpiled into JavaScript. We spiral through each section of code until it works. We are using Flash as an "Artwork GUI Builder"[158]. Creating this initial code, we often notice repetitive patterns or simpler ways to achieve the same results by generalizing, pushing, or pulling content code; at this point, we are re-factoring and "abstracting".

In 2010, we were considering the option of moving our ActionScript source code off the main-timeline into external source files. This was a significant rework and divergent work-flow with low priority. In 2014, MochiMedia died, and Flash is meeting its "end-of-life" DEC 2020. We began transpiling all our Flash game products (at that time we had 88 products of which 60% were still in the development pipeline). Today (20200907), we have 255 games in our development pipeline of which 105 are deployed as either Phaser v2.x.x and/or Phaser III.

[158] Refer to Project Management RAD.

# Prototyping


**Prototyping Project Management**
**Framework Type: Iterative**

## Basic Principles

1. Not a standalone, complete development methodology, but rather an

approach to handling selected portions of a larger, more traditional development methodology (i.e., Incremental, Spiral, or Rapid Application Development (RAD)).

2. Attempts to reduce inherent project risk by breaking a project into

smaller segments and providing more ease-of-change during the development process.

3. User is involved throughout the process, which increases the

likelihood of user acceptance of the final implementation.

4. Small-scale mock-ups of the system are developed following an

iterative modification process until the prototype evolves to meet the users' requirements.

5. While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototypes toward a working system.

6. A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem.

### Strengths:

1. "Addresses the inability of many users to specify their information needs, and the difficulty of systems analysts to understand the user's environment, by providing the user with a tentative system for experimental purposes at the earliest possible time." (Janson and Smith, 1985)

2. "Can be used to realistically model important aspects of a system during each phase of the traditional life cycle." (Huffaker, 1986)

3. Improves both user participation in system development and communication among project stakeholders.

4. Especially useful for resolving unclear objectives; developing and validating user requirements; experimenting with or comparing various design solutions; or investigating both performance and the human-computer interface.

5. A potential exists for exploiting knowledge gained during earlier

iteration as later iterations are developed.

6. Helps to easily identify confusing or difficult functions and missing functionality.

7. May generate specifications for a production application.

8. Encourages innovation and flexible designs.

9. Provides a quick implementation of an incomplete, but functional, application.

### Weaknesses:

1. The approval process and control are not strict.

2. Incomplete or inadequate problem analysis may occur whereby only the most obvious and superficial needs will be addressed, resulting in current inefficient practices being easily built into the new system.

3. Requirements may frequently change significantly.

4. The identification of non-functional elements is difficult to document.

5. Designers may prototype too quickly, without sufficient up-front user's "needs analysis", resulting in an inflexible design with a narrow focus that restricts future system potential.

6. Designers may neglect documentation, resulting in insufficient justification for the final product and inadequate records for the future.

7. Can lead to poorly designed systems. Unskilled designers may

substitute prototyping for sound design, which can lead to a “quick and dirty system” without global consideration of the integration of all other components. While initial software development is often built to be a “throwaway”, attempting to retroactively produce a solid system design can sometimes be problematic.

8. Can lead to false expectations, where the customer mistakenly

believes that the system is “finished” when in fact it is not; the system looks good and has adequate user interfaces, but is not truly functional.

9. Iterations add to project budgets and schedules, thus the added

costs must be weighed against the potential benefits. Very small projects may not be able to justify the added time and money, while only the high-risk portions of very large, complex projects may gain benefit from prototyping.

10. The prototype may not have sufficient checks and balances incorporated.

### Situations where *most* appropriate:

1. The project is for the “development phase” of an online system requiring extensive user dialogs, or for a less well-defined expert and decision support system.

2. The project is large with many users, interrelationships, and functions,

where project risk relating to requirements definition needs to be reduced.

3. Project objectives are unclear.

4. Pressure exists for an immediate implementation of something.

5. Functional requirements may change frequently and significantly.

6. The user is not fully knowledgeable.

7. Team members are experienced (particularly if the prototype is not

a throw-away).

8. The team composition is stable.

9. The Project Manager is experienced.

10. No need exists to absolutely minimize resource consumption.
11. No strict requirement exists for approvals at designated milestones.
12. Analysts/users appreciate the business problems involved, before they begin the project.
13. Innovative, flexible designs that will accommodate future changes are not critical.

### Situations where *least* appropriate:

1. Mainframe-based or transaction-oriented batch systems.

2. Web-enabled e-business system

3. The project team composition is unstable.

4. Future design scalability is critical.

5. Project objectives are very clear; project risk regarding

requirements definition is low.

## Incremental

**Framework Type: Combination Linear and Iterative**

### Basic Principles:

Various methods are acceptable for combining linear and iterative system development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process:

1. A series of mini-waterfalls are performed, where all phases of the

Waterfall development model are completed for a small part of the system, before proceeding to the next increment; OR

2. Overall requirements are defined before proceeding to evolutionary,

mini-Waterfall development of individual increments of the system, OR

3. The initial software concept, requirements analysis, and design of

architecture and system core are defined using the Waterfall approach, followed by iterative Prototyping, which culminates in installation of the final prototype (i.e., working system).

### Strengths:

1. Potential exists for exploiting knowledge gained in an early

increment as later increments are developed.

2. Moderate control is maintained over the life of the project through

the use of written documentation and the formal review and approval/signoff by the user and information technology management at designated major milestones.

3. Stakeholders can be given concrete evidence of project status

throughout the life cycle.

4. Helps to mitigate integration and architectural risks earlier

5. Allows delivery of a series of implementations that are gradually

more complete and can go into production more quickly as incremental releases.

6. The gradual implementation provides the ability to monitor the effects

of its incremental changes, isolate issues, and make adjustments before the organization is negatively impacted.

### Weaknesses:

1. When utilizing a series of mini-Waterfall for a small part of the

system before moving on to the next increment, there is usually a lack of overall consideration of the business problem and technical requirements for the overall system.

2. Since some modules will be completed much earlier than others,

well-defined interfaces are required.

3. Difficult problems tend to be pushed to the future to demonstrate

early success to product management.

### Situations where *most* appropriate:

1. Large projects where requirements are not well understood or are

changing due to external changes, changing expectations, budget changes or rapidly changing technology.

2. Web Information Systems (WIS) and event-driven systems.

3. Leading-edge applications.

### Situations where *least* appropriate:

1. Very small projects of very short duration.

2. Integration and architectural risks are very low.

3. Highly interactive applications where the data for the project

already exists (completely or in part), and the project largely comprises analysis or reporting of the data.

## Spiral

**Framework Type: Combination Linear and Iterative**

### Basic Principles:

1. Focus is on risk assessment and on minimizing project risk by

breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of the project continues throughout the life cycle.

2. “Each cycle involves a progression through the same sequence of

steps, for each portion of the product, and each of its levels of elaboration, from an overall “concept-of-operation” document down to the coding of each individual program.” (Boehm, 1986)

3. Each trip around the spiral traverses four basic quadrants: (1)

determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration. (Boehm, 1986 and 1988)

4. Begin each cycle with an identification of stakeholders and their

win conditions, and end each cycle with review and commitment. (Boehm, 2000)

### Strengths:

1. Enhances risk avoidance.

2. Useful in helping to select the best methodology to follow for

development of a given software iteration, based on project risk.

3. Can incorporate Waterfall, Prototyping, and Incremental

methodologies as special cases in the framework, and provide guidance as to which combination of these models best fits a given software iteration, based on the type of project risk. For example, a project with low risk of not meeting user requirements, but high risk of missing budget or schedule targets would essentially follow a linear Waterfall approach for a given software iteration. Conversely, if the risk factors were reversed, the Spiral methodology could yield an iterative Prototyping approach.

### Weaknesses:

1. Challenging to determine the exact composition of development

methodologies to use for each iteration around the Spiral.

2. Highly customized to each project, and therefore is quite complex,

limiting reusability.

3. A skilled and experienced project manager is required to determine

how to apply it to any given project.

4. There are no established controls for moving from one cycle to

another cycle. Without controls, each cycle may generate more work for the next cycle.

5. There are no firm deadlines. Cycles continue with no clear

termination condition, so there is an inherent risk of not meeting budget or schedule.

6. The possibility exists that a project ends up implemented

following a Waterfall framework

### Situations where *most* appropriate:

1. Real-time or safety-critical systems.
2. Risk avoidance is a high priority.
3. Minimizing resource consumption is not an absolute priority.
4. The Project Manager is highly skilled and experienced.
5. A requirement exists for strong approval and documentation control.
6. The Project might benefit from a mix of other development methodologies.
7. A high degree of accuracy is essential.
8. Implementation has priority over functionality, which can be added

in later versions.

### Situations where *least* appropriate:

1. Risk avoidance is a low priority.
2. A high degree of accuracy is not essential.
3. Functionality has priority over implementation.
4. Minimizing resource consumption is an absolute priority.

## Rapid Application Development (RAD)

***Framework Type: Iterative***

## Basic Principles:**

1. The key objective is for the fast development and delivery of a high-quality system at a relatively low investment cost.

2. Attempts to reduce inherent project risk by breaking a project into

smaller segments and providing more ease-of-change during the development process.

3. Aims to produce high-quality systems quickly, primarily through the

use of iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer-Aided Software Engineering (CASE) tools, Database Management System (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.

4. The primary emphasis is on fulfilling the business need, while

technological or engineering excellence is of lesser importance.

5. Project control involves prioritizing development and defining

delivery deadlines or “time frames”. If the project starts to slip, emphasis is on reducing requirements to fit the time box, not in increasing the deadline.

6. Generally includes Joint Application Development (JAD), where users

are intensely involved in system design, either through consensus building in structured workshops, or through electronically facilitated interaction.

7. Active user involvement is imperative.

8. Iteratively produces production software, as opposed to a

throwaway prototype.

9. Produces documentation necessary to facilitate future development and maintenance.

10. Standard systems analysis and design techniques can be fitted into this framework.

### Strengths:

1. The operational version of an application is available much earlier than with Waterfall, Incremental, or Spiral frameworks.

2. Because RAD produces systems more quickly and to a business focus, this approach tends to produce systems at a lower cost.

3. Engenders a greater level of commitment from stakeholders, both business and technical, than Waterfall, incremental, or Spiral frameworks. Users are seen as gaining more of a sense of ownership of a system, while developers are seen as gaining more satisfaction from producing successful systems quickly.

4. Concentrates on essential system elements from a client's viewpoint.

5. Provides the ability to rapidly change system design as demanded by users.

6. Produces a tighter fit between user requirements and system specifications.

7. Generally produces dramatic savings in time, money, and man-hours.

## Weaknesses:

1. More speed and lower costs may lead to lower overall system quality.

2. The danger of the developed system misalignment with the business drivers due to missing information.

3. The project may end up with more requirements than

needed (aka “gold-plating”).

4. Potential for feature creep where more and more features are added

to the system course of development.

