

# Phaser III Game Prototyping

Building 100s of games using HTML and Phaser3 JavaScript Gaming Framework

Stephen Gose

# Phaser III Game Prototyping

Building 100s of games using HTML and Phaser3 JavaScript Gaming Framework

Stephen Gose



This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© Copyright, 2008-2018, Stephen Gose. All rights reserved.

## **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 readin the new Phaser III Game Prototyping.

The suggested hashtag for this book is #Phaser3.

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

#Phaser3



## **Also By Stephen Gose**

Voice of Foreign Exchange

Game Template Mechanics for ActionScript JigSaw Puzzles

Phaser Game Prototyping

Phaser Multiplayer Gaming Systems

Phaser Game Starter Kit Collection

Kiwi Game Design Workbook

Phaser Game Design Workbook

Making Dress-UP Browser Games with Phaser v2

Using JavaScript OOO in game development

Making Dating & Quiz Games

Making Puzzle Browser Games with Phaser v2

Phaser v2 Game Design Workshop Course

Phaser III Game Design Workshop

Making Peg Solitaire Browser Games with Phaser v2

Phaser III Game Design Workbook

Phaser III Game Starter Kit Collection

Olympic Game Design System™

Making RPG Games with Phaser v2

Making Phaser III Peg Solitaire Browser Games

Blood Pit™ - a "Walk-Thru Tutorial Series"

WarLord Tourney - a Collection of Gaming Systems



*For my student@ Early Career Academy, Tempe, AZ*

*and @ ITT Technical Institute, Tempe, AZ*

*and more currently*

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

# Contents

<b>Distribution Permission</b> .....	i
Viewing this e-Book .....	i
<b>Disclaimer</b> .....	iii
<b>Forwards</b> .....	v
<b>About this Workbook:</b> .....	vii
<b>Workbook Content:</b> .....	ix
<b>Book formatting:</b> .....	xi
<b>Who This Book Is For:</b> .....	xiii
Your newly obtained skills... .....	xiii
<b>Bonus Content (4th Edition)</b> .....	xv
Game Design System™ Recipes: .....	xv
Our References: .....	xvi
Tweet This Book! .....	xvii
<b>Book Examples &amp; Samples</b> .....	xix

## Part I - Concept & Design ..... 1

<b>1. Game Studio &amp; Project Setups</b> .....	5
1.1 Workstation Setup .....	6
Batteries not included ... Web Server Required .....	6
Development Tools .....	9
1.2 Project Setup .....	10
Deeper Dive: Project Data Structure .....	10
Deeper Dive: And its name shall be called .....	11
Project Directories & Files .....	15
1.3 Game Project Preparations .....	18
What makes a Good Game? .....	18

## CONTENTS

1.4	Preparing a “Game Recipe™” . . . . .	19
	What are you making? . . . . .	20
	What technology will you use? . . . . .	21
	What features are included? . . . . .	26
	What features are mandatory? . . . . .	32
	How will you encode it? . . . . .	36
	Design Architecture: “Oh! Oh!” . . . . .	36
	Design Architecture: “Top-down” . . . . .	39
	Design Architecture: “Bottom-up” . . . . .	39
	“Oh! Oh!” vs. Top-Down vs. Bottom-Up . . . . .	40
	What’s your time-line? . . . . .	40
	Are you ready? . . . . .	41
1.5	Game Recipe™ Summarized: . . . . .	42
	Development: . . . . .	42
	Design: . . . . .	42
	Encoding: . . . . .	43
1.6	Summary . . . . .	43
1.7	Chapter References: . . . . .	44
<b>2.</b>	<b>Building a Game Launch Pad . . . . .</b>	<b>47</b>
2.1	Example: Box Graphics Prototypes . . . . .	49
2.2	“ToTo, ... we’re not in Kansas anymore” — Dorothy . . . . .	54
2.3	Creating Prototype Mechanisms — 4-Step method . . . . .	56
2.4	Game Recipe™ Step #1) the Front-Door . . . . .	58
	Task #1-1 Instructions: . . . . .	59
	Compare your code . . . . .	62
	Mobile Single Web Page Applications . . . . .	62
2.5	Task #2: Launching a Game . . . . .	64
	Launching a Phaser III Game . . . . .	65
	Deeper Dive: Launching the Game. . . . .	68
	Deeper Dive: Optional Game Config . . . . .	68
	Deeper Dive: To Infinity and Beyond! . . . . .	71
2.6	Summary . . . . .	72
2.7	Chapter References: . . . . .	72
<b>3.</b>	<b>Building Game Phases, Scenes &amp; Roses. . . . .</b>	<b>73</b>
3.1	Step #1 of 4: Generate a Game’s Phase . . . . .	73
	Deeper Dive: Creating Scenes using Phaser.Class . . . . .	76
	Deeper Dive: D.R.Y. Stand-alone . . . . .	76
	Deeper Dive: Scene Transitions . . . . .	78
	Deeper Dive: The CMS Game Shell . . . . .	79
	Deeper Dive: When to use the game shell model . . . . .	79
3.2	Bare Bones Prototypes . . . . .	81

	Inside a Game Phase . . . . .	83
3.3	Deeper Dive: “Phaser Essential Functions” . . . . .	84
3.4	Game Phase Prototypes . . . . .	88
	Vanilla, Strawberry, or Chocolate Creme-filled? . . . . .	88
	Deeper Dive: Overriding Scenes from Phaser.Scene . . . . .	89
	Deeper Dive: Creating Scenes using ES5 Prototypes . . . . .	89
	Deeper Dive: Creating Scenes by extending Phaser.Class . . . . .	92
	ES6 Considerations: Strawberry . . . . .	92
	Deeper Dive: Separate Scene Configuration files . . . . .	94
	Deeper Dive: Defining Other Game Properties . . . . .	95
3.5	Game Phases as Modules . . . . .	98
	Deeper Dive: ES9 Modules . . . . .	99
	“Phaser.Game” — One File to Rule them all . . . . .	100
	Main.js (aka “launch” or index.js) . . . . .	101
	Boot.js . . . . .	101
	Preload.js . . . . .	102
	Deeper Dive: Security . . . . .	103
	Deeper Dive: Cache . . . . .	103
	Deeper Dive: Loader Examples . . . . .	105
	Splash.js or Language.js? . . . . .	107
	Menu.js . . . . .	109
	Play.js . . . . .	112
	Deeper Dive: JS Modules . . . . .	112
3.6	Summary . . . . .	114
3.7	Chapter References: . . . . .	115
<b>4.</b>	<b>Building Game Prototypes, Mechanisms &amp; Tools . . . . .</b>	<b>117</b>
4.1	Task #3: Mini-Me . . . . .	118
	Creating an Avatar (the visual display) . . . . .	118
	Deeper Dive: Display selected sprite from sprite-sheet. . . . .	122
	Deeper Dive: Using Base64 Images . . . . .	123
	Creating an Avatar’s metadata . . . . .	123
4.2	Task #4: Moving Game Elements . . . . .	125
	Deeper Dive: Phaser III Input Manager . . . . .	128
	Deeper Dive: Future Proofing your source code. . . . .	130
	Deeper Dive: Configuring the Keyboard (Phaser v3.16+ updated) . . . . .	131
4.3	Task #5: Things that go bump . . . . .	132
	Walls and Camera boundaries . . . . .	132
	Interior Decoration . . . . .	134
	Deeper Dive on Game Objects hit areas. . . . .	136
	Doors, Knobs and Buttons . . . . .	137
	Deeper Dive: Writing Optimized Code . . . . .	143
	Deeper Dive: Buttons as a “Class” or “Scenes”?!!? . . . . .	143

## CONTENTS

Deeper Dive: Button size considerations . . . . .	144
Deeper Dive: Adding Buttons & Mobile Touch . . . . .	144
4.4    Task #6: When Worlds Collide . . . . .	145
4.5    Task #7: It's curtains for you . . . . .	149
4.6    Other Game Mechanism Categories . . . . .	154
4.7    The Finish Line: You're AWESOME ... Gloat, Gloat . . . . .	158
4.8    Chapter Source Code & Demo . . . . .	159
4.9    Summary . . . . .	160
4.10    Chapter References . . . . .	161
<b>5. Game Recipe™ Automation Tool . . . . .</b>	<b>163</b>
5.1    Deeper Dive: Database Protection Considerations . . . . .	164
5.2    Database Schema Construction (Copyright-able!!) . . . . .	165
Database Record Construction . . . . .	166
Database structure . . . . .	167
5.3    Remote Codebase Using AppML . . . . .	167
5.4    Building an AppML application . . . . .	169
5.5    Sample AppML codebase (Public Access) . . . . .	169
5.6    Remote codebase Using JSON . . . . .	169
5.7    Chapter Source Code & Demo . . . . .	171
5.8    Summary . . . . .	172
5.9    Chapter References . . . . .	173
<b>Part II - Mechanisms into Mechanics . . . . .</b>	<b>175</b>
<b>6. Game Mechanics &amp; Systems . . . . .</b>	<b>177</b>
6.1    Game-Play vs Game Mechanics vs Game Mechanism . . . . .	177
6.2    Game Mechanics (GM) . . . . .	179
Game Mechanics Suggested by Schell . . . . .	180
Actions Game Mechanics: . . . . .	180
Game Mechanics as: Attributes, Objects, & States . . . . .	181
Deeper Dive: Game Phases Revisited . . . . .	181
Deeper Dive: StateManager . . . . .	182
Deeper Dive: Object Manipulation in ES5/6/7/8/9 . . . . .	182
Chance Game Mechanics: . . . . .	183
Rules Game Mechanics: . . . . .	184
Deeper Dive: Rules . . . . .	185
Deeper Dive: Rule Categories . . . . .	186
"Skills" Game Mechanics: . . . . .	187
"Space" Game Mechanics: . . . . .	188
6.3    Phaser III API into Game Mechanics (GM) . . . . .	188
Deeper Dive: Input Manager Event Horizon . . . . .	191