5. The potential for inconsistent designs within and across systems.

6. Potential for violation of programming standards related to

inconsistent naming conventions and inconsistent documentation.

7. The difficulty with module reuse for future systems.

8. The potential for the designed system to lack scalability.

9. Potential for lack of attention to later system administration needs

built into the system.

10. The high cost of commitment on the part of key user personnel.
11. Formal reviews and audits are more difficult to implement that for a complete system.
12. The tendency for difficult problems migrated to future iterations to demonstrate early success to senior management.
13. Since some modules will be completed much earlier than others, well-defined interfaces are required.

### Situations where most appropriate:

1. Project is of small-to-medium scale and short duration (no more than 6 man-years of development effort).

2. Project scope is focused, such that the business objectives are well defined and narrow.

3. The application is highly interactive, has a clearly defined user group, and is not computationally complex.

4. The functionality of the system is clearly visible at the user interface.

5. Users possess detailed knowledge of the application area.

6. Senior management commitment exists to ensure end-user involvement.

7. The requirements of the system are unknown or uncertain.

8. It is not possible to define requirements accurately ahead of time because the situation is new or the system being employed is highly innovative.

9. Team members are skilled both socially and in terms of business.

10. Team composition is stable; continuity of the core development team can be maintained.
11. Effective project control is definitely available.
12. Developers are skilled in the use of advanced tools.
13. Data for the project already exists (completely or in part), and the project largely comprises analysis or reporting of the data.
14. Technical architecture is clearly defined.
15. Key technical components are in place and tested.

16. Technical requirements (e.g., response times, throughput, database sizes, etc.) are reasonable and well within the capabilities of the technology being used. Targeted performance should be less than 70% of the published limit of the technology.
17. The development team is empowered to make design decisions on a day-to-day basis without the need for consultation with their superiors and decisions can be made by a small number of people who are available and preferably co-located or in communication.

### Situations where least appropriate:

1. Very large, infrastructure projects; particularly large, distributed information systems such as corporate-wide databases.

2. Real-time or safety-critical systems.

3. Computationally complex systems, where complex and voluminous data must be analyzed, designed, and created within the scope of the project.

4. The project's scope is broad and the business objectives are obscure.

5. Applications in which the functional requirements have to be fully specified before any programs are written.

6. Many people must be involved in the decisions on the project, and the decision-makers are not available on a timely basis or they are geographically dispersed.

7. The project team is large or there are multiple teams whose work needs to be coordinated.

8. When user resource and/or commitment is lacking.

9. There is no project champion at the required level to make

things happen.

10. Many new technologies are to be introduced within the scope of the the project or the technical architecture is unclear and much of the technology will be used for the first time within the project.
11. Technical requirements (e.g., response times, throughput, database sizes, etc.) are tight for the equipment that is to be used.

# Test-Driven Development

**Framework Type: Iterative**

## Basic Principles:

"Test-driven development" refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests), and design (in the form of refactoring).

It can be succinctly described by the following set of rules:

- write a "single" unit test describing an aspect of the program
- run the test, which should fail because the program lacks that feature
- write "just enough" code, the simplest possible, to make the test pass
- "refactor" the code until it conforms to the simplicity criteria
- repeat, "accumulating" unit tests over time

## Expected Benefits

Many teams report significant reductions in defect rates, at the cost of a moderate increase in initial development effort the same teams tend to report that these overheads are more than offset by a reduction in effort in projects' final phases although empirical research has so far failed to confirm this, veteran practitioners report that TDD leads to improved design qualities in the code, and more generally a higher degree of "internal" or technical quality, for instance improving the metrics of cohesion and coupling

## Common Pitfalls

Typical individual mistakes include:

- forgetting to run tests frequently
- writing too many tests at once
- writing tests that are too large or coarse-grained
- writing overly trivial tests, for instance omitting assertions
- writing tests for trivial code, for instance, “accessors”

## Typical team pitfalls include:

- partial adoption — only a few developers on the team use TDD
- poor maintenance of the test suite — most commonly leading to a test suite with a prohibitively long running time
- an abandoned test suite (i.e. seldom or never run) — sometimes as a result of poor maintenance, sometimes as a result of team turnover

## Signs of Use

- “Code coverage” is a common approach to evidencing the use of TDD; while high coverage does not guarantee appropriate use of TDD, coverage below 80% is likely to indicate deficiencies in a team’s mastery of TDD
- version control logs should show that test code is checked in each time product code is checked in, in roughly comparable amounts

## Skill Levels

### *Beginner*

- able to write a unit test before writing the corresponding code
- able to write code sufficient to make a failing test pass

### *Intermediate*

- practices “test-driven bug fixing”: when a defect is found, writes a test exposing the defect before a correction

- able to decompose a compound program feature into a sequence of several unit tests to be written
- knows and can name several tactics to guide the writing of tests (for instance "when testing a recursive algorithm, first write a test for the recursion terminating case")
- able to factor out reusable elements from existing unit tests, yielding situation-specific testing tools

### *Advanced*

- able to formulate a "road-map" of planned unit tests for macroscopic features (and to revise it as necessary)
- able to "test drive" a variety of design paradigms: object-oriented, functional, event-drive
- able to "test drive" a variety of technical domains: computation, user interfaces, persistent data access...

### Further Reading on Test Driven Development

- ***Test-Driven Development By Example***[159], by Kent Beck

## Game Project Management Foot Notes:

1. See Game Business Development Appendix & References
2. See Appendix for Game Design & References.
3. See article at Gamasutra — ***A Primer for the Design Process***[160].

[159] http://amzn.to/2AtVxgV
[160] http://www.gamasutra.com/view/feature/131558/a_primer_for_the_design_process_.php

# Appendix: Consolidated Phaser Examples

***Instructions:*** Open the ***"bareBonesIndex.html"***[161] and ***game.js***[162] in your favorite text editor. Use your browser's "developers Tools -> Console" to watch and monitor the live code in operations. Some examples are embed the JS code inside the index.html page instead of using externally loaded files.

## Phaser III (1st to 6th editions):

### Demonstrations:

***https://makingbrowsergames.com/p3gp-book/_p3-demos/***
***https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js***
***https://makingbrowsergames.com/p3gp-book/_p3-demos/game-noNameSpace.js***

### Searching for Game Mechanics and Mechanisms.

***https://makingbrowsergames.com/p3gp-book/_p3-demos/Ch5-game.js***

### Content Management System embedded in *HTML5 <canvas> tag.*

***https://makingbrowsergames.com/p3design/project-starterKit-demo/***

[161] https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html
[162] https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js

# Phaser III Examples

***Instructions:*** Search inside the file for “Example n.n” where “n” is the number of the example.

- ***Example: 1.1 Creating Namespace for games***[163]
- ***Sample: Bare Bones Index.html***[164]
- ***Sample: Bare Bones with no name-space***[165]
- ***Sample: Mobile Index.html***[166]

—

- ***Example: 2.1 Prototyping Graphics (Lines 876 to 889)***[167]
- ***Example: 2.2: Starting the Game.js (Preload Game Phase)***[168]

—

- ***Example: 3.1 Creating Game Phases (traditional method)***[169]
- ***Example: 3.1a Creating Game Phase Continued - Game.js (traditional method)***[170]
- ***Example: 3.2 Additional Phaser Properties JS (Lines 218 to 270)***[171]
- ***Example: 3.3 Additional Phaser Properties***[172]

—

[163] https://makingbrowsergames.com/p3gp-book/_p3-demos/_index.html
[164] https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html
[165] https://makingbrowsergames.com/p3gp-book/_p3-demos/game-noNameSpace.js
[166] https://makingbrowsergames.com/p3gp-book/_p3-demos/index-mobile.html
[167] https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js
[168] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson02.html
[169] https://makingbrowsergames.com/p3gp-book/_p3-demos/index3.html
[170] https://makingbrowsergames.com/p3gp-book/_p3-demos/index3a.html
[171] https://makingbrowsergames.com/p3gp-book/_p3-demos/game.js
[172] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson03.html

## Phaser III Game Prototyping Demonstrations

Open the the index.html in your favorite text editor.
Use your browser's "developers Tools -> Console" to watch and monitor the live code.
Reference the book explanations and find the embedded sourced code in the html as script tags.

***Table of Contents:***[173]



—

[173] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/index.html
[174] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson01.html
[175] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson02.html
[176] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson03.html
[177] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson04.html
[178] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson05.html
[179] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson06.html
[180] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson07.html
[181] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson08.html
[182] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson09.html
[183] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson10.html
[184] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11.html
[185] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11a.html
[186] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson11a.html
[187] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/lesson12.html

## Game Mechanics & Mechanisms identified

***Instructions:*** Search inside the file for "GM" where game mechanics and mechanisms are identified according to Schell's definitions.

- ***Example: 6: Identifying Schell's Game Mechanics***[188]

—

## WebSockets, Dynamic Menus, Combat, and FSM

Open the the index.html in your favorite text editor.
Use your browser's "developers Tools -> Console" to watch and monitor the live code.
Reference the book explanations and find the embedded sourced code in the html as script tags.

***Table of Contents***[189]

- ***Example: 7.1: Launching Web Sockets***[190] ( see Appendix for details)
- ***Example: 7.2: Dynamic Combat Menus***[191]
- ***Example: 7.3: Dynamic Combat Menus supporting function***[192]
- ***Sample: Projectile Templates Phaser v3.5+***[193]
- ***Example: 7.4: Combat Grid***[194] (checker-board using Phaser III grid feature)
- ***Example: 7.5: Grid-less Combat Encounter Demo***[195]
- ***Example: 7.6: Hexagonal Combat Grid***[196] (Lines 216 to 233; using Phaser III Shapes feature as hexagonal vertical-align)
- Sample: 7.7: Simple AI reactions (see AI chapter)
- Sample: Combat Finite State Machine (see AI chapter)

[188] https://makingbrowsergames.com/p3gp-book/_p3-demos/Ch5-game.js
[189] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/index.html
[190] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson01.html
[191] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson02.html
[192] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson03.html
[193] https://labs.phaser.io/view.html?src=src/games/top%20down%20shooter/topdown_combatMechanics.js&v=3.50.0-beta.5
[194] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson04.html
[195] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/
[196] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch7-examples/lesson06.html

- ***Sample: SCAVT game (play demo)***[197]: review lines ***292-318 (here)***[198]
- ***Example: Dynamic Story Narrative***[199] (from ***"Dating Veronica Darlene"***[200])
- ***Example: Combat Narrative (demo here)***[201] and review ***code here***[202] — journaled narrative for each combat turn using Single-Player optimized FSM menus.

—

Sample: 8.1: Prototyping a HUD (in book)
Sample: 8.2: Heads Up Display Plugin (in book)
Example: 8.3: HUD Menu Grouping (in book)

—

Sample: 9.1: Combat Pseudo Code (in book)
Sample: 9.2: Combat Pseudo Code (in book)
***Example: 9.3: Enemy mirrored movement***[203]
Sample: 9.4: Combat Pseudo Code (in book)
Sample: 9.5: New Combat States Module Added (in book)

[197] https://makingbrowsergames.com/p3gp-book/p3-sca/
[198] https://makingbrowsergames.com/p3gp-book/p3-sca/js/state/play1.js
[199] https://makingbrowsergames.com/starterkits/quiz/p3game3/
[200] http://leanpub.com/mbg-dating
[201] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/
[202] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/js/state/combat-SinglePlayer-OptimizedMenu.js
[203] https://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1

# Appendix: Game Automation Tools

We've generated a bunch of ***code snippets***[204] so far. This ***codebase***[205] won't stop with just the ***Phaser III JS Gaming Framework*** but you might eventually include Phaser v2.x.x and ***any other JavaScript Gaming Frameworks***[206] based upon your clients' requests. This process is a gentle ***introduction into "cloud-based"***[207] services and eventual ***MMoG development.***[208]

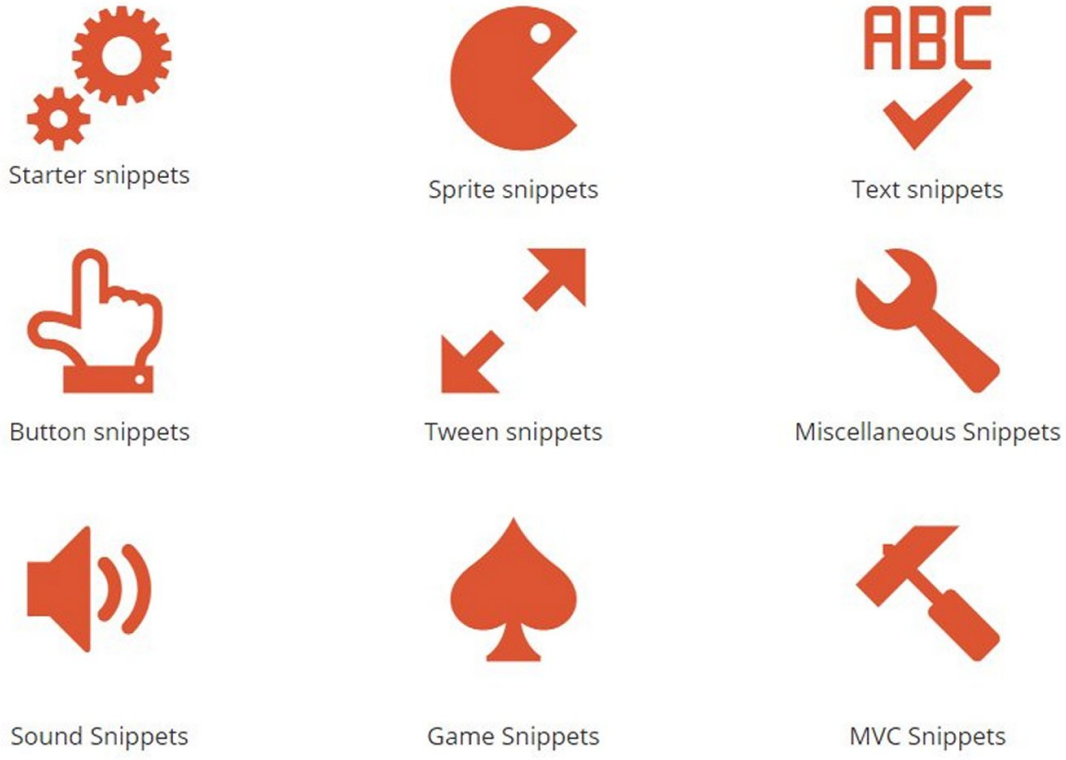
***Exercise:*** Research applications that save "snippets". Here's a good starting point:

- ***JetBrains WebStorm***[209] or,
- ***Snippet Designer for Visual Studio***[210] or,
- ***Snip2Code***[211] a community "Where Coders Share Snippets". Their base service is ***Free!***
- https://codeburst.io/how-to-share-code-and-make-it-shine-f5ffcea1794f
- https://www.codepile.net/home
- Purchase? https://www.qsnipps.com/

You'll quickly discover that "snippet codebases" are not new and have been around for quite some time. The real problem is ***"HOW"*** to organize your codebase. Yes, you could mimic the Phaser website; but to be quite blunt, their naming syntax never made sense to me. Yes, thank you, I realize it's listed by OOP Classes, but I would have arranged the associations different in my own mind. But to use it in "bottom-up design" you must understand completely their "vocabulary" ***taxonomy.***[212] AND, the newest **Phaz3r "documentation"** ... well, enough ranting for one sentence ;)

[204] https://en.wikipedia.org/wiki/Snippet_(programming)
[205] https://en.wikipedia.org/wiki/Codebase
[206] https://makingbrowsergames.com/gameDesigner/index-randommechanic.html
[207] https://www.w3schools.com/appml/appml_architecture.asp
[208] https://www.ics.uci.edu/~wscacchi/GameIndustry/Lecture06-MMORPG-Planning.html
[209] https://blog.jetbrains.com/webstorm/2018/01/using-and-creating-code-snippets/
[210] https://marketplace.visualstudio.com/items?itemName=vs-publisher-2795.SnippetDesigner
[211] https://www.snip2code.com/
[212] https://en.wikipedia.org/wiki/Taxonomy

Let's learn something from one of my colleagues and ***how he organizes his code.***[213] Organization and correct labeling (in your own mind) is the foundation. ***It will greatly help when we begin to create our "Entity Relationship Diagrams"***[214] ***for our codebase.***




**William Clarkson online codebase**

This is an excellent start into ***category classification;***[215] — software professionals cannot hide their mastery in **"taxis" (Greek for "arrangement") and "nomos" (Greek for "law") = Taxonomy.**[216] His website is also a public ***FREE*** access site. Is that what you'd like to have? … Providing your competitors with your raw codebase? My friend, Will, is an instructor —like myself— and makes many **excellent online tutorials**[217] and books. His focus is on helping and training others. If that is your goal then follow our lead. ***However, I'm going to take you on a different path into a private codebase repository similar to what I use.***

[213] https://williamclarkson.net/code/phaser-snippets/
[214] https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/
[215] https://www.britannica.com/science/taxonomy
[216] https://www.britannica.com/science/taxonomy
[217] https://click.linksynergy.com/link?id=sfDExpt0ZWY&offerid=507388.2034380&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fmaking-html5-games-with-phaser-3%2F

## Deeper Dive: Database Protection Considerations

Remember how we cataloged the various prototype components in Chapter 1 ***in the "second (2nd)" Chart?*** Let's use that ***"second (2nd)" Chart*** to begin creating our private codebase. You can store your snippets in simple text files and directories — that's the oldest style of data-basing — using your O/S. OR, we could decide to persist our codebase in the "Cloud" online (i.e., GitHub publicly or privately) so that it's available whenever an idea presents itself.

***Database collections can carry their own legal copyrights!*** Your collection of code snippets, in database form, could become an alternate game product and alternate source of income for your studio.

***Exercise:*** Research these articles:

- ***Intellectual Property Rights: Copyright and databases***[218] quote: "In principle, the facts themselves can not be protected ***but the order and organization can*** if they show a certain level of creativity on the part of the author. When referring to databases it is necessary to distinguish between creative and non-creative databases because each is dealt with under a different set of legal rules. … However, the Directive ***does not protect software used to create the database or for the material contained in the database.*** It is the scheme of the database that is protected."
- ***Database Legal Protection***[219] quote, "Protection for databases under copyright law is provided under the concept of a ***compilation copyright. Compilation copyrights protect the collection and assembling of data or other materials.*** The extent of the protection provided to databases is explained in the following sections … (read more)"

## Database Schema Construction (Copyright-able!!)

Let's catalog the various game prototypes we've created thus far. You'll find in Chapter 1 we***'ve begun***[220] this process with "Chart 2" already. Here are other categories to include:

[218] https://www.esa.int/About_Us/Law_at_ESA/Intellectual_Property_Rights/Copyright_and_databases
[219] https://www.bitlaw.com/copyright/database.html
[220] https://brians.wsu.edu/2016/05/19/began-begun/

- ***Visual —*** which can be any "images, sprites or geometrical shapes" representation displayed. We will keep ***"what is seen separate from what its data information is."***[221]
- ***Objects — "buttons", "text" and "game object" —*** a generic building block containing ***metadata.***[222] Everything in JavaScript is an object, but we'll use this category to mean **non-visual items and source code.**
- ***Events — "coded" —*** will be our "messengers" that look and listen for some set of conditions and respond with and action notice.
- ***Behaviors — "animation" —*** we'll place source code in this category to modify any objects' actions. For example, ***"`physics`"*** behavior makes objects react to events.
- ***Scenes — "game phase/menu" —*** as we will see in later chapters these "camera" views will be our game phases. Each autonomous scene will contain its own objects, events, and HUD.

Aki Järvinen, in his article ***"Introducing Applied Ludology: Hands-on Methods for Game Studies" on page 2 (available here),***[223] provides a way to help identify game mechanisms. He suggests nine (9) categories common in all games. Furthermore, with what we've learned in Chapter 1, we can create our database catalogs.

> Quote: ***"Introducing Applied Ludology: Hands-on Methods for Game Studies"***
>
> Component — The resources for play; what is being moved or modified – physically, virtually, in transactions — in the game, between players and the system. Tokens, tiles, balls, characters, points, vehicles are common examples of game components.
> Context — Where, when, and why the gaming encounter takes place.
> Environment — The space for play – boards, grids, mazes, levels, worlds.
> Game Mechanics — What actions the players take as means to attain goals when playing. Placing, shooting, maneuvering are examples of what players are put to perform in many games.
> Information — What the players need to know and what the game system stores and presents in game states: Points, clues, time limits, etc.
> Interface — In case there are no direct, physical means for the player to access game elements, interface provides a tool to do that.
> Players — Those who play, in various formations and with various motivations, by performing game mechanics in order to attain goals.
> Rule Set — The procedures with which the game system constrains and moderates play, with goal hierarchy as an especially important subset.
> Theme — The subject matter of the game which functions as a metaphor for the system and the rule-set.

[221] https://softwareengineering.stackexchange.com/questions/229479/how-did-separation-of-code-and-data-become-a-practice
[222] https://cuahsi.zendesk.com/hc/en-us/articles/208639877-What-is-Metadata-
[223] https://makingbrowsergames.com/book/IntroducingAppliedLudology.pdf

> By minimum, a game has to have Components, Environment, and at least one Game Mechanic. When the relationships of these three elements are defined and implemented, it means that a Rule-set emerges, as does Information. Then we need Players, and any gaming encounter brings about various Contexts, that may vary from one encounter to the next one.