6.4	Game Design System™ . . . . .	191
	How it works . . . . .	194
6.5	Game Genres . . . . .	196
	Deeper Dive: Game Genres . . . . .	198
	Deeper Dive: Game Modes . . . . .	198
6.6	Summary . . . . .	200
<b>7.</b>	<b>Dem's fightin' words . . . . .</b>	<b>203</b>
7.1	Launching Web Sockets . . . . .	203
7.2	Dynamic Combat Menus . . . . .	205
7.3	So, Give Me Some Space ... . . . . .	208
	Melee Weapons . . . . .	211
	Ranged Weapons . . . . .	211
7.4	OO!, OW! AH!, OW! Stayin' alive! Stayin' alive! . . . . .	215
	Grid-less Combat . . . . .	215
	Grid-ed Combat . . . . .	218
7.5	Tactical Tiled-Maps . . . . .	218
7.6	Squares and Checkered Grids . . . . .	219
	Deeper Dive: Phaser III Grids . . . . .	222
	Hexagonal Grids . . . . .	224
	Deeper Dive: Real hexagonal grids . . . . .	226
	Squishes . . . . .	226
7.7	Rules of Engagement: Take 5 paces, turn and ... . . . . .	227
	Been there ... done that ... . . . . .	227
7.8	"Where's the beef?" . . . . .	228
	Click-fest . . . . .	229
	Guitar hero - Time to get it Right! . . . . .	231
	Days of our Lives - Drama Theater . . . . .	232
	SCA Virtual "Fighter Practice" by Steve Echos . . . . .	233
	En Guard method . . . . .	236
	Yeap! Ya betcha' 'ur life! . . . . .	237
7.9	Story narrative . . . . .	237
7.10	Frisking, Fondling or Groping . . . . .	239
7.11	Chapter Source Code . . . . .	239
7.12	Complete Combat Prototypes . . . . .	240
7.13	Summary . . . . .	240
7.14	Footnotes . . . . .	241
<b>8.</b>	<b>Whazzz-sUP! .... HUD Development . . . . .</b>	<b>243</b>
8.1	HUD Housing Development . . . . .	244
8.2	HUD as Panels . . . . .	247
8.3	HUD Panels outside the Canvas?!? . . . . .	248
8.4	HUD Demos . . . . .	250

## CONTENTS

8.5	Summary . . . . .	254
8.6	Footnotes . . . . .	254
<b>9.</b>	<b>Don't make me think or "Artificial Intelligence for Dummies"</b> . . . . .	<b>255</b>
9.1	The "6 of 9" . . . . .	255
9.2	Chasing . . . . .	256
9.3	Evading . . . . .	256
9.4	Patterns . . . . .	257
9.5	Fuzzy logic . . . . .	259
9.6	Finite State Machines (FSM) . . . . .	260
	FSM Resolving Combat Outcomes . . . . .	263
	FSM Resolving AI behaviors . . . . .	264
9.7	Recursive World Feedback . . . . .	266
	Probability Data Tables . . . . .	268
9.8	Complete AI Prototypes . . . . .	269
9.9	Chapter Source Code . . . . .	269
9.10	Summary . . . . .	269
9.11	Footnotes . . . . .	270
<b>10.</b>	<b>Common Pitfalls</b> . . . . .	<b>271</b>
10.1	Lacking Debugging Tools? . . . . .	271
	Deeper Dive: Console Commands . . . . .	273
10.2	Same "Name-spaces" . . . . .	279
10.3	Callbacks . . . . .	279
10.4	Missing Documentation . . . . .	280
	Deeper Dive: What is Dragon Speak . . . . .	281

## Part III - Project Walk-through & Resources 283

<b>11.</b>	<b>Phaser III Game Prototype Library</b> . . . . .	<b>285</b>
<b>12.</b>	<b>Walk-through Tutorial Series</b> . . . . .	<b>287</b>
12.1	Difficulty Rating: Easy . . . . .	287
12.2	Difficulty Rating: Moderate or Intermediate . . . . .	287
12.3	Difficulty Rating: Advanced (aka "The Full Monty!") . . . . .	287
12.4	Source Code is here (online) . . . . .	290

## Part IV - Next Steps ... Distribution! 291

<b>13.</b>	<b>Distribution Preparation: Your Game Product</b> . . . . .	<b>295</b>
13.1	Development vs. Production . . . . .	295
13.2	Create A Game Pipeline . . . . .	296

13.3	Preparing for Mobile Deployment . . . . .	297
13.4	Chapter References: . . . . .	300
<b>14.</b>	<b>Marketing Channels Deployment . . . . .</b>	<b>301</b>
14.1	Channel Selection . . . . .	301
14.2	What do I need? . . . . .	302
14.3	Targeting Markets with the “Tower of Babel” . . . . .	304
14.4	Channel Preparations . . . . .	306
14.5	Generating a Profit . . . . .	308
	In-Game Purchases . . . . .	309
	Advertising . . . . .	310
	Partnerships & Sponsors . . . . .	311
	Retail . . . . .	312
	Billing . . . . .	313
	Data . . . . .	313
	Player Interactions . . . . .	313
	Paraphernalia Merchandising . . . . .	313
14.6	Chapter Reference . . . . .	314
	<b>Appendix . . . . .</b>	<b>315</b>



# Distribution Permission

This is the 4th edition 9 MAR 2019.  
Originally released edition 5 AUG 2016.

**This book is distributed through LeanPub and Amazon with author's permission.** All copyrights are reserved under the Pan-American and International Copyright Conventions. You may not reproduce this book, in whole or in part, in any form or by any means electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system now known or hereafter invented, without written permission from the author.

***NOTE: Amazon Book Editions "Bonus Download Content" is available from the book's website: <http://makingbrowsergames.com/book/>*** You will find errata information, source code and software updates. Thank you for your patronage.

## Viewing this e-Book

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



# Disclaimer

All the information contained within is for the convenience of its reader. All information is accurate as can be reasonably verified at the time of original publication. However, content suggested may not reflect current industry recommendations after original publication date for **ECMA-262**<sup>1</sup> (also known as — aka — “JavaScript”, ES5, ES6, ES7, ES8 or ES9), **Phaser Game Framework versions 2.6.2, Community Editions (CE) or Phaser III (as v3.16+ only)**. There are no guarantees nor warranties stated or implied by the distribution of this information. Using the information in this document 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.

This book’s intent is **not** to teach HTML5<sup>2</sup>, CSS<sup>3</sup> nor JavaScript fundamentals<sup>4</sup>, game design best practices of game design, nor software encoding paradigms. Its content provides simple-to-follow worksheets, step-by-step instructions, and a straightforward, **yet innovative approach**, in building **Massive Multi-Player online Game(s) (MMGs)** from **component prototypes** using my **Game Design System™**<sup>5</sup> method. This process and source code are merely one way, and hence do not claim to be the best nor most efficient way of implementing this game mode, mechanisms, and mechanics. **Stephen Gose LLC reserves the right, at any time and without notice, to change modify or correct the information contained in this publication.**

<sup>1</sup><http://www.ecma-international.org/publications/standards/Ecma-262.htm>

<sup>2</sup><http://amzn.to/2nAYjxr>

<sup>3</sup><http://amzn.to/2mG01Zv>

<sup>4</sup><http://amzn.to/2nAYA34>

<sup>5</sup><http://www.businessdictionary.com/definition/system.html>



**Warning:** The Phaser newsletter dated 21 September, 2018 includes projected development on Phaser III. In August 2017, many features in pre-Phaser 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 these 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. My best guess is that ***any and all books, tutorials and "how to" articles — written prior to Phaser v3.16.0 (NOV 2018) — are not fully functional with Phaser III (as v3.16+) and should be re-written to the Phaser v3.16.0 minimum standard baseline. Hence the reason this book is dedicated and updated to the official Phaser III (release v3.16.x) and has removed any references to previously released versions prior to v3.16+ (See newsletter #139 dated 20190211) "Breaking Changes"***<sup>6</sup>

---

<sup>6</sup><https://madmimi.com/p/f0b3bd>

# 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<sup>7</sup>, — **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.”
- **by Christer Kaitila: *The McFunky-pants Method*** — “Challenge yourself to create a code-base 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. No-art prototypes also have one other major advantage:** in previous games, I would make beautiful mock-ups 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<sup>8</sup> before coding;** these days I know that the tech specs and evolving game-play mechanics will mean that much of what you make at the start won’t make it into the finished game.”

---

<sup>7</sup>[http://en.wikipedia.org/wiki/Art\\_forgery](http://en.wikipedia.org/wiki/Art_forgery)

<sup>8</sup><https://www.gamedevmarket.net/?ally=GVgAVs0j>



# About this Workbook:

This 4th edition offers new development tools and production methods I call the ***Game Design System***™ in which you create ***Game Recipes***™.<sup>9</sup> Expert game developers already understand the “***Don’t Repeat Yourself***” (**D.R.Y.**) concept, yet few have taken a step back to the “10,000 foot view” on their game production pipelines. We’ll do that aerial view in this book as we follow my newly proposed game development process for prototypes, mechanics and artwork integration. Along with our game construction, we’ll build tools to automate our pipeline. I believe you will be surprised how quickly and easily we build games using the ***Game Design System***™ with its ***Game Recipes***™ **tools**. Although this workbook is intended to be a hands-on guide to HTML5 game development with emphasis on the ***Phaser JavaScript Gaming Frameworks***. Yet, our project management applies **to any popular game development framework**.

Most of the content in this book is actually **not inside these pages**. The Internet is a living, dynamic resource of information that doubles every 35 days! I explain why this book refers to external resources in the “Book Formatting” Section below. You’ll find the external content in the footnotes, as reference links, and in the **Bonus Content Downloaded files** (which are available from the **supporting website**<sup>10</sup> or from your **LeanPub.com personal library**).