## Database Record Construction

Suggested snippet record fields:

- snippet (ID)entification record number (primary key as ***"sID"***)
- category grouping (foreign key into "category table" mentioned above using ***"catID"***)
- ***"framework" ID*** grouping (foreign key into "framework table"; 1:1; )
- ***"deployed"*** on either "client", "client-proxy", "server" or "both" (enumeration text)
- mechanism's ***"description"*** narrative (text)
- source code snippet (text as ***"src"***)
- anything else you might like to associate?? Remember to follow ***1NF, 2NF, and 3NF.***[224] when supplementing this suggest record format.

**SQL sample table creation**

```
1   //created with SQLite browser; visit http://sqlitebrowser.org/
2   BEGIN TRANSACTION;
3   CREATE TABLE IF NOT EXISTS `snippets` (
4       `sID`           INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
5       `catID`         INTEGER,
6       `frameworkID`   INTEGER,
7       `deployed`      TEXT NOT NULL DEFAULT 'client, server or both?',
8       `description`   TEXT NOT NULL, DEFAULT 'designer notes here.'
9       `src`           BLOB NOT NULL DEFAULT 'source code goes here.'
10  );
11  COMMIT;
```

Do I need to say that all source code snippets ***must be "self-contained functional programming paradigm??*** They should never expect nor require anything externally.

[224] https://www.guru99.com/database-normalization.html

### Database structure

There are several technologies we might consider to "store" information as either "static pages" or "dynamic pages". I use XML for avatar records and I've found that ***"Microsoft XML Notepad" (GitHub)***[225] a valuable editing tool ( ***learn more about it here***[226] ). It is open-source and easy to use. XML Notepad 2.8.0.11 / 31 May 2020 in the most current release. Another option is the XAML tool integrated into Visual Studio. You can find it by searching for "xmlnotepad".

***Exercise:*** Download the following XML or JSON databases for our codebase:

- **XML:** GetSimple CMS http://get-simple.info/ is an XML-based ***FREE*** content management system. It perfect for storing a codebase of snippets.
- **JSON:** The advantage here is that we can fully automate the game construction process by "gluing together" the various snippets ***directly*** into game mechanic templates.
- ***TaffyDB***[227] — An "open source" library that brings database features into your JavaScript applications using JSON recordsets — similar to ***MongoDB***[228]. *This is a "hint"* on my private snippet collection and ***game production pipeline***[229].

***Hint:*** Once you have a spiffy XML database file, you might like to convert it into JSON. I use ***this online tool***[230] or you might like to pick your own.

## Remote Codebase Using AppML

A simple local snippet pool would have been sufficient for our games and similar to what we did on the ***Game Designer***[231] website. ***Google Actions***[232] uses an "Excel

[225] https://github.com/microsoft/xmlnotepad
[226] https://en.wikipedia.org/wiki/XML_Notepad
[227] http://taffydb.com/
[228] https://en.wikipedia.org/wiki/MongoDB
[229] https://makingbrowsergames.com/gameDesigner/
[230] http://convertjson.com/xml-to-json.htm
[231] https://makingbrowsergames.com/gameDesigner/
[232] https://developers.google.com/actions/

spreadsheet-style" for its pool with similar data content which I plan to use, but we will create a JSON file for our collection. **The data format will be like that used in the Game Designer**[233]. Also, I have decided to use the ***W3School's version of AppML***[234] for this codebase **remote access from my server**. AppML is far more technology than what this simple tool demonstration demands, but it will provide a way to add more features if I choose to grow and upgrade this codebase tool into a deluxe version for purchase at some later time. Instead of AppML, we have another alternative in AngularJS, SQLite, or PouchDB. But let's first review AppML.

**AppML** stands for "Application Modeling Language". AppML runs in any standard HTML page; we will have to do some "tweaking" to display snippets inside a canvas or textbox. AppML was based on HTTP request communication between a web client and the web server. **The AppML-based system was launched in 2001, several months before schedule, as the world's first commercial AJAX application.** Thank goodness AppML provides full HTML, CSS, and JavaScript freedom. AppML makes it easy to create ***Single Page Applications (SPA)*** in a very clean and efficient way. You should visit the ***Game Designer,***[235] if you haven't already. It easily adapts AppML and retrieves generated snippets from my remote server. Originally, AppML was abandoned by its creators in September 2007 but was revived by W3Schools in 2015. Other potential data formats are listed below, and are compared at https://www.w3schools.com/js/js_json_xml.asp:

- **XML — Extensible Markup Language** is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. This was a popular protocol at the turn of the millennium but has been replaced by JSON. It is a very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. Learn more about XML at https://www.w3schools.com/xml/
- **JSON — JavaScript Object Notation** is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types (or any other serializable value). It is a very common data format used for the asynchronous browser-server communication, including as a replacement for XML in some AJAX-style systems. JSON is a language-independent data format. It was derived from JavaScript, but as of 2017, many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is `application/json`. JSON filenames use the extension `.json`. Learn more about https://www.w3schools.com/js/js_json_intro.asp

[233] https://makingbrowsergames.com/gameDesigner/index-randommechanic.html
[234] https://www.w3schools.com/appml/default.asp
[235] https://makingbrowsergames.com/gameDesigner/

**AppML** is a modern JavaScript library for bringing data into HTML applications currently maintained by W3Schools; ***it is free to use. No license is necessary. No installation is required.*** Even if you have never worked with web development before, you will find AppML very easy to use. If you are an experienced web developer, you will soon discover the power of AppML. The AppML language and syntax conform to XML nicely.

* AppML uses XML to describe Internet applications.
* AppML applications are self-descriptive.
* AppML is a declarative language.
* AppML is independent of operating systems.
* AppML uses AJAX asynchronous technology.
* AppML is Open Source.
* AppML is a language created and maintained by the W3Schools.

## Building an AppML application

AppML applications are simple to build. The AppML tutorial[236] could be summarized in this 4-step process.

1. Describe the elements of the snippets application with AppML. (style / XML conformance) Review the record construction above.
2. Save that XML file to a web (or database) server.
3. Link the file to an AppML Web service. ***(See this example)***[237] or ***this page.***[238] or consider putting your snippets ***in the "cloud"***[239] as a service.
4. To change the application later, just change the contents of the XML file and save it, the web service will do the rest.

## Sample AppML codebase (Public Access)

- ***https://makingbrowsergames.com/gameDesigner/***
- ***https://makingbrowsergames.com/gameDesigner/index-genre.html***

Membership subscriptions available for ***Game Design Documentation Tool***[240] and private ***Game Construction Tool.***[241]

[236] https://www.w3schools.com/appml/default.asp
[237] https://www.w3schools.com/appml/appml_data.asp
[238] https://www.w3schools.com/appml/appml_php.asp
[239] https://www.w3schools.com/appml/appml_google_cloud_sql.asp
[240] https://makingbrowsergames.com/gameDesigner/?#
[241] https://www.renown-games.com/shop/index.php?id=prod-members

# Remote codebase Using JSON

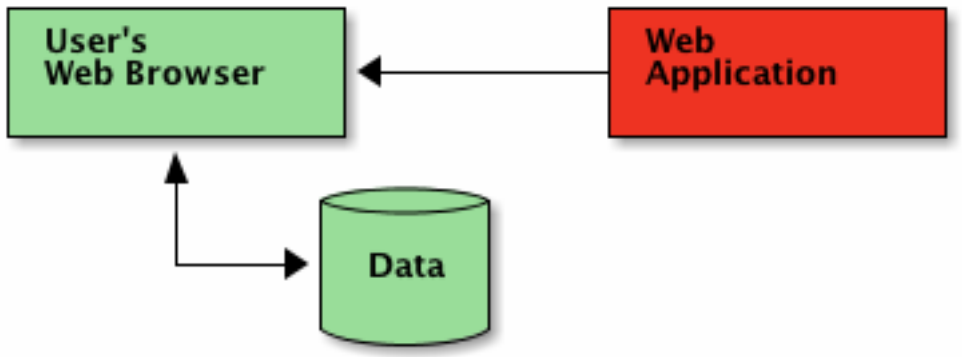
Since AppML is basically the template engine part of AngularJS. Let's also consider using AngularJS and deliver JSON snippet codebase. Research the following instructions:

https://phaser.io/news/2014/11/building-multiplayer-games-with-angular

***Exercise:*** Review the following JSON databases for ***Game Designer***[242]:

## Per-user storage

Nowadays, mobile users expect their data to be synced to "the cloud", and to all their other ***"IoT" devices.***[243] This attitude is also becoming prominent for desktop applications. Traditionally in the past, ISPs stored data in their database servers. However, ***DevOps and new innovations***[244] can allow clients to synchronize and store application data inside their own devices. This is possible from modern browsers **— using CORS —** without going into your gaming server. Allowing gamers to store their data in their local device gives them full control of their data content




**New "unhosted web app" architecture**

It's possible to include such storage capabilities by using simple JavaScript libraries on the client-side. Consider these:

[242] https://makingbrowsergames.com/gameDesigner/
[243] https://internetofthingsagenda.techtarget.com/definition/IoT-device
[244] https://www.atlassian.com/devops

| Per-User Backend | Dropbox | Google Drive | Open Remote Storage | Per-user Hoodie |
|---|---|---|---|---|
| Dropbox.js | Yes | x | x | x |
| Google Drive JS | x | Yes | x | x |
| Hoodie.js | x | x | x | Yes |
| Nimbus Base | Yes | Yes | x | x |
| Remote storage.js | (alpha) | (alpha) | Yes | (planned) |

***NOTE:*** None of these libraries support Apple iCloud nor Microsoft OneDrive. The reason is that those services don't provide a cross-origin "REST+CORS" API. "Open Remote Storage" providers, such as Dropbox and Google, do support "REST+CORS" API.

***Exercise:*** Excited about "per-gamer storage"? Read more from this **FREE 147-page** online book[245] from "Unhosted".

[245] https://unhosted.org/book/

# Chapter Source Code & Demo

book website: https://makingbrowsergames.com/p3gp-book/

Complete Chapter ***Source Code in the online appendix.***[246]

***Play III Game Prototype Demo thus far***[247]

- ***Example 2.4 Bare-bones Index Page - Traditional Method***[248]
- ***Example 2.5: Starting the Game.js***[249]
- ***Example 3.1a: Creating State Objects in Game.js - traditional method***[250]
- ***Example 4.1: Prototyping a Visual Avatars***[251]
- ***Example 4.2: Prototyping Movement Properties in v3***[252]
- ***Example 4.3: Movement Arrows v3 Integration***[253]
- ***Example 4.4: World Boundaries Grouping***[254]
- ***Example 4.5: World Boundaries Integration***[255]
- ***Example 4.6: Interior Boundaries Integration***[256]
- ***Example 4.7: Collision Detection Integration***[257]
- ***Example 4.8: Collision Results Determination***[258]
- ***Example 4.9: New Game Over State***[259]
- ***Example 4.10: Elementary HUD Creation***[260]
- ***Example 4.11: Collecting User Input***[261]
- ***Example 4.12: Responding to User Input***[262]

[246] https://makingbrowsergames.com/p3gp-book/tools.html
[247] https://makingbrowsergames.com/p3gp-book/_p3-demos/ch4-examples/
[248] https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html
[249] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson02.html
[250] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson03.html
[251] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson04.html
[252] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson04.html
[253] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson05.html
[254] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson06.html
[255] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson07.html
[256] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson08.html
[257] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson09.html
[258] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson10.html
[259] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11.html
[260] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson12.html
[261] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11a.html
[262] https://makingbrowsergames.com/p3gp-book/_p3-demos/lesson11a.html

## Summary

Examples:

- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/bareBonesIndex.html***
- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/index.html***
- ***https://makingbrowsergames.com/p3gp-book/_p3-demos/index-OLOO.html***

Here's an inventory of what we've learned thus far.

- Game Prototyping uses simple graphics and focuses on game mechanics.[263]
- Created Game Prototype that accepts inputs.
- Created Game Prototype that moves various game components.
- Created Game Prototype that reacts with internal objects.
- Created a web page to launch our Phaser Prototype.
- Learned about Content Delivery Networks.
- Discovered various game phases and states to ***modularize***[264] our game.
- Learned each Phaser game state has separate functions of which create and update are the most active.
- Studied a typical Skeleton state file.
- Reviewed the traditional game menu states.
- Discovered a Phaser game can use multiple physics engines, but only one physic engine is assigned to one graphics sprite.
- Created a gamer's representation in the game world.
- Learned how to generate sprite graphics from code.
- Attached speed and velocity to moving game objects.
- Attached various input signals to manipulate game objects.
- Attached reactions to immovable and movable objects.
- Learned how to trigger various behaviors.
- Created game stage boundaries.
- Discovered how to transition game between states.

[263] http://www.lostgarden.com/2006/10/what-are-game-mechanics.html
[264] http://www.dictionary.com/browse/modularize

# Chapter References

***(See more "Our references" in the front)***

- ***How to Prototype a Game in Under 7 Days***[265]
- ***MDN Game development***[266]
- ***Game Design Concepts 5.1: Prototyping***[267]
- ***Plain English Guide to JavaScript Prototypes***[268]
- ***JavaScript Classes***[269]
- ***https://www.nolo.com/legal-encyclopedia/types-databases-that-cant-be-protected.html***
- ***https://www.bitlaw.com/copyright/database.html***
- ***https://data.research.cornell.edu/content/intellectual-property***
- ***https://en.wikipedia.org/wiki/Sui_generis_database_right***
- ***https://www.michalsons.com/blog/the-rights-to-a-database/2937***

[265] http://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php?print=1
[266] https://developer.mozilla.org/en-US/docs/Games
[267] https://learn.canvas.net/courses/3/pages/level-5-dot-1-prototyping
[268] http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/
[269] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

# Appendix: OLOO - Safe JavaScript

***A Tutorial for Creating Games in JavaScript using OLOO***

An excerpt from ***"JavaScript 'Objects Linking to Other Objects' (OLOO) in Game Development"***[270] — a **FREE** online course for book patrons studying and contrasting ***JS delegation vs OOP Inheritance.***

## JS Objects: *"TL;DR"*

JavaScript has been plagued since the beginning with misunderstanding and awkwardness around its **"prototypal inheritance"** system, mostly because ***"inheritance" isn't how JS works at all,*** and trying to do that only leads to "gotchas" and "confusions" that we have to pave over with "user-land helper libraries". Instead, embracing that JS has "behavior delegation" — simple delegation links between objects — fits naturally with how JS syntax works, which creates more sensible code without the need of helpers. ...

When you set aside distractions like ***mixins***[a], ***polymorphism,***[b] composition, ***classes,***[c] constructors, and instances, **and only focus on** the objects that link to each other, you gain a powerful tool in behavior delegation that is easier to write, reason about, explain, and code-maintain. ***Simpler is better.*** JS is "objects-only" (OO). Leave the classes to those other languages! ...

At this point of understanding, we should really ask ourselves: is the difficulty of ***expressing classes and inheritance in pure JavaScript***[d] a failure of the language (one which can ***temporarily***[e] be ***solved***[f] with ***user libraries***[g] and ultimately solved by ***additions to the language***[h] like `class { ..... }` syntax), as many game developers ***feel,***[i] or is it something deeper?

Is it indicative of a more fundamental disparity, that we're ***trying to do something in JS***[j] that it's ***just not meant to do?***[k]

[a] https://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/
[b] https://davidwalsh.name/javascript-objects-distractions
[c] https://davidwalsh.name/javascript-objects-deconstruction
[d] http://javascript.crockford.com/inheritance.html
[e] http://prototypejs.org/learn/class-inheritance
[f] http://mootools.net/docs/core/Class/Class
[g] http://ejohn.org/blog/simple-javascript-inheritance/
[h] http://wiki.ecmascript.org/doku.php?id=strawman:maximally_minimal_classes

[270] https://leanpub.com/c/jsoloo

[i] http://www.nczonline.net/blog/2012/10/16/does-javascript-need-classes/
[j] http://www.kirupa.com/html5/objects_classes_javascript.htm
[k] http://webreflection.blogspot.com/2010/01/better-javascript-classes.html

JavaScript features stalled around 2007 to 2008; we've recently seen JavaScript language development making a fair amount of progress with promised releases every year (typically in June). In 2012, `Object.create` appeared in the standards. It allowed us to create objects with a selected prototype but didn't allow us to `get` nor `set` them. So, browsers implemented a non-standard `__proto__` accessor that permitted `getting` and `setting` a prototype at any time. Later in the year 2015, `Object.setPrototypeOf` and `Object.getPrototypeOf` were added to the standards. The `__proto__` was the "de-facto" — implemented everywhere — so, it made its way into the standard's Annex B — a description for optional non-browser environments.

The ***ECMAScript 6-standard***[271] is 4+ years old and ***all major browsers currently support it (see this table).***[272] The next ***ECMA version***[273] was ES6 (or ES2015, or "ESNext" (at that time), there are a lot of names for ***JavaScript release versions***[274]) that were only ***partially supported by browsers back then.***[275] However, since the ES5 specification is fully defined, software engineers wrote tools called ***"transpilers"***[276] — it takes ES6/7/8/9 formatted code and returns it into the standard ES5 code. You must be thinking, **"Why would they do that?"** Because, it allows software engineers to use the newest released versions of the JavaScript specification and all the newest features; yet, still allow their code to be run in any browser. So, this is a perfect time to latch onto[277] the new JS features and migrate to the modern style of game development using Phaser Gaming Frameworks v2.x.x and/or v3.x.x.

Before ECMAScript 6, there was a lot of confusion about how to use ***Object Oriented Programming (OOP) in JavaScript;***[278] the two methods used were either the ***"factory pattern"***[279] or the ***"constructor function pattern".***[280] ES6 added a new keyword to resolve this confusion and provide a single primary method to insert OOP into JS — the `class` keyword was introduced. Many believed, by adding this `class` keyword, it would solve many problems. In reality, it didn't! ***It simply poisoned the minds and added another layer of abstraction that misrepresented the JS prototype-based***

[271] http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm
[272] http://kangax.github.io/compat-table/es2016plus/
[273] https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5
[274] https://www.w3schools.com/js/js_versions.asp
[275] http://kangax.github.io/compat-table/es2016plus/
[276] https://babeljs.io/
[277] https://idioms.thefreedictionary.com/latch+onto
[278] http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/
[279] https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch09s10.html
[280] https://www.safaribooksonline.com/library/view/learning-javascript-design/9781449334840/ch09s01.html

***inheritance chain as masquerading as standard classic OOP inheritance.*** If you're using ES6 with classical Object-Oriented Programming, you will need a different structured approach known as ***Objects Linking to Other Objects (OLOO).*** Yes, there are many ways to style your JS source code. One prevalent way is to ***shoehorn***[281] JS into a ***Classical OOP format with inheritance;***[282] but, to do so is a misuse of the native objects and prototypical delegation found in the core of the JavaScript syntax. In fact, the ***Gang of Four (GoF) states,***[283] ***"... favor object composition over class inheritance ...".***[284] Read what Apple Game Develop says about ***"Inheritance-Based Architecture Hinders Game Design Evolution"***[285]

***ES9 (June 2018)***[286] says this about OOP in JS, (quote pg 48), "... In a class-based object-oriented language, in general, 'state' is carried by instances, methods are carried by classes, and inheritance is only of structure and behavior. In ECMAScript, the state and methods are carried by objects, while structure, behavior, and state are all inherited. All objects that do not directly contain a particular property that their prototype contains share that property and its value."

Furthermore, "ES6 restricts what a `class` body content might contain." Quoted from Exploring ES6 by Dr. Axel Rauschmayer[287]

> ### 15.2.2 Inside the body of a class definition
>
> A class body can ***only contain methods, but not data properties.*** Prototypes having data properties is generally ***considered an anti-pattern,*** so this just enforces a best practice.

## Deeper Dive: JS Delegation (aka "Inheritance"?)

- ***3 Different Kinds of Prototypal Inheritance: ES6+ Edition***[288] by Eric Elliott.
- ***The Gang of Four is wrong and you don't understand delegation***[289] by Jim Gay.

[281] https://www.urbandictionary.com/define.php?term=shoehorn
[282] https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
[283] https://hackernoon.com/favor-object-composition-over-class-inheritance-they-said-9f769659b6e
[284] https://en.wikipedia.org/wiki/Composition_over_inheritance
[285] https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html#//apple_ref/doc/uid/TP40015172-CH6-SW1
[286] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[287] http://exploringjs.com/es6/
[288] https://medium.com/javascript-scene/3-different-kinds-of-prototypal-inheritance-es6-edition-32d777fa16c9
[289] https://www.saturnflyer.com/blog/the-gang-of-four-is-wrong-and-you-dont-understand-delegation

## The old way

Let's assume you're building a game studio/workshop and you need to create some release game products. In the old days (1997 to 2008) – ***before ECMAScript 5 (ES5),*** you would have written something like a function this way:

```
1       //"pseudo-constructor" JS Function; functions are "hoisted"
2       function game (name) {
3               this.name = name;
4       };
5
6       game.prototype.showName = function() {alert(this.name);};
7
8       ////////////////////////////////////////////////
9       // OR when ES5 appeared, you may have shifted to an "object literal"
10      // "pseudo-constructor" JS object literal; object literals are
11      // NOT "hoisted".
12      ////////////////////////////////////////////////
13       var game = functionName(name) {
14              this.name = name;
15      };
16
17      game.showName = function() {alert(this.name);};
18
19      //Phaser v2.x.x uses object literals with "new" keyword.
20      var game = new Phaser.Game(640,450,Phaser.Auto,"gameDiv",boot);
21
22      document.addEventListerner('DOMContentLoaded',
23              function(){ window.game() );, false  });
```

This wasn't so bad. You have a function acting as a pseudo-***"constructor"***[290] (an adopted terminology from OOP) to create your ***game object singleton***[291] and attach methods to the game's prototype. All individual games would have these methods. "Constructor functions" are technically just normal, old, regular functions ***— nothing more!*** For instance:

[290] https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)
[291] http://gameprogrammingpatterns.com/singleton.html

```
function User(name) {
  this.name = name;
  this.isAdmin = false;
}

let user = new User("Jack");

alert(user.name);               // Jack
alert(user.isAdmin);        // false
```

When ***"new User(...)"*** is called, it does several things:

- A new empty object was created.
- The ***"this"*** keyword was assigned to that newly created object. Additionally, the ***"constructor"*** property was changed to the parent function and ***"__proto__"*** was set to that parent's ***"constructor"*** prototype. If no return value was set at the end of this function, then the function would return ***"this"*** — a reference to the object itself.
- The function body executes its statements and usually, it modifies `this` reference and adds other new properties into itself.
- The value of ***"this"*** was returned upon completion.