Therefore, if you purchased this book from:

- **LeanPub.com**, you will have a continually updated edition in your personal library.
- **Amazon.com, you may access the latest information from our supporting website**

**<http://makingbrowergames.com/book/>**

All the source code is written in pure JavaScript or ***Phaser v3.16+ Framework***; it doesn’t use any additional “abstraction layers” such as TypeScript, CoffeeScript, or JQuery. JSLinking, minification/compression and obfuscation/security topics are discussed in the Marketing and Distribution Chapter.

---

<sup>9</sup><https://www.nateliason.com/blog/become-expert-dreyfus>

<sup>10</sup><http://makingbrowergames.com/book/>

I've gone to great lengths<sup>11</sup> to make this book "skim-friendly" — ***even for my International customers.*** I have provided links to "***English (American) Jargon phrases***"<sup>12</sup> that will help translate this content directly into your native language. The entire new 4th edition has more screen shots, step-by-step worksheets, thoroughly annotated source code listings and diagrams. 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 assume that many readers will want to use this book as a reference (as demonstrated from the previous three editions) as well as a tutorial workbook. So, I've included **references to other games, gaming engines/frameworks, indie developers, authors, their open source contributions, their articles and books, artwork**<sup>13</sup>, **applications tools and their wisdom.** If you would like your product, articles, or community contributions included in future updates, please use my contact information on LeanPub.com<sup>14</sup> or Amazon.com Author's page<sup>15</sup>

<sup>11</sup><https://dictionary.cambridge.org/us/dictionary/english/go-to-great-lengths>

<sup>12</sup><https://www.smartling.com/blog/40-american-slang-words-and-phrases-you-need-to-know/>

<sup>13</sup><https://www.gamedevmarket.net/?ally=GVgAVs0j>

<sup>14</sup><https://leanpub.com/u/pbmcube>

<sup>15</sup>[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](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)

# Workbook Content:

This ***abridged book***<sup>16</sup> is a hands-on guide to HTML5 game development with an emphasis on the ***Phaser III JavaScript Gaming Frameworks***. By following my steps toward developing a complete system of games, you will translate my process into your own ***bespoke***<sup>17</sup> game design product(s) in a matter of hours! I will explain my rationale behind each decision so that you can adapt my methods into your products' pipeline in a similar manner. When you finish this workbook and all the exercises, **unlike every book on the market today, you should have a working copy of your own game design — not just a “replica/copy” of my game(s)!** Here are the main topics we cover in the **4th Edition's NEW content**:

- Game Generation Tools construction for “***Game Recipes™***”.
- Game Category alignment with App Stores and the gaming industry.
- **Distribution and Marketing:** who, where, and how to deploy finished game products with supporting data from current marketing analysis.
- **Coding Style Appendix:** Migration away from classical OOP to OLOO compositions (A section for interested Senior Programming Engineers and the rationale on why and how.)
- **“The Deeper Dive:”** dedicated sections for interested Software Engineers concerning Phaser v3.x.x.
- The ***Game Design System™*** development method for Game Project Management.

Here are the main topics we will cover:

**Part I. Introduction** (Chapters 1 to 5): sets the ground work for a new game studio, workstation, and game project(s) structures. ***Seasoned programmers may wish to skip chapter 1 entirely,*** and merely skim over the **Coding Style Appendix**.

- Chapter 1: (New Comers): Game studio & Project setups.
- Chapter 2: Building Game Prototypes & Mechanisms.
- Chapter 3: Building Game Phases, States & ***Roses***.
- Chapter 4: Building Game Prototypes & Tools.
- Chapter 5: “***Game Recipes™***” Automation Tools.

---

<sup>16</sup><https://en.wikipedia.org/wiki/Abridgement>

<sup>17</sup><https://www.urbandictionary.com/define.php?term=bespoke>

**Part II. Binding Mechanisms into Mechanics** (Chapters 6 to 9): **reveals an innovative approach** to game production pipelines based on the ***Game Design System™***.

- Chapter 6: Game Mechanics & Systems.
- Chapter 7: Conflict Systems.
- Chapter 8: Heads Up Display (HUD) Systems.
- Chapter 9: AI Systems.
- Chapter 10: Common Pitfalls

**Part III. Walk-through Examples** provides several ***FREE game licenses (some are valued at \$48 each***<sup>18</sup>). we'll take our crafted "Part I game components" and apply "Part II game mechanic logic/rule systems." I demonstrate "separation of concerns", and my method for rapid game generation using my unique "***Game Recipes™" inside the Game Design System™***.

**Part IV. Next Steps ... Distribution and Marketing Preparation.** Topics include preparing a game product for various distribution or marketing channels, pricing, and international marketing efforts.

**Part V. Appendices, Resources and Supporting Website** available online at **<http://makingbrowergames.com/book/>**

**Bonus Content:** Phaser Game Development Certification Courses available here  
**<https://leanpub.com/bookstore/type/course>**

---

<sup>18</sup><http://shop.pbmcube.net/>

# Book formatting:

**Editor's Note on Abridged format:<sup>19</sup> "Why have separate external links and files?", you say?**

- **Firstly,** the WWW is a dynamic place with new content and methods appearing daily. What was once the “cutting edge” becomes deprecated and obsolete in a matter of a few years or even months! (**for example, using “window.onload” initially mentioned in 2004 and now recanted/retracted by the original author.**)
- References linking to external concepts preserve this workbook’s integrity and **“freshness”** (and thereby your investment), avoids copyright infringements, and **permits referenced authors to recant or update their own recommendations (as in the case of “using window.onload” and especially in the case pre-Phaser v3.16.x JavaScript Gaming Framework frequent updates).**
- You are purchasing my months of research, investigations, synthesis, and experiments derived from my investigations. I have **“trail-blazed”<sup>20</sup>** the path for other game pioneers to use. It saves you the time of doing it yourself.
- In general, **This book’s main discussions show you how things work, whereas the aside call-outs, footnotes, and reference links delve into why things work as they do.**
- It provides valuable information from other notable game industry experts. My research places the best current recommendation in this workbook, and filters out the **“cruft”<sup>21</sup>**. **If you follow all my reference links, you will have a greater store of knowledge and develop a better understanding about my method concepts since we’ll share the same common knowledge foundation.**
- It shortens my book by hundreds of pages and lowers my publishing costs which then, in turn, lowers your purchase price. By keeping the associated source code separated into a Bonus Content areas on the supporting website, it allows you to focus on and compare this content side-by-side with the source code. This permits you to read this workbook without interruptions — unlike many technical books to date. **Now, Admit it, Honestly!** Don’t you just hate reading miles of **UN-commented, UN-documented source code** only to arrive finally, dozens of pages later, at the text explaining what you just studied? Or some authors place their pages of explanation in front of their **UN-commented, UN-documented**

---

<sup>19</sup><https://en.wikipedia.org/wiki/Abridgement>

<sup>20</sup><https://en.wiktionary.org/wiki/trailblaze#English>

<sup>21</sup><https://en.wikipedia.org/wiki/Cruft>

**source code.** Neither methods are satisfactory in my opinion. I would rather have the text explanations and source code side-by-side. **Wouldn't you?**

- It further allows me to update my source code and supplements without republication. This is a great benefit for both of us concerning Phaser III rapid release trends since January 2017.

# Who This Book Is For:

This workbook is aimed at both ***hands-on leaner***<sup>22</sup> — those who enjoy “learning by experimentation”, and at ***experienced expert programmers***<sup>23</sup> in web application development; and, of course, those who want a finished game by their own designs and efforts. If you are interested in making browser games, ***especially for the mobile market***, then this book is a perfect choice along with its companion volumes for ***Phaser v2.x.x or III***:<sup>24</sup> (books are listed in the upcoming Reference Section below). With this in mind, you will do a lot of writing, thinking, and coding in both HTML5 and JavaScript in this workbook. You may prefer using paper (external physical- or soft-“paper”) to organize your development ideas and processes.

This ***abridged workbook***<sup>25</sup> is a tutorial guide into the ***Game Design System™*** method using HTML5<sup>26</sup>, CSS3<sup>27</sup> and JavaScript<sup>28</sup> technologies with ***an emphasis on either Phaser III***. I know that many *senior software developers* already have these technologies in their arsenal; but, I have received dozen of email complaints about this book’s former editions as being “... too difficult for those just starting their own game studios.” Therefore, ***if learning any of these mentioned technologies is what you are initially seeking***, then I recommend a quick visit to ***W3Schools***<sup>29</sup> as your ***first FREE starting point***. By following their instructions, you will learn a complete foundation in HTML, 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.***

## Your newly obtained skills...

By the end of this workbook, you’ll have integrated into your own ***bespoke***<sup>30</sup> game design:

- Organized file structure for rapid game development;

---

<sup>22</sup><https://www.nap.edu/read/9853/chapter/5>

<sup>23</sup><https://www.nateliason.com/blog/become-expert-dreyfus>

<sup>24</sup><https://leanpub.com/u/pbmcube>

<sup>25</sup><https://en.wikipedia.org/wiki/Abridgement>

<sup>26</sup><http://amzn.to/2nAYjxr>

<sup>27</sup><http://amzn.to/2mG01Zv>

<sup>28</sup><https://amzn.to/2lw9RZj>

<sup>29</sup><http://www.w3schools.com/js/default.asp>

<sup>30</sup><https://www.urbandictionary.com/define.php?term=bespoke>

- Created “**Game Recipes™**” to scaffold further game projects;
- Built an **automation tool** for your game studio production pipeline.
- Imported various **game resources and artwork assets**:<sup>31</sup>
- Displayed, animated and moved game avatars on various game phases;
- Managed groups, containers and they’re pros and cons;
- Deployed heads-up displays (HUD) both “inside and out from” the `canvas` tag;
- Used customized web fonts;
- Incorporated multiple game-inputs;
- Implemented several physics systems on various components;
- Defined and Included “game juice”;
- Created and managed game phases properly using D.R.Y.;
- Managed permanent game assets across various game phases;
- Optimized your game for various mobile devices using SWPA, PWA and AMP;
- Integrated available 3rd-party plugins.
- Deployed single—player games from the workbooks step-by-step tutorials.
- Demystified Web Sockets for optimized game deployments.
- Tested multi-player game development from single-player foundation.