In other words, ***"new User(...)"*** really does something like this:

```
function User(name) {
  // this = {};  (implicitly)

  // add properties to this
  this.name = name;
  this.isAdmin = false;

  // return this;  (implicitly)
}

//or something like this
var User = {
    constructor:
        this.name = name;
        this.isAdmin = false;

    //If we put methods here; every instance will use these
```

```
18          //  same methods.
19      }
```

However, there are several concepts to consider:

- There are **no private attributes** in the JavaScript language unless you choose to modify the `Object.defineProperty read-only` ***into a `writable` flag to `false`.***[292] ***Be warned!*** Once you change this, you ***cannot*** reverse your selection. The value inside a named variable can be changed at any time from outside its function; unless you set the property as "ready only".
- The **methods and properties are scattered**. Even if you put them all in one tight collection (aka a `class`), there is not a `single structure concept` to define your classic classful objects due to "hoisting" — just as you would do with classes in many other purely classical object-oriented languages. Objects use "values by reference".
- You might easily **forget to use the `new` keyword**. It doesn't throw an error and is perfectly acceptable in JS. It just leads to completely different behavior than you would expect and some nasty unintended bugs that become difficult to discover.
- **Classical OOP Inheritance** will provide further problems. There isn't an agreed-upon method in the JS-community about how to do this properly. In fact, in JS, ***it is not inheritance at all (i.e., copying attributes and method into the `new` instance). Nothing in an object/function is "copied" in JavaScript.*** When an object variable is copied — it's the reference (aka the memory address where the value lives) that is copied, ***the object is not duplicated.***

If an internal property doesn't exist in an object, ***JavaScript refers up the protocol chain to find it; it is "delegation".*** John Dugan has an excellent illustration.

> ***"Object Oriented JavaScript Pattern Comparison"***
>
> "When your parents had you, you inherited their DNA — you ***received a copy of it. When they broke their leg, yours did not break.*** JavaScript is the opposite of this. In JavaScript, ***when your parents break their leg, yours breaks too.*** A term better suited than prototypical inheritance to JavaScript is ***prototypical delegation.*** When a `new` object is created from another object in JavaScript, ***it links back to the parent object's prototype properties and methods as opposed to copying them."***

[292] http://www.ecma-international.org/ecma-262/7.0/index.html#sec-property-attributes

( Read more on this topic from, ***"You don't know JS" Chapter 6***[293] ).

Here's a summary of the ***Feature changes in ES6 here***[294])

**A class with only a single instance with global access points.**

```
1    // v2.x.x
2    var game = new Phaser.Game(480, 320, Phaser.AUTO, null, game.Boot);
3
4    // v3.x.x
5    var game = new Phaser.Game(config);  OR
6    var game = Object.assign({},Phaser.Game(config));  OR
7    var game.prototype = Object.create(Phaser.prototype);
```

When the `new` keyword is placed in front of any `function` call, four things happen:

1. A `new` object is created and assigned to the variable; `new` helps create an object from the parent function.
2. The `new` object gets linked into the parent object's `prototype`.
3. The `new` object gets associated with the keyword `this` within the `constructor function` call.
4. If the `constructor function` does not return a value, JavaScript ***implicitly inserts `this` context*** and returns it as a reference at the end of the constructor function's execution.

### ECMA-262 7th Edition / June 2016

ECMAScript® 2016 Language Specification[a]

(QUOTE) "A function object is an object that supports the `[[Call]]` internal methods. A constructor (also referred to as a constructor function) is a function object that supports the `[[Construct]]` internal method."

Refer to ***Table 6***[b]
***A function object is not necessarily a constructor and such non-constructor function objects do not have a `[[Construct]]` internal method.***

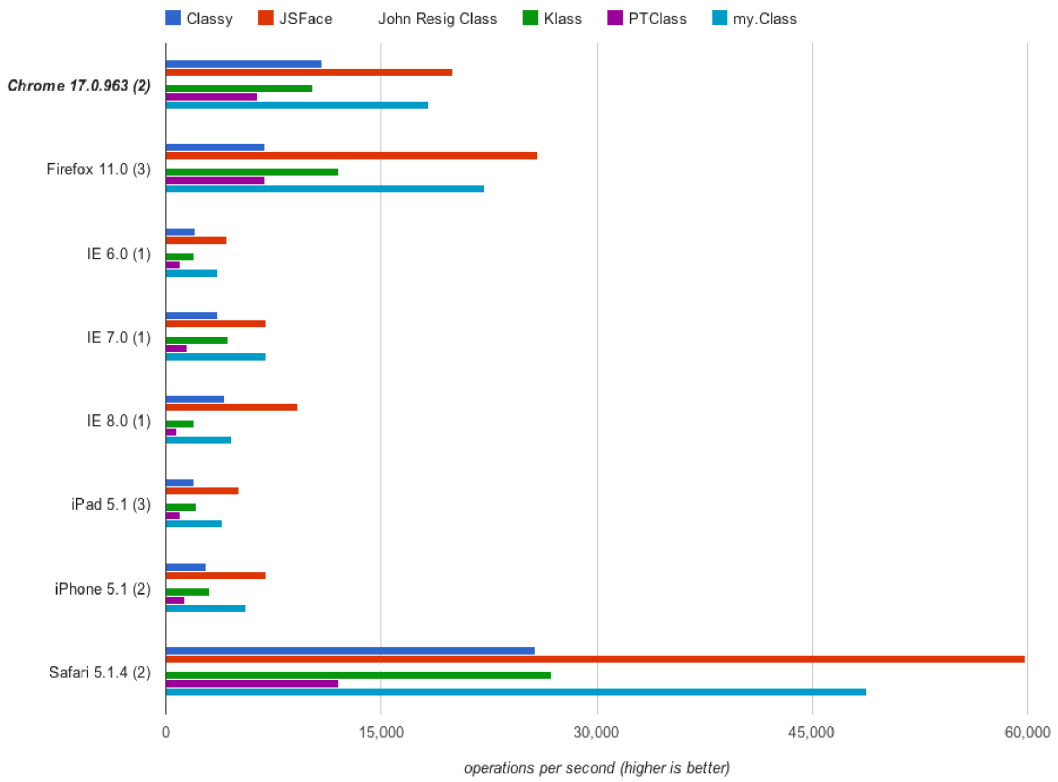
[a] http://www.ecma-international.org/ecma-262/7.0/index.html#table-6
[b] http://www.ecma-international.org/ecma-262/7.0/index.html#table-6

[293] https://github.com/getify/You-Dont-Know-JS/blob/1st-ed/this%20%26%20object%20prototypes/ch6.md
[294] http://es6-features.org/#Constants

Because of these points, many developers have created libraries, frameworks, and tools that provide all types of object creation and instantiation logic ***to subjugate JS into the classical OOP comfort-zone.*** Many of these "shackles", introduced by OOP classes (e.g. Prototype[295], ES6, TypeScript, LiveScript[296], or CoffeeScript[297]), are nothing more than ***"syntactic sugar"***[298] ***to fatten up***[299] JS. I trust you haven't drunk this pre-sweetened "Kool-Aid"!?[300] (oh! I offer my apology[301] to those who've misunderstood my meaning).

You can skip down this yellow-brick road[302] of classic OOP if you want, but I wouldn't recommend it because ***your game will perform slower.*** Test it for yourself[303]; you'll see that ***using OOP is 12% slower (click to see results)***[304] ***because of hierarchy "tree walking".*** It ***may be hard to swallow (Kool-Aid reference above)***[305], yet prototypical delegation — Objects Linking to Other Objects (OLOO) — is much easier than classful-based OOP and provides additional further benefits. Just look at other prototypical languages such as ***IO***[306] ***or Self***[307]***.*** These are an old pre-JavaScript syntax that made prototypes initially difficult to use.

[295] http://www.prototypejs.org/
[296] http://livescript.net/
[297] http://coffeescript.org/
[298] https://github.com/getify/You-Dont-Know-JS/blob/2nd-ed/objects-classes/apA.md#class-gotchas
[299] https://idioms.thefreedictionary.com/fatten+up
[300] https://www.urbandictionary.com/define.php?term=drink%20the%20kool-aid
[301] https://www.washingtonpost.com/posteverything/wp/2014/11/18/the-phrase-drank-the-koolaid-is-completely-offensive-we-should-stop-saying-it-immediately/?noredirect=on&utm_term=.c0835724bc46
[302] https://imgur.com/gallery/NUKAetS
[303] https://jsperf.com/crockford-object-create-with-cached-function
[304] https://makingbrowsergames.com/starterkits/_CrockfordObjectComparisonTest.pdf
[305] https://idioms.thefreedictionary.com/hard+to+swallow
[306] https://en.wikipedia.org/wiki/Io_(programming_language)
[307] https://en.wikipedia.org/wiki/Self_(programming_language)




**Comparison of JS OOP Class Systems**

Douglas Crockford developed ***this approach***[308]. He wrote a short method — called `Object.create` — and it was adopted into the ES5 standards.