---

<sup>31</sup><https://www.gamedevmarket.net/?ally=GVgAVsoJ>

# Bonus Content (4th Edition)

***Included with your purchase!***

- Free Affiliate Guide,
- Bonus Content Download package.
- Dynamically updated references, footnotes and supplements.
- Supporting Website access:

***<http://makingbrowergames.com/p3gskc/>***

***OR***

***<http://makingbrowergames.com/p3gp-book/>***

## Game Design System™ Recipes:

***"Game Recipes™"*** (purchased separately) on specific gaming mechanics using the ***Game Design System™ method*** and concepts (available from ***Amazon.com***<sup>32</sup> or earn your ***Game Development Certifications*** from LeanPub.com courses or ***Zenva Game Academy***<sup>33</sup>):

- ***Ultimate Game Studio Starter Kit Collection Certification Tutorials (LeanPub.com)***<sup>34</sup> ***or (Amazon.com)***<sup>35</sup>
- Action & Arcade (Coming Soon)
- ***Adventure, Mazes & RPG (Amazon.com)***<sup>36</sup>
- ***Collapsing Blocks (Amazon.com)***<sup>37</sup>
- Connect 4 & Go (Coming Soon)
- ***Dating Simulations & Quizzes — Certification Tutorial (LeanPub.com)***<sup>38</sup> ***or (Amazon.com)***<sup>39</sup>

<sup>32</sup>[https://www.amazon.com/gp/search/ref=as\\_li\\_qf\\_sp\\_sr\\_tl?ie=UTF8&tag=pbmcube-20&keywords=stephen%20gose&index=aps&camp=1789&creative=9325&linkCode=ur2&linkId=35a07e5471ef728a49ce48b6009957ca](https://www.amazon.com/gp/search/ref=as_li_qf_sp_sr_tl?ie=UTF8&tag=pbmcube-20&keywords=stephen%20gose&index=aps&camp=1789&creative=9325&linkCode=ur2&linkId=35a07e5471ef728a49ce48b6009957ca)

<sup>33</sup><https://academy.zenva.com/?a=47&campaign=PhaserGamePrototyping>

<sup>34</sup><https://leanpub.com/set/leanpub/ugsskitc>

<sup>35</sup><https://amzn.to/2qQsq3z>

<sup>36</sup><https://amzn.to/2vXickO>

<sup>37</sup><https://amzn.to/2PnijUX>

<sup>38</sup><https://leanpub.com/c/mbg-dating>

<sup>39</sup><https://amzn.to/2nP0fjF>

- Defensive Towers (Coming Soon)
- ***Dress Up & Fashion — Certification Tutorial (LeanPub.com)***<sup>40</sup> or (***Amazon.com***)<sup>41</sup>
- Hidden Objects (Coming Soon)
- ***"Jump to Capture" — Certification Tutorial (LeanPub.com)***<sup>42</sup> or (***Amazon.com***)<sup>43</sup>
- MahJong (Coming Soon)
- ***Match 3 (Amazon.com)***<sup>44</sup>
- ***Memory Match (both pairs & sequence) (Amazon.com)***<sup>45</sup>
- Music & Rhythm (Coming Soon)
- ***Puzzle (Jigsaw & sliders) — Certification Tutorial (LeanPub.com)***<sup>46</sup> or (***Amazon.com***)<sup>47</sup>
- ***Role Playing Games Certification Tutorial (LeanPub.com)***<sup>48</sup> or (***Amazon.com***)<sup>49</sup>
- Strategy (Coming Soon)



**Hint:** Also available on itch.io<sup>50</sup>, or Google Play<sup>51</sup> or ***Amazon.com*** search for "Stephen Gose"<sup>52</sup>

## Our References:

- ***Phaser III Game Starter Kit Collection***<sup>53</sup> 16+ Classic Game mechanics for Phaser v3.16.x Framework using the ***Game Design System™*** methodology.
- ***Phaser III Game Design Workbook***<sup>54</sup> updated edition dedicated to Phaser III and follows the ***Game Design System™*** methodology.
- ***Phaser III Game Prototyping***<sup>55</sup> is a sister companion to this book for the new Phaser III API. All examples are translated in Phaser III and takes advantage of all the new capabilities.

<sup>40</sup><https://leanpub.com/c/mbg-dressup-p2>

<sup>41</sup><https://amzn.to/2MDvEUi>

<sup>42</sup><https://leanpub.com/c/mbg-peg-p2>

<sup>43</sup><https://amzn.to/2vWuqdd>

<sup>44</sup><https://amzn.to/2MmwTld>

<sup>45</sup><https://amzn.to/2OK8JnI>

<sup>46</sup><https://leanpub.com/c/mbg-puzzle-p2>

<sup>47</sup><https://amzn.to/2vV63Na>

<sup>48</sup><https://leanpub.com/c/mbg-rpg-p2>

<sup>49</sup><https://amzn.to/2DC0OYn>

<sup>50</sup><https://pbmcube.itch.io/making-browser-games-starter-kit-collection-volume-1>

<sup>51</sup>[https://play.google.com/store/books/details/Stephen\\_Gose\\_Phaser\\_Game\\_Prototyping?id=JUFJDwAAQBAJ](https://play.google.com/store/books/details/Stephen_Gose_Phaser_Game_Prototyping?id=JUFJDwAAQBAJ)

<sup>52</sup>[https://www.amazon.com/gp/search/ref=as\\_li\\_qf\\_sp\\_sr\\_tl?ie=UTF8&tag=pbmcube-20&keywords=stephen%20gose&index=aps&camp=1789&creative=9325&linkCode=ur2&linkId=35a07e5471ef728a49ce48b6009957ca](https://www.amazon.com/gp/search/ref=as_li_qf_sp_sr_tl?ie=UTF8&tag=pbmcube-20&keywords=stephen%20gose&index=aps&camp=1789&creative=9325&linkCode=ur2&linkId=35a07e5471ef728a49ce48b6009957ca)

<sup>53</sup><https://leanpub.com/p3gskc>

<sup>54</sup><https://leanpub.com/phaser3gamedesignworkbook>

<sup>55</sup><https://leanpub.com/phaser3gameprototyping>

- **Phaser Game Design Workbook**<sup>56</sup> in the newly revised and expanded 4th Edition.
- **3rd Edition remains available**<sup>57</sup> in paper-back or Kindle on **Amazon.com**<sup>58</sup> or from **LeanPub.com**<sup>59</sup>
- **Kiwi Game Design Workbook**<sup>60</sup> the grandfather of Phaser v2.x.x
- **Phaser Game Starter Kit Collection**<sup>61</sup> 16+ Classic Game mechanics recipes for Phaser v2.x.x Framework using the **Game Design System™** methodology.
- **Phaser Multi-player Gaming Systems**<sup>62</sup> A **Massive Multi-Player Online Game Design Workbook** for HTML5, Phaser and other popular Mobile Gaming Frameworks.
- **Phaser Game Prototyping**<sup>63</sup> in its new 4th edition.
- **Olympic Game Design System™**<sup>64</sup> similar to this Phaser Game Design Workbooks but uses other popular **HTML5 and Unity Gaming Frameworks**.
- **Multi-player Gaming Systems**<sup>65</sup> A **Massive Multi-Player Online Game Design Workbook** for HTML5 and other JS Game engines.



**Hint:** Individual chapters, using the **Game Design System™ mechanics** are available from Amazon.com by searching for "**Stephen Gose Phaser**".<sup>66</sup>

## Tweet This Book!

Please help me spread the word about this book on Twitter!<sup>67</sup>

The suggested hashtag for this book is #PBMCube<sup>68</sup>.

---

<sup>56</sup><https://leanpub.com/phaserjsgamedesignworkbook>

<sup>57</sup><https://phaser.io/news/2016/11/phaser-game-design-workbook>

<sup>58</sup><https://www.amazon.com/Phaser-js-Game-Design-Workbook-development/dp/1520154550>

<sup>59</sup><https://leanpub.com/phaserjsgamedesignworkbook>

<sup>60</sup><https://leanpub.com/kiwigamedesignworkbook>

<sup>61</sup><https://leanpub.com/pgskc>

<sup>62</sup><https://leanpub.com/rrgamingssystem>

<sup>63</sup><https://leanpub.com/LoRD>

<sup>64</sup><https://leanpub.com/gds>

<sup>65</sup><https://leanpub.com/rrgamingssystem>

<sup>66</sup>[https://www.amazon.com/s/ref=nb\\_sb\\_noss\\_2?url=search-alias%3Daps&field-keywords=stephen+gose](https://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Daps&field-keywords=stephen+gose)

<sup>67</sup><http://twitter.com/pbmcube>

<sup>68</sup><https://twitter.com/search?q=%23PBMCube>



# Book Examples & Samples

## *Table of Content:*

[\*\*http://makingbrowsergames.com/book/\\_p3demos/game.js\*\*](http://makingbrowsergames.com/book/_p3demos/game.js)

[\*\*http://makingbrowsergames.com/book/\\_p3demos/game-noNameSpace.js\*\*](http://makingbrowsergames.com/book/_p3demos/game-noNameSpace.js)

[\*\*http://makingbrowsergames.com/book/\\_p3demos/Ch5-game.js\*\*](http://makingbrowsergames.com/book/_p3demos/Ch5-game.js)

—  
Example: 1.1 Creating Namespace for games<sup>69</sup>

—  
Example: 2.1 Prototyping Graphics<sup>70</sup>

Example: 2.2 Launching a Game<sup>71</sup>

—  
Example: 3.1 Creating Game Phase (traditional method)<sup>72</sup>

Example: 3.1a Creating Game Phase Continued - Game.js (traditional method)<sup>73</sup>

Example: 3.4 Additional Phaser v2 Properties<sup>74</sup>

—  
Example: 4.1: Prototyping a Visual Avatars<sup>75</sup>

Example: 4.2: Prototyping Movement Properties in v2<sup>76</sup>

Example: 4.3: Movement Arrows Integration<sup>77</sup>

Example: 4.4: World Boundaries Grouping<sup>78</sup>

Example: 4.5: World Boundaries Integration<sup>79</sup>

Example: 4.6: Interior Boundaries Integration<sup>80</sup>

Example: 4.7: Collision Detection Integration<sup>81</sup>

Example: 4.8: Collision Results Determination<sup>82</sup>

---

<sup>69</sup>[http://makingbrowsergames.com/book/\\_p3demos/\\_index.html](http://makingbrowsergames.com/book/_p3demos/_index.html)

<sup>70</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>71</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>72</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>73</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>74</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>75</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>76</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>77</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>78</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>79</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>80</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>81</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

<sup>82</sup>[http://makingbrowsergames.com/book/\\_p3demos/game.js](http://makingbrowsergames.com/book/_p3demos/game.js)

Example: 4.9: New Game Over State<sup>83</sup>

Example: 4.10: Elementary HUD Creation<sup>84</sup>

Example: 4.11: Collecting User Input<sup>85</sup>

Example: 4.12: Responding to User Input<sup>86</sup>

—

Example: 7.1: Launching Web Sockets<sup>87</sup>

Example: 7.2: Dynamic Combat Menus<sup>88</sup>

Example: 7.3: Dynamic Combat Menus supporting function<sup>89</sup>

Sample: Projectile Template<sup>90</sup>

Example: 7.4: Grid-ed Combat<sup>91</sup>

Example: 7.5: Grid-less Combat Encounter<sup>92</sup>

Example: 7.6: Grid-ed Combat<sup>93</sup>

Example: 7.7: Grid-ed Combat Squares<sup>94</sup>

Sample: Combat Finite State Machine

Sample: SCAVT game: lines 292-318

—

Sample: 8.1: Prototyping a HUD

Sample: 8.2: Heads Up Display Plugin

Example: 8.3: HUD Menu Grouping

—

Sample: 9.1: Combat Pseudo Code

Sample: 9.2: Combat Pseudo Code

Example: 9.3: Enemy mirrored movement

Sample: 9.4: Combat Pseudo Code

Sample: 9.5: New Combat States Module Added

---

<sup>83</sup>[http://makingbrowergames.com/book/\\_p3demos/game.js](http://makingbrowergames.com/book/_p3demos/game.js)

<sup>84</sup>[http://makingbrowergames.com/book/\\_p3demos/game.js](http://makingbrowergames.com/book/_p3demos/game.js)

<sup>85</sup>[http://makingbrowergames.com/book/\\_p3demos/game.js](http://makingbrowergames.com/book/_p3demos/game.js)

<sup>86</sup>[http://makingbrowergames.com/book/\\_p3demos/game.js](http://makingbrowergames.com/book/_p3demos/game.js)

<sup>87</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson01.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson01.html)

<sup>88</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson02.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson02.html)

<sup>89</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson03.html)

<sup>90</sup><http://labs.phaser.io/index.html?dir=games/topdownShooter/&q=>

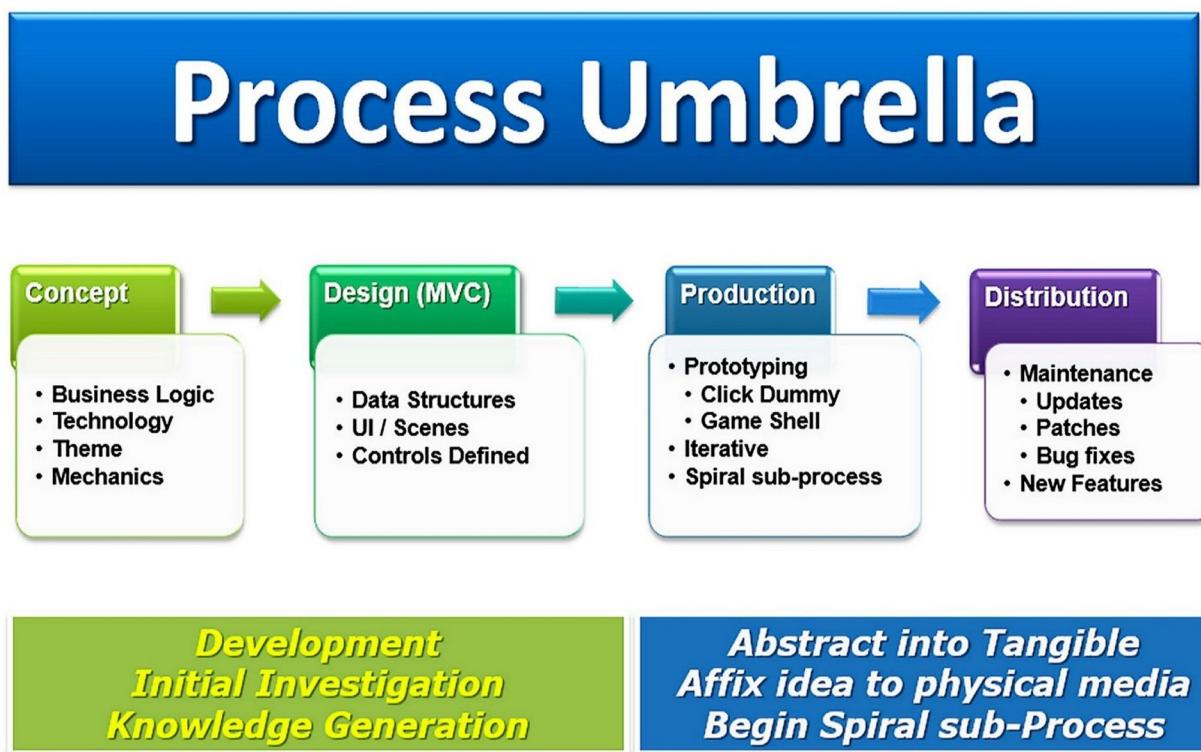
<sup>91</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson03.html)

<sup>92</sup>[http://makingbrowergames.com/p3gp-book/\\_p3-bloodPitv1/](http://makingbrowergames.com/p3gp-book/_p3-bloodPitv1/)

<sup>93</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson03.html)

<sup>94</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch7-examples/lesson03.html)

# Part I - Concept & Design



Part I Chapters 1 to 5 is an introduction to my **Game Design System™** and building **Game Recipes™**. By the end of "Part I" we will have created everything a game prototype uses:

- Creating visual avatars and their associated data structures,
- Collecting players' input,
- Detecting collisions and Interactions among various entities,

- Migrating to various menus, and Scenes,
- Monitoring the gaming loop, and
- Creating and Updating heads-up displays (aka HUD).
- Tweens
- Events
- Sound effects (sfx), audio and
- Sprite Animation(s)

## ***Introduction***

“Why study systems-based design?”, you say? The earliest decisions about what kind of games a studio will build impacts the following development and production activities of a game’s project. It affects how the programmers encode features, how the designers construct levels and optimize game mechanics, and how the time-consuming animations and cutscenes 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 v3.16.x JavaScript Game Framework (or with any Gaming Framework for that matter)***; and, at the same time, it’s fun! However, dealing with all those technical details such as web pages, artwork, collisions, sprites, game phases (and there’s more!) can be intimidating — especially, if this is your ***first experience with such components.***<sup>95</sup>

***JavaScript Gaming Frameworks***, in general, are resource libraries that run inside your browser or mobile device. Using them, you can quickly build various two dimensional (2D, 2.5D<sup>96</sup> and even ***3D!!***<sup>97</sup>) games inside a simple HTML5 canvas tag. The ***Phaser v3.16.x JavaScript Gaming Frameworks***<sup>98</sup> 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***<sup>99</sup>. So, let’s begin by creating a simple game prototypes and mechanisms while exploring many of the basic concepts found in ***Game Design System™ method.***

In Part I, our goal is building a fully functional Phaser v3.16.x Game Prototype. There are simple step-by-step worksheets for each task we plan to do. We’ll catalog and

<sup>95</sup>[http://www.michael-richardson.com/processes/rup\\_for\\_sqa/core.base\\_rup/guidances/supportingmaterials/use\\_component\\_architectures\\_CBC2F6B5.html](http://www.michael-richardson.com/processes/rup_for_sqa/core.base_rup/guidances/supportingmaterials/use_component_architectures_CBC2F6B5.html)

<sup>96</sup><https://en.wikipedia.org/wiki/2.5D>

<sup>97</sup><https://en.wikipedia.org/wiki/Three.js>

<sup>98</sup><http://phaser.io>

<sup>99</sup><http://www.w3schools.com>

create various ***game component mechanism***.<sup>100</sup> From this simple foundation, we can combine them (aka widgets or components or code-blocks) into various **Game Mechanics** and Genres as easily as a child would use "Lego"™ blocks to construct a toy castle. Furthermore, you can review this construction process throughout ***Phaser III Game Starter Kit Collections***.<sup>101</sup> We'll study Game Mechanics coming up in Part II.

---

<sup>100</sup>[http://www.michael-richardson.com/processes/rup\\_for\\_sqa/core.base\\_rup/guidances/concepts/component\\_A2E2B3B1.html](http://www.michael-richardson.com/processes/rup_for_sqa/core.base_rup/guidances/concepts/component_A2E2B3B1.html)

<sup>101</sup><https://leanpub.com/p3gskc>



# 1. Game Studio & Project Setups

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



**Hint:** Google and Mozilla both provide excellent resources for **Game**<sup>1</sup> and **Web Developers**.<sup>2</sup>

The first impressions you'll develop while reading this chapter is: **THERE 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)**<sup>3</sup>. 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.**"<sup>4</sup>

## 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 **"preparing to start the damn game"** (ed.: **and "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.

<sup>1</sup><https://developer.mozilla.org/en-US/docs/Games>

<sup>2</sup><https://developers.google.com/web/tools/>

<sup>3</sup>[https://en.wikipedia.org/wiki/Ad\\_nauseam](https://en.wikipedia.org/wiki/Ad_nauseam)

<sup>4</sup><https://medium.freecodecamp.org/making-sense-of-front-end-build-tools-3a1b3a87043b>

Which brings me to the next tip..." Read more such advice [here ...<sup>a</sup>](#)

<sup>a</sup><http://makegames.tumblr.com/post/1136623767/finishing-a-game>

**Well, ... Ok then,** let's get our game started and finished; then, we'll address those "**Front-end Building Tools**".<sup>5</sup>

## 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**<sup>6</sup>; 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 (folder) to save and review each game projects' development files. (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 . . .)

### 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**.<sup>7</sup> It's all about browser security and the **same-origin policies**<sup>8</sup> — prohibiting files loading from different "**domains**"<sup>9</sup> and the protocols used to access your locally stored files. When you request anything from the Internet you're typically using the "hyper-text 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**

<sup>5</sup><https://developers.google.com/web/tools/setup/setup-buildtools>

<sup>6</sup><https://caniuse.com/#search=html5>

<sup>7</sup><http://gose-internet-services.net/data-centers/uk-data-center/>

<sup>8</sup>[https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\\_policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)

<sup>9</sup>[https://en.wikipedia.org/wiki/Domain\\_Master\\_Browser](https://en.wikipedia.org/wiki/Domain_Master_Browser)

**a different protocol**<sup>10</sup> 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**<sup>11</sup> — 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)**<sup>12</sup> per **RFC 990, November 1986.**<sup>13</sup> 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**. (See Appendix — Security)



**Exercise:** Read more about browser security from the **Chromium Blog**<sup>14</sup>



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

## Deeper Dive: Testing MMoGs Locally

Wikipedia states<sup>a</sup>, “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

<sup>10</sup><https://www.w3.org/Addressing/>

<sup>11</sup><https://en.wikipedia.org/wiki/Localhost>

<sup>12</sup><https://www.lifewire.com/network-computer-special-ip-address-818385>

<sup>13</sup><http://tools.ietf.org/html/rfc990#page-6>

<sup>14</sup><https://blog.chromium.org/2008/12/security-in-depth-local-web-pages.html>

be modified to provide appropriate aliases for each such address.”

<sup>a</sup><https://en.wikipedia.org/wiki/Localhost>

“So then! What’s a girl to do?!” The solution is simple. Run your game development files **from a local web server or remotely from the Internet**<sup>15</sup>. Depending on your workstation operating system (and what you have installed already), there are several ways to launch web pages from a “web service”<sup>16</sup>.

- Some text editors and Integrated Development Environments (**IDE**) **already include a local web server.** **Brackets**<sup>17</sup> is one such example.
- Another simple solution I discovered is the **Google’s Chrome Web Server.**<sup>18</sup> 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.**<sup>19</sup>
- The official Phaser instructions (and sanctioned method) are:

**Quote:** “We would recommend either **WAMP Server**<sup>a</sup> or **XAMPP**<sup>b</sup> 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.<sup>c</sup> (**overlooked was MAMP or MAMP Pro**<sup>d</sup> **available for those with MAC OS X.**)

<sup>a</sup><http://www.wampserver.com/en/>

<sup>b</sup><http://www.apachefriends.org/en/xampp.html>

<sup>c</sup><https://phaser.io/tutorials/getting-started-phaser3/part2>

<sup>d</sup><https://www.mamp.info/en/>

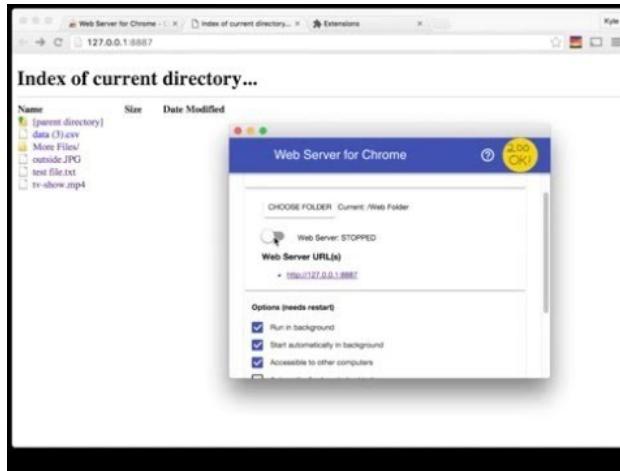
<sup>15</sup><http://gose-internet-services.net/data-centers/uk-data-center/>

<sup>16</sup>[http://www.webopedia.com/TERM/W/Web\\_Services.html](http://www.webopedia.com/TERM/W/Web_Services.html)

<sup>17</sup><http://brackets.io/>

<sup>18</sup><https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en>

<sup>19</sup><http://uri.thephpleague.com/4.0/components/overview/>



View this Video at <https://www.youtube.com/watch?v=AK6swHiPtew><sup>20</sup>.

**Google Chrome Web Server Install Instructions (movie: 1:51 minutes)**

## Development Tools

Although Phaser web site has a well-documented section on **how to get started**.<sup>21</sup> I recommend you test and develop your game with a **source code editor**<sup>22</sup> or an **Integrated Development Environment (IDE)**<sup>23</sup> editor of your choice. (**See examples here**)<sup>24</sup> I use several: **Jetbrain's WebStorm**<sup>25</sup>, **Phaser Editor v2 for Phaser III**<sup>26</sup>, and **NotePad++ (FREE)**<sup>27</sup>; although, **Brackets**<sup>28</sup> editor runs a close second in my daily website construction. What's everyone else doing? **OVER one-third of all professional web developers use this IDE.**<sup>29</sup> Another fourth of all professional web developers use **TypeScript**<sup>30</sup> found inside the **Microsoft Visual Studio**.<sup>31</sup>

**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 into the source code that will lead to problems. If you do not have a favorite text editor, I have some recommendations based on your status:

<sup>21</sup><https://phaser.io/tutorials/getting-started-phaser3>

<sup>22</sup>[https://en.wikipedia.org/wiki/Source\\_code\\_editor](https://en.wikipedia.org/wiki/Source_code_editor)

<sup>23</sup>[https://en.wikipedia.org/wiki/Integrated\\_development\\_environment](https://en.wikipedia.org/wiki/Integrated_development_environment)

<sup>24</sup>[https://en.wikipedia.org/wiki/Source\\_code\\_editor#Notable\\_examples](https://en.wikipedia.org/wiki/Source_code_editor#Notable_examples)

<sup>25</sup><https://www.jetbrains.com/student/>

<sup>26</sup><https://phasereditor2d.com/>

<sup>27</sup><https://notepad-plus-plus.org/>

<sup>28</sup><http://brackets.io/>

<sup>29</sup><https://www.jetbrains.com/research/devcosystem-2018/javascript/>

<sup>30</sup><https://www.jetbrains.com/research/devcosystem-2018/javascript/>

<sup>31</sup><https://code.visualstudio.com/>

- **FREE Access** to the online ***Game Designer Tool Kit***<sup>32</sup> — from which you can:
  - Randomly generate game ideas.
  - Collect game prototypes.
  - Automatically generate game design documentation and source code.
- **Remote Web Server**<sup>33</sup> pre-installed with Node.js, PHP, and Python 3.5.
- Are you a student or instructor? Then obtain a FREE copy of **PHPStorm**<sup>34</sup> or **WebStorm**<sup>35</sup> — a savings of \$199 per year; other enticements and discounts available for indie start-ups. ***OVER one third of all professional web developers use this IDE. See the proof here***<sup>36</sup>
- Are you an indie game development start-up or hobbyist with little cash assets? Then obtain a FREE copy of **NotePad++**<sup>37</sup>; it's the editor I use. Or, use **Microsoft Visual Source Code**<sup>38</sup> if you are inclined to develop in Typescript. ***One fourth of all professional web developers use Typescript. See the proof here***<sup>39</sup>
- 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 assets**.<sup>40</sup>

## 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 Game Prototype Starter Kit” from inside the Bonus Content directory — the Bonus Content was an additional download available from your LeanPub.com personal library. For **Amazon patrons**, you can download this template from the book’s website — ***Phaser template kit***.<sup>41</sup> at [http://makingbrowergames.com/p3gp-book/\\_v3.x.x-p3gp-book.zip](http://makingbrowergames.com/p3gp-book/_v3.x.x-p3gp-book.zip)

### Deeper Dive: Project Data Structure

The arrangement of directories (*i.e. (id est) or in other words*,<sup>42</sup> folders) and files is an important consideration. If you use some of those ***“Front-end Build tools”***,<sup>43</sup> your

<sup>32</sup><http://makingbrowergames.com/gameDesigner/>

<sup>33</sup><http://gose-internet-services.net/wordpress-web-hosting/>

<sup>34</sup><https://www.jetbrains.com/phpstorm/buy/#edition=discounts>

<sup>35</sup><https://www.jetbrains.com/webstorm/buy/#edition=discounts>

<sup>36</sup><https://www.jetbrains.com/research/devcosystem-2018/javascript/>

<sup>37</sup><https://notepad-plus-plus.org/>

<sup>38</sup><https://code.visualstudio.com/>

<sup>39</sup><https://www.jetbrains.com/research/devcosystem-2018/javascript/>

<sup>40</sup><https://www.gamedevmarket.net/?ally=GVgAVsoJ>

<sup>41</sup>[http://makingbrowergames.com/p3gp-book/\\_v3.x.x-p3gp-book.zip](http://makingbrowergames.com/p3gp-book/_v3.x.x-p3gp-book.zip)

<sup>42</sup><https://www.grammarly.com/blog/know-your-latin-i-e-vs-e-g/>

<sup>43</sup><https://frontendmasters.com/books/front-end-handbook/2018/tools.html>

project file structure is dictated; this provides less security since your game follows a known directory structure that everyone uses.