[308] http://javascript.crockford.com/prototypal.html

# Objects Linking to Other Objects (OLOO)

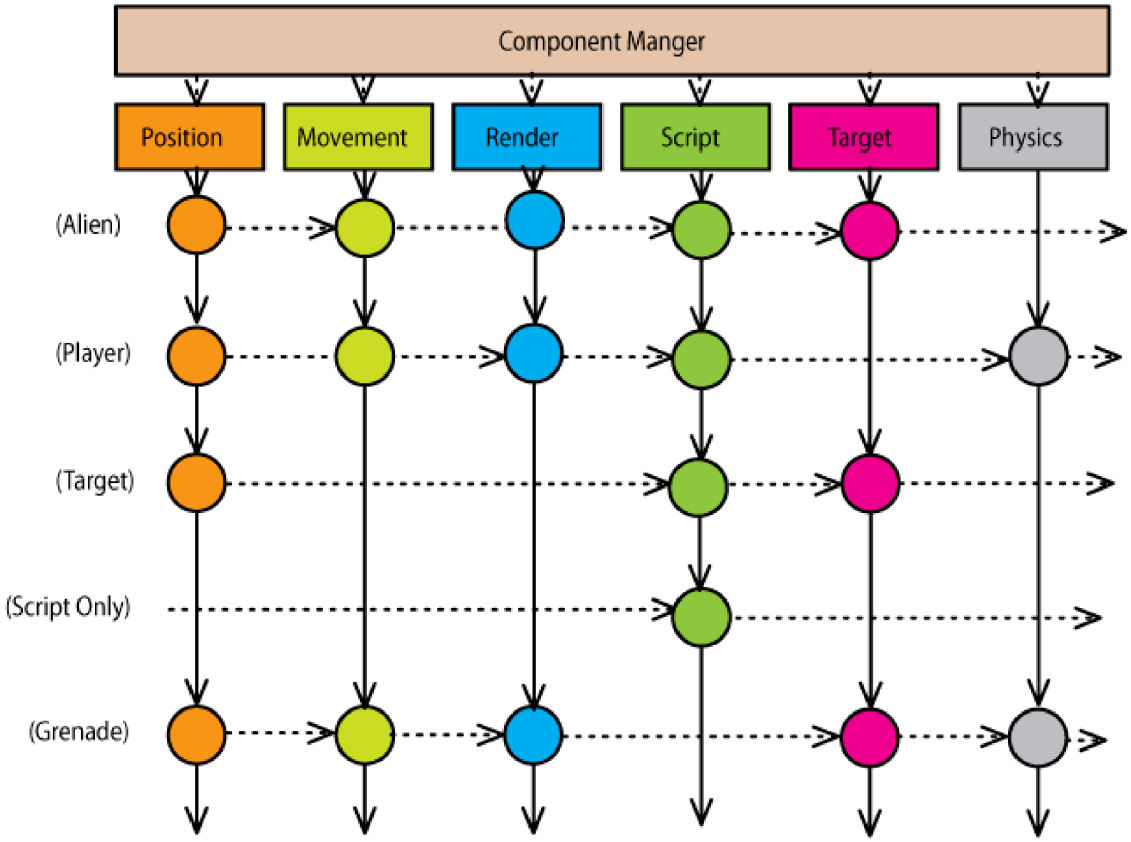



Figure 2 Object composition using components, viewed as a grid.

**Entities and Component Game Design viewed as a cross-reference**

In the OLOO style of creating objects, we strip away the "class" focus of objects typically seen in ES6 oriented programming and embrace the true nature of JavaScript's prototype features. In OLOO, objects delegate directly from other objects without needing to use a `constructor` as a middleman.

OLOO takes advantage of the `Object.create` method to take care of object creation and `inheritance`.

```
var myGame = {
    init: function config(width, height) {
        this.width = width;
        this.height = height;
    },

    gameDim: function gameDim(config) {
        console.log('Dimensions: ' + this.width + 'x' + this.height);
    }
}

var gameCanvas = Object.create(myGame); //.init('800', '500');

//Debug Review
console.log("Game prototype (myGame): =======");
console.log("Match TYPE: "+myGame.isPrototypeOf(gameCanvas));
console.log("GET myGame: "+Object.getPrototypeOf(myGame));
console.log("myGame Properties: "+Object.values(myGame));
console.log("End of (myGame): =======");
```

Read more on other `Object` properties here.[309]

## Compare your code

You can check the finished code for this lesson in the live demo below, and run it to understand how it works:

***https://makingbrowsergames.com/jsoloo/index-OLOO.pdf***

## Object.create

But what does `Object.create` do exactly? How is it different from OOP constructors?

[309] http://www.ecma-international.org/ecma-262/7.0/index.html#table-5

`Object.create` is like an OOP `constructor` in that it creates a `new` object from the referenced object passed into it and builds a chain (aka inherits). In other words, it changes the value of `__proto__` on the newly created object pointing to the object referenced. However, the true advantage of `Object.create` is its emphasis on prototype chains and delegation. To get a better idea of what `Object.create` does, you can rewrite it as a new function:

```
1     function createObject(obj) {
2         var newObj = {};                 //new object created
3         newObj.__proto__ = obj;         //chain to properties of referenced obj
4         return newObj;                          //return "THIS" newObj reference
5     }
```

As you can see (in the code above), the new object (`newObj`) created doesn't have properties nor method behaviors. It inherits all of those from the referenced object handed in as an argument. If we should call any method on this new bare object, our program would find that method following the `__proto__` property (instead of inside the object itself).

However, if you plan to create unique properties for this object, you need to create an `initialize` method as in the former example using `config`.

There is a drawback for `Object.create`. It doesn't allow you to use `instanceof` for quick "inheritance" checks ***because it doesn't touch the constructor property.*** Instead, to check for an inheritance, you use the `.isPrototypeOf` method on the originally referenced object.

The good news is that you don't need a modern browser with any ES5 implementation. Mozilla Developer Network provides a polyfill[310]. It allows you to use this way of creating objects ***since 2011, even inside older browsers.*** But before you get all happy with this free gift, refer to https://caniuse.com/#search=ES5 and update your information — today this is not an issue anymore. This is important, considering the adoption trends of browsers[311]. It is not an issue if you only target Firefox, Chrome, Safari, or Opera users. If you have doubts, research this compatibility chart[312] or go to http://canIuse.com.

[310] https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/create/
[311] http://www.w3schools.com/browsers/browsers_explorer.asp
[312] http://kangax.github.com/es5-compat-table/

```
1     //Mozilla Polyfill:
2     if (! Object.create) {
3             Object.create = function (o) {
4                     if (arguments.length &gt; 1) {
5                             throw new Error('Object.create implementation
6                             only accepts the first parameter.');
7                     }
8             }
9     }
10
11    function F() {
12            F.prototype = o;
13            return new F();
14    };
```

You will have a function for the Object.create by just including the above source code. The first line checks whether `Object.create` exists already. ***This ensures that it won't override any native implementation should the browser produce one***.

***Basic Syntax:***

```
Object.create(prototype_object, propertiesObject)
```

## Exercise Lesson 9:

The issues derive from writing source code in ES6 and using transpilers; don't believe me?? Research and record information from here[313]

***Instructions:***
- Click on the "6" or "2016+" tab at the top of the page.
- Find the "Syntax" in the Feature Name column and then `drop-down` "default function parameters" Slide down the new list that appears and find "**new** Function() support"

Record which compilers/polyfills support the "new Function()" feature.

Record which compilers/polyfills support the "new.target" feature.

[313] http://kangax.github.com/es5-compat-table/

Record which compilers/polyfills support the Functions “class” feature.

Record which compilers/polyfills support the Functions “super” feature.

Do any compilers/polyfills support Annex b.

How many total features are supported by:

1. TypeScript + core.js;
2. Babel 6 or 7 + core.js

## Game Singletons

> “Let’s take another look at our Phaser game project. If you would like to create only one game, there is no need for Object.create. Just create the object directly. ***All JS objects are “singletons”.*** A Singleton is a class with only a single instance with global access points. In my example below, you don’t need (and cannot) create an instance of the game object (mistaken for a class), it already exists. “... So, you simply start using the instance. In “classical” languages such as Java, singleton means that you can have only one single instance of this class at any time, you cannot create more objects of the same class. However, in JavaScript (no classes, remember?) this concept makes no sense anymore since all objects are singletons to begin with.” quoted from Stoyan Stefanov — a Facebook engineer and O’Reilly author[a]
>
> [a] http://www.phpied.com/3-ways-to-define-a-javascript-class/

```
1     // creates our game name-space
2     var game = {} || game;
3
4     /**
5     ////////////////////////////////////////////
6     // PHASER v2.x.x
7     ////////////////////////////////////////////
8     // using traditional new keyword found in all Phaser Tutorials
9
10    game = new Phaser.Game(
11        this.viewportWidth,
12        this.viewportHeight,
13        Phaser.AUTO,
```

```
14          document.body,
15          window.GAMEAPP.state.boot);
16
17      */
18
19      // main function - using Object.create
20      main: function(){
21          //game IS Phaser!!
22          game = Object.create(Phaser);
23      },
24
25      ////////////////////////////////////////////
26      // NEW Phaser v3.x.x method
27      ////////////////////////////////////////////
28      //    passing via Object.create
29      //    config submitted as an external object
30      main: function(){
31              this.game = Object.create(Phaser);
32      },
33
34
35      /**
36      // config object embedded into Phaser v3 instantiation
37      this.game = new Phaser.Game(
38          //configuration object submitted to Phaser v3
39          {
40              type: Phaser.AUTO,
41              parent: document.body,
42              scene: [],
43              width: window.Game.viewportWidth,
44              height: window.Game.viewportHeight
45          }
46      );
47      */
```

Download the ***main.js***[314] example file.

However, this isn't our best solution. You might manipulate those properties inside the object easily from outside and there are no precautions for the assigned values. Take into consideration an assignment like `game.price = -20`. A mistake like this would ensure the sale of your game products quickly with really happy customers —

[314] https://makingbrowsergames.com/jsoloo/OLOO-v2gameSkeleton.pdf

customers who receive $20 bucks from you with every game purchased. Your studio wouldn't be able to do that for very long!

Years of experience in object-oriented programming and design tells us to separate the **inner states** of an object from its **outer interfaces**. You usually want to make your game attributes private and provide some protection methods.

## Deeper Dive: Object Manipulation objects in ES5/6

Objects get a major overhaul in ES6. Things like object destructuring and rest/spread operators made working with objects very easy. Let's jump to the code and try to merge two objects in ES5.

```
1    var obj1 = { a: 1, b: 2 }
2    var obj2 = { a: 2, c: 3, d: 4}
3    var obj3 = Object.assign(obj1, obj2)
```

We have to merge the object using Object.assign() which takes both objects as input and outputs the merged object. Let's take a look at how we can tackle this problem in ES6.

```
1    const obj1 = { a: 1, b: 2 }
2    const obj2 = { a: 2, c: 3, d: 4}
3    const obj3 = {...obj1, ...obj2}
```

Simple isn't it? The spread operator makes merging objects a breeze for the developer. But how does that apply to Phaser v2.x.x or III? Well, you've noticed by now that there are segregated `preload`, `config`, `create`, and `updates` function along with the global window. Why not simply write them once and ***"Merge"*** them like this.

```
1    var preload = { loads all game assets };
2    var create = { assigns cached assets to scene }
3
4    var GameMechanics = Object.assign(preload, create);
5    var scene1 = Object.assign(GameMechanics,Phaser.Scene);
```

## Lesson Summary

As of 20180720, "328 out of 330 liked/approved" this answer provided on ***Understanding the difference between Object.create() and new SomeFunction()***[315]

> (Quote) "Very simply said, `new X` is `Object.create(X.prototype)` with additionally running the constructor function. (And giving the constructor the chance to return the actual object that should be the result of the expression instead of this.)
>
> That's it. :)
>
> The rest of the answers are just confusing ***because apparently nobody else reads the definition of `new` either. ;)"***

## Resource References:

- ***Not Awesome ES6 Classes***[316]
- ***How to fix the ES6 class keyword***[317]
- ***ECMA-262 7.0***[318]
- http://www.crockford.com/javascript/inheritance.html
- http://crockford.com/javascript/

[315] https://stackoverflow.com/questions/4166616/understanding-the-difference-between-object-create-and-new-somefunction
[316] https://github.com/petsel/not-awesome-es6-classes/blob/master/README.md
[317] https://medium.com/javascript-scene/how-to-fix-the-es6-class-keyword-2d42bb3f4caf
[318] http://www.ecma-international.org/ecma-262/7.0/index.html#sec-createdynamicfunction

# Appendix: Common Pitfalls

This section helps those new to game development. Senior Software Engineers, no doubt, have learned these from past experiences — it's not fun smashing your thumb, yet sometimes accidents happen all over again ***" ;) "!!***

## Lacking Debugging Tools?

While writing code, it's common to write errors. Errors come incorrect syntax; these are easily repaired with a "good" Integrated Development Environment (IDE) editor. Hidden logical and code flow are more difficult to find. Such errors come from ambiguity in logic flow as a result of client and programmers' product definitions. These errors can remain invisible to the programmer's eye and can create havoc or appear under unexpected circumstances. To identify these errors, we need "Debugger Tools" that can run through the program, and provide hints on why the code is not working as expected.