I have read dozens of game development authors who literally dictate a rigid organization up to 10+ layers deep. "Why should I follow this?", has always been my question; **How does their organization structure 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**<sup>44</sup> 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**.<sup>45</sup>

On the other hand, if you are working on a game development team, then directory structure takes on a new meaning as: "**Name Space**"<sup>46</sup> for local and public variables. Consistency and standardization rules as King **in software collaborative efforts**.<sup>47</sup> Many game developer turn to **tools such as Vagrant**<sup>48</sup> for help.

When you are creating a game project (**also known as**<sup>49</sup> —aka— Blueprints, templates or Game Starter Kits, make-up your own name) its File Structure should be consistent across all your projects. **"Why's that?" Well, I'll tell you**<sup>50</sup>; **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 IV.

Coming next is the structure I use and believe 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 deploy **away from the global setting (browser window object)**. They are used to organize blocks of functionality into logical groups having a unique identity.

---

<sup>44</sup><https://addyosmani.com/blog/essential-js-namespacing/>

<sup>45</sup><https://namingschemas.com/>

<sup>46</sup><https://en.wikipedia.org/wiki/Namespace>

<sup>47</sup>[https://en.wikipedia.org/wiki/Collaborative\\_software](https://en.wikipedia.org/wiki/Collaborative_software)

<sup>48</sup><https://www.vagrantup.com/>

<sup>49</sup><https://www.frethesaurus.com/also+known+as>

<sup>50</sup>[https://www.youtube.com/watch?v=x2Y7\\_1dILIQ](https://www.youtube.com/watch?v=x2Y7_1dILIQ)

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 `window` global scope of your browser, and make all the game's functions, variables, and properties inside that global object (aka, singleton). Read more here<sup>51</sup> and here<sup>52</sup>.



**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, those additions all share the same global object. Furthermore, there is a chance those additional libraries may use similar naming conventions as yours for their variables and functions; this could cause "name collisions" and override your code's logic. The chances are even higher when libraries that use "\$" as an alias (e.g.: JQuery, Prototype, and others).

### **Concerns using Browserify with Phaser**

Quotes from Phaser 2.7.5 Browserify<sup>a</sup> "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**,<sup>b</sup> 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/>

<sup>a</sup><https://photonstorm.github.io/phaser-ce/index.html>

<sup>51</sup><http://stackoverflow.com/questions/8862665/what-does-it-mean-global-namespace-would-be-polluted/13352212>

<sup>52</sup><http://stackoverflow.com/questions/9773964/understanding-the-javascript-global-namespace-and-closures>

<sup>b</sup><https://github.com/browserify/browserify-handbook>

"Notice the lack of JavaScript namespace here. If you use 3rd party libraries, you might consider using **namespace to isolate your code**". Here's such an example:

### Example: Creating Namespace for game

---

```

118 "use strict";
119 window.GAMEAPP = {
120     // reference to the Phaser.Game instance
121     game: null,
122
123     //Canvas dimensions: world and viewports' Height and Width
124     //**TODO** adjust for your game deployment
125     viewportWidth: 800, //game view using Golden Ration
126     viewportHeight: 500,
127     worldWidth: 800,    //world view using Golden Ration
128     worldHeight: 500,
129     ...
130     // main function
131     main: function(){
132         this.game = new Phaser.Game( config );
133     },
134     // here we will store all game phase/states
135     // state object filled as js files load.
136     state: {}
137 };
138
139 /** DEPRECATED METHOD - NEVER EVER USE THIS AGAIN!
140 * See Phaser.js Game Design Workbook for complete explanation
141 * http://leanpub.com/phaser3gamedesignworkbook
142 * window.onload = function () {
143 *     let game = new Phaser.Game(0, 0, Phaser.AUTO, document.body);
144 * };
145 */
146 //preferred lauch method for BOM.
147 window.addEventListener('DOMContentLoaded', function(){
148     window.GAMEAPP.main();
149 }, false);

```

---



**Hint:** Refer to the line numbers in the source code; it is available [here](#).<sup>53</sup>

---

<sup>53</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/](http://makingbrowsergames.com/p3gp-book/_p3demos/)

## Project Directories & Files

```

1   .URI/<PROJECT NAME>/ //game root directory (single player)
2   └── favicon.ico      //game logo
3   └── index.html       //game front-door entrance
4   └── license.txt      //game EULA @ http://shop.pbmcube.net/
5   └── manifest.json    //game mobile app
6   └── package.json     //for Progressive Web Applications
7   └── purchaseOrd.pdf //how to buy your game
8   └── ReadMe.md        //game info, contact and metadata
9
10  └── 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/
23    └── prefabs/
24    └── states/
25    └── utilities/     //game helpers

```



**Exercise:** Download this Phase Game Prototype starter kit here (35 MB zipped)<sup>54</sup> <http://makingbrowergames.com/book/standalone.zip>



**Exercise:** Read what others say about “**How Do I Organise Files in A Phaser.js Project?**”<sup>55</sup>

### Project File Descriptions:

- **index.html** — Main game container file, your example game should be viewed from within this page.

---

<sup>54</sup><http://makingbrowergames.com/book/standalone.zip>

<sup>55</sup><https://glcheetham.name/2016/03/18/organise-files-phaserjs-project/>

- **.htaccess** — The default web server configs are for Apache. For more information, please refer to the Apache Server Configs documentation<sup>56</sup>. Host your site on a server other than Apache? You're likely to find the corresponding server configs project listed here<sup>57</sup>
- **apple-touch-icon-precomposed.png** — If you want to use different Apple Touch Icons for different resolutions refer to this documentation<sup>58</sup>.
- **crossdomain.xml** — A template for working with cross-domain requests. more about crossdomain.xml here<sup>59</sup>.
- **favicon.ico** — refer to Hans' handy HTML5 Boilerplate Favicon and Apple Touch Icon PSD-Template<sup>60</sup>.
- **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** or readme.md 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).
  - **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.

<sup>56</sup><https://github.com/h5bp/server-configs-apache/tree/master/doc>

<sup>57</sup><https://github.com/h5bp/server-configs/blob/master/README.md>

<sup>58</sup><https://github.com/h5bp/html5-boilerplate/blob/v4.3.0/doc/extend.md#apple-touch-icons>

<sup>59</sup><https://github.com/h5bp/html5-boilerplate/blob/v4.3.0/doc/crossdomain.md>

<sup>60</sup><https://drublic.de/archive/html5-boilerplate-favicons-psd-template/?file=blog/html5-boilerplate-favicons-psd-template/>

- **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 "js" (as I do), "scripts" or "**"source (src as other develops do here)"**".<sup>61</sup> You could include Libraries, plugins, and custom code; or all can be included in a separate sub-directory or directory. It includes some 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.
- **lib/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**"<sup>62</sup> 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.<sup>63</sup>: 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"<sup>64</sup>. Finally, your web server should deliver the machine code with gzip compression for extra speediness." Read more tips here<sup>65</sup>

<sup>61</sup><https://github.com/jdotrjs/phaser-guides/blob/master/Blueprint/Part1.md>

<sup>62</sup><https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/>

<sup>63</sup><https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/>

<sup>64</sup><https://24ways.org/2013/grunt-is-not-weird-and-hard/>

<sup>65</sup><https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/>

## 1.3 Game Project Preparations

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**<sup>66</sup> offers many suggestions from experts in the gaming industry.

### What makes a Good Game?

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

Exactly! A "**good game**" (18-page Bonus Content!)<sup>67</sup> 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**"<sup>a</sup>)

<sup>a</sup><https://amzn.to/2D8c7qM>

<sup>66</sup><http://makingbrowsergames.com/book/Appendix-GameDesignOverview.pdf>

<sup>67</sup><http://makingbrowsergames.com/book/WhatMakesaGoodGame.pdf>



**Hint:** Generally speaking, if a game has over 50% of a **market's share**,<sup>68</sup> it'll be difficult to take on and defeat that "**Boss**".<sup>69</sup>

The best place, to begin our **Game Recipe™**, is to **jot down**<sup>70</sup> your ideas on paper — that's right, draw what you're visualizing — what is your brains "cooking up"? 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 prototypes; you can think of a prototypes as a **"recipe" for an object!** 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 (artwork)**<sup>71</sup>". 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**<sup>72</sup> for some excellent suggestions.



**Exercise:** Download the Terry Paton mind-map on "What makes a Great Game"<sup>73</sup> or find it in your Bonus Content (file name: WhatMakesAGreatGame.pdf). <http://makingbrowergames.com/design/WhatMakesAGreatGame.pdf>

## 1.4 Preparing a "Game Recipe™"

*otherwise known as "Planning your Game Project"*

If you want to develop game recipes, you need to know how to program JavaScript, but you also need to know how to create common solutions for various programming problems. In addition to the problem solving skill, some of those solutions may be more generic than others, and some of the solutions may be more efficient than others. In this regard, game programming is sometimes a trade-off between solving

<sup>68</sup><https://economictimes.indiatimes.com/definition/market-share>

<sup>69</sup>[https://en.wikipedia.org/wiki/Boss\\_\(video\\_gaming\)](https://en.wikipedia.org/wiki/Boss_(video_gaming))

<sup>70</sup><https://idioms.thefreedictionary.com/jot+down>

<sup>71</sup><https://www.gamedevmarket.net/?ally=GVgAVs0j>

<sup>72</sup><http://buildnewgames.com/making-games-fun/>

<sup>73</sup><http://makingbrowergames.com/design/WhatMakesAGreatGame.pdf>

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 tight deadlines. So, we need to think about our development approach very carefully. Writing nice, reusable code — our ultimate goal in game prototyping — won't always take more time than writing "***quick-and-dirty*** ***code***.<sup>74</sup> As you gain more experience in this ***Game Design System™***, you'll find that you start developing a mindset that lets you quickly gauge the kind of solutions that are required for a certain gaming problems.

## 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***.<sup>75</sup> 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?

If you don't have any ideas for a game yet, here are some suggestions to begin your brainstorming:

- Use my online idea generator — the ***Random Game Mechanics Generator***<sup>76</sup> ***for many popular JS Gaming engines!*** We'll cover how to build your own private generator later in Chapter 4.
- What's your favorite game genre? Is it arcade, board, sports? Not Sure? ***Download the Game Categories document***<sup>77</sup>. Could you make a simplified version of that? That's what I'm showing you here in Part II. Could you mix it up a bit, like give it a different theme or main characters? That's what I'll show you with the ***Game Design System™*** automation tools we'll build later in this book.
- What are your other favorite academic fields? If you love art, could you make an art-making tools to support your game? If you love history, how about an interactive story time-line of various characters? If you love science, how about a simulation or analytic tool for your game such as a "Bot" or Artificial Intelligence opponents?
- What's your favorite entertainment? Could you make a game version of a scene or character from a movie, cartoon or TV show? Of course be careful here; you may want to ask permissions. Perhaps you could make a game based on a parallel story around it?
- Do you love a real-life gadget? Could you make that a game widget or prototype?

<sup>74</sup><https://www.urbandictionary.com/define.php?term=quick-and-dirty>

<sup>75</sup><https://www.collinsdictionary.com/us/dictionary/english/jotting>

<sup>76</sup><http://makingbrowsergames.com/gameDesigner/>

<sup>77</sup>[http://makingbrowsergames.com/design/\\_GameCategoriesSubmissions.pdf](http://makingbrowsergames.com/design/_GameCategoriesSubmissions.pdf)

## 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 or framework appearing on the **technology horizon — game-based learning?**.<sup>78</sup> To help us maintain focus and guide us in our selection, here a list of probing questions:

- Is the framework or library **well-used?**<sup>79</sup> 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**"<sup>80</sup> and stamina.
- Who comprises its supporting community? Are they corporations, universities, passionate hobbyists? How does the community respond to each other ... with civility or impassioned fanatical opinions or **one-upmanship?**<sup>81</sup> What is their "**welcome wagon**"<sup>82</sup> 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 collection portfolio. Furthermore, backward compatibility may pose significant problems within your marketing channels and worldwide distributions.
- What live, currently active features make this framework or library better than 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?

<sup>78</sup>[https://www.researchgate.net/publication/216566471\\_What's\\_on\\_the\\_Technology\\_Horizon](https://www.researchgate.net/publication/216566471_What's_on_the_Technology_Horizon)

<sup>79</sup><https://www.thesaurus.com/browse/well-used>

<sup>80</sup><https://www.merriam-webster.com/dictionary/staying%20power>

<sup>81</sup><https://dictionary.cambridge.org/us/dictionary/english/one-upmanship>

<sup>82</sup><https://dictionary.cambridge.org/us/dictionary/english/welcome-wagon>

- Is the framework truly ***performant***<sup>83</sup> or merely an abstraction layer?



**Exercise:** Read this article about "***Shiny New Objects***"<sup>84</sup>



**Exercise:** Read Game Making Tools Features and Comparisons<sup>85</sup> to learn the 180 degree shift in game industry development.

Well, as difficult as this may sound, Phaser III 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 III. **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.

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<sup>86</sup> are not Copyright-able!*** You need to read what the US Copyrights Office says on this matter.



**Exercise:** Read ***Works Unprotected by Copyright Law***<sup>87</sup>

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***

<sup>83</sup><https://www.techopedia.com/definition/28231/performant>

<sup>84</sup><https://dev.to/aspittel/navigating-the-spooky-world-of-javascript-3h45>

<sup>85</sup><https://instabug.com/blog/game-making-tools/>

<sup>86</sup><https://www.bitlaw.com/copyright/unprotected.html#ideas>

<sup>87</sup><https://www.bitlaw.com/copyright/unprotected.html#ideas>



**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”<sup>88</sup>**

QUOTE: **US Government Copyrights Office.**<sup>a</sup>

**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 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.

<sup>a</sup><https://www.bitlaw.com/copyright/license.html>



**Exercise:** Research if “**copy-left**” is a valid form of EULA or implied license according to **US Government Copyrights Office.**



**Hint:** I strongly encourage you to purchase “**How to Copyright Software**” by M. J. Salone (a lawyer!)<sup>89</sup> who shows information “**over-looked**”<sup>90</sup> by the **copy-left movement** and **open source software licenses.**

<sup>88</sup><https://www.bitlaw.com/copyright/unprotected.html>

<sup>89</sup><http://amzn.to/2bmIAcH>

<sup>90</sup>[https://en.wikipedia.org/wiki/First-sale\\_doctrine](https://en.wikipedia.org/wiki/First-sale_doctrine)

And hence, the reasons to write down our ideas in a “**tangible form**”,<sup>91</sup> and further more, **affix a properly labeled notice — use the legal** ©<sup>92</sup> 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**”<sup>93</sup>:

**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: **Copyright © 2014-2018, Stephen Gose. All rights reserved.**

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.



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

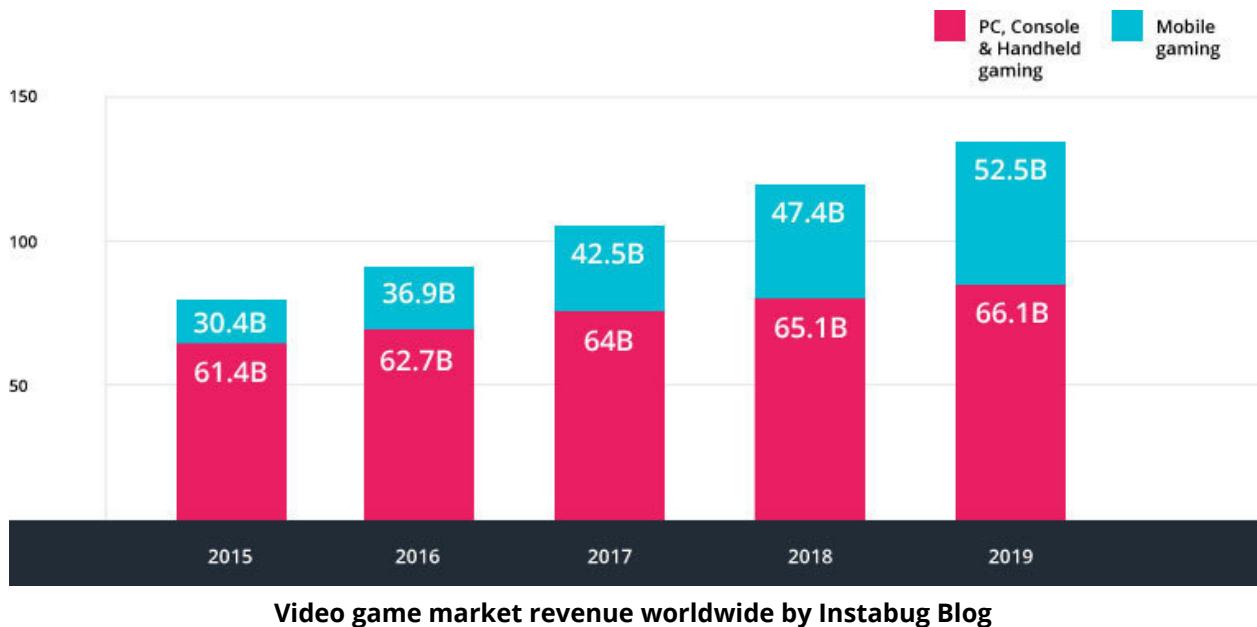
Before we leave the topic about “*What technology ...*”, I’d like to bring your attention to something I’ve discovered recently. It’s an alarming fact, that can’t go “**UN-noticed**”, the current technology trends. The following chart from Instabug on Mobile Game Development<sup>94</sup> shows a significant growth in mobile gaming. Phaser v3.x.x Gaming Framework alone may not fulfill everything required to enter the mobile market.

<sup>91</sup><https://www.bitlaw.com/copyright/formalities.html>

<sup>92</sup>[https://en.wikipedia.org/wiki/Copyright\\_notice#Form\\_of\\_notice](https://en.wikipedia.org/wiki/Copyright_notice#Form_of_notice)

<sup>93</sup>[https://en.wikipedia.org/wiki/Elevator\\_pitch](https://en.wikipedia.org/wiki/Elevator_pitch)

<sup>94</sup><https://instabug.com/blog/mobile-game-development-tools/>



**Exercise:** Read this report from Instabug on Mobile Game Development.<sup>95</sup>

Let me guide your attention to the “News Press Releases” so far this year (**as of 20180901: Google’s 1st page listing** for the search term **“The Best Game Development Tools 2018”**).



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

- **16 Best JavaScript Game Engine**<sup>96</sup> 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<sup>97</sup> — 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<sup>98</sup> — April 25, 2018 (16 months after formal Phaser v2.6.2 release.)

<sup>95</sup><https://instabug.com/blog/mobile-game-development-tools/>

<sup>96</sup><https://www.dunebook.com/16-best-javascript-game-engine/>

<sup>97</sup><https://www.cloudwards.net/best-game-development-tools/>

<sup>98</sup><https://thetool.io/2018/mobile-game-development-platforms>

- Game Development Software by Capterra<sup>99</sup> The Smart Way to Find Business Software
- Top 5 Mobile Game Development Tools 2018<sup>100</sup> 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<sup>101</sup> March 18, 2018 (16 month after formal Phaser v2.6.2 release.)
- Best Game Development Software by G2 Crowd<sup>102</sup> (publication data unavailable