I've found ***"Log Rocket"***[319] extremely helpful in both local testing **AND IN THE CLOUD.**

Mozilla states, "The Performance tool gives you insight into your site's general responsiveness, JavaScript and layout performance. With the Performance tool, you create a recording, or profile, of your site over some time. The tool then shows you an overview of the things the browser was doing to render your site over the profile and a graph of the frame rate over the profile."

One of the simplest tools is in JavaScript already — ***debugger!*** The debugger keyword is used in the code to force stop the execution of the code at a breaking point and calls the debugging function. The debugger function is executed if any debugging is needed at all else no action is performed.

[319] https://logrocket.com/

**JavaScript debugger**

```
    // html page
    <p>The solution of 20 * 5 is: <span id="test"></span> </p>

    <script>
        var x = 20;
        var y = 5;
        var z = x * y;

        debugger;

    document.getElementById("test").innerHTML = z;
    </script>
```

Generally, you can enter the "Developers tool" section by pressing **the** F12 key and go to the "Sources" tab. In the source tab section, select any JavaScript file and set breakpoints by either selecting from the provided list like DOM breakpoints or "Event listener" breakpoints. This will stop the code execution whenever your chosen event occurs.

***Note:*** Uses the frame rate and Waterfall tools to highlight performance problems caused by long-running JavaScript — such a Phaser Game running in the canvas — and how using workers can help in this situation.

## Deeper Dive: Console Commands

***Exercise:*** Review ***Console Commands here.***[320]

***Exercise:*** Study how to use the console from the ***Chrome Console Tutorial here***[321]

- Console.assert() — Log a message and stack trace to console if the first argument is false.
- Console.clear() — Clear the console.

[320] https://developer.mozilla.org/en-US/docs/Web/API/Console
[321] https://developers.google.com/web/tools/chrome-devtools/console/

- Console.count() — Log the number of times this line has been called with the given label.
- Console.countReset() — Resets the value of the counter with the given label.
- Console.debug() — Outputs a message to the console with the log level "debug". ***Note:*** Starting with Chromium 58 this method only appears in Chromium browser consoles when level "Verbose" is selected.
- Console.dir() — Displays an interactive listing of the properties of a specified JavaScript object. This listing lets you use disclosure triangles to examine the contents of child objects.
- Console.dirxml() — Displays an XML/HTML Element representation of the specified object if possible or the JavaScript Object view if it is not possible.
- Console.error() — Outputs an error message. You may use string substitution and additional arguments with this method.
- Console.exception() — An alias for error().
- Console.group() — Creates a new inline group, indenting all following output by another level. To move back out a level, call ***"`groupEnd()`".***

**Chrome Tutorial: Using Console Groups**

```
1     function name(obj) {
2       console.group('name');
3       console.log('first: ', obj.first);
4       console.log('middle: ', obj.middle);
5       console.log('last: ', obj.last);
6       console.groupEnd();
7     }
8
9     function doStuff() {
10      console.group('doStuff()');
11      name({"first":"Wile","middle":"E","last":"coyote"});
12      console.groupEnd();
13    }
14
15    doStuff();
```

**Chrome Tutorial: Using Console Groups**

- Console.groupCollapsed() — Creates a new inline group, indenting all following output by another level. However, unlike group() this starts with the inline group collapsed requiring the use of a disclosure button to expand it. To move back out a level, call groupEnd().
- Console.groupEnd() — Exits the current inline group.
- Console.info() — Informative logging of information. You may use string substitution and additional arguments with this method.
- Console.log() — For general output of logging information. You may use string substitution and additional arguments with this method. ***NOTE:*** Precision formatting doesn't work in Chrome
- Console.profile() — Starts the browser's built-in profiler (for example, the Firefox performance tool). You can specify an optional name for the profile.

- Console.profileEnd() — Stops the profiler. You can see the resulting profile in the browser's performance tool (for example, the Firefox performance tool).
- Console.table() — Displays tabular data as a table.

**Chrome Tutorial: Using Console Groups**

```
    let data = [
      { name: "Yusuf", age: 26 },
      { age: 34, name: "Chen" }
    ];

    console.table(data);
```

**Chrome Tutorial: Using Console Table**

- Console.time() — Starts a timer with a name specified as an input parameter. Up to 10,000 simultaneous timers can run on a given page.
- Console.timeEnd() — Stops the specified timer and logs the elapsed time in seconds since it started.
- Console.timeLog() — Logs the value of the specified timer to the console.
- Console.timeStamp() — Adds a marker to the browser's Timeline or Waterfall tool. Refer to more details in ***"Using the Timeline Tool" guide***[322]

[322] https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool

**Chrome Tutorial: Using Console timeStamp Label**

- Console.trace() — Outputs a stack trace.

**Chrome Tutorial: Using Console Trace**

- Console.warn() — Outputs a warning message. You may use string substitution and additional arguments with this method.

***Hint:*** Not all of these console commands are available in all browsers. Review their ***availability here***[323]

[323] https://developer.mozilla.org/en-US/docs/Web/API/Console#Browser_compatibility

## Same "Name-spaces"

Whenever "`animations`", prefabrication, and audio sounds files share the same namespace, every resource is managed by the `Resource Manager`; and thereby, taxing the CPU processing capabilities. For this reason, their name must be uniquely stored in separate namespaces. For example, a sound file cannot have the same name as an animation file.

It is recommended to prefix the audio file names with a common moniker label. For example, sound effects with the ***"sfx"*** file-name prefix, to avoid such conflicts.

## Callbacks

Quote from ***"How Do I Organise Files in A Phaser.js Project?"***[324]

> When using ***<script>*** tags in an HTML page, many new JavaScript developers mistakenly include the ***<script>*** tags referencing their libraries in the wrong order (for instance, by adding their reference to Phaser after the reference to their source code). This would result in their game's code executing, but not knowing about Phaser yet. The code will break and throw an error similar to Phaser is not defined in the console. ***... read more***[a]
>
> [a] https://glcheetham.name/2016/03/18/organise-files-phaserjs-project/

**JS scripts inserted in the wrong order.**

```
1  <html>
2      <head>
3          <script src="localhost:3000/my-game.js"></script>
4          <script src="localhost:3000/phaser.js"></script>
5      </head>
6      <body>
7          <!-- WRONG ... Will break. Phaser is not defined -->
8      </body>
9  </html>
```

[324] https://glcheetham.name/2016/03/18/organise-files-phaserjs-project/

# Missing Documentation

Lacking source code documentation and internal comments, in my opinion, is ***the worst sin***[325] ***a software engineer could commit!*** I can recall, dozens of times to my confessed shame, when I return to a game written decades ago only to discover missing documentation and no clue *"what the @#$^!!"* (aka ***naughty explanatory***[326]) I was thinking at that time. ***Can you relate senior software engineers??***

It takes mere moments to insert comments into source code; and **if you don't have time** then use ***"Dragon Speak"***[327] or its ***professional version***[328] if you're running a financially successful studio.

Creating documentation is a snap with open source tools such as ***JSDoc 3***[329] JSDoc 3 is an API documentation generator for JavaScript only. JSDoc even has a ***toolkit***[330] that can export into ***"`html`" or "`JSON`"*** to simplify the process of creating supporting documentation.

***Exercise:*** Return to Chapter 4 Game Recipe™ Automation Tool and see how the JSDoc toolkit can become a supporting feature when exporting parameter types.

Some of the more popular annotation tags used in modern JSDoc are:

| Tag | Description |
|---|---|
| @author | Developer's name |
| @constructor | Marks a function as a constructor |
| @deprecated | Marks a method as deprecated |
| @exception | Synonym for @throws |
| @exports | Identifies a member that is exported by the module |
| @param | Documents a method parameter; a data type indicator can be added between curly braces |
| @private | Signifies that a member is private |
| @return | Documents a return value |
| @returns | Synonym for @return |
| @see | Documents an association to another object |
| @todo | Documents something that is missing/open |
| @this | Specifies the type of the object to which the keyword "this" refers within a function. |

[325] https://www.merriam-webster.com/dictionary/sin
[326] https://biblehub.com/commentaries/proverbs/6-12.htm
[327] https://amzn.to/2q51UCN
[328] https://amzn.to/2S4jVOA
[329] https://github.com/jsdoc3/jsdoc
[330] https://en.wikipedia.org/wiki/JSDoc

| Tag | Description |
|---|---|
| @throws | Documents an exception thrown by a method |
| @version | Provides the version number of a library |

***Exercise:*** Follow the JSDoc 3 tutorials, commands, and articles on ***http://usejsdoc.org/***

***Exercise:*** ***Compare various documentation generators here***[331] and select the one best suited to your workflow.

## Deeper Dive: What is Dragon Speak

- Dragon Professional Individual 15 makes it easy to get started with speech recognition and become proficient quickly with regular use, delivering up to 99% speech recognition accuracy
- Define simple voice commands to shortcut repetitive processes speed up document creation and boost your productivity; easily create custom words such as proper names and specific industry terminology
- Supports Nuance-approved digital voice recorders and smart-phones for advanced recording functionality and can automatically transcribe the audio files to text back at your PC
- Sync with separate Dragon Anywhere Mobile Solution; letting you create and edit documents of any length by voice directly on your iOS or Android device
- Helps prevent fatigue and repetitive stress injuries by offering an ergonomic alternative to the keyboard; supports Section 508 standards to eliminate barriers for those with disabilities that limit their ability to use a keyboard and mouse

[331] https://en.wikipedia.org/wiki/Comparison_of_documentation_generators

# Answers to Exercises

## Appendix

### Appendix: OLOO - Safe JavaScript

1.
2.
3.
4.
5.
6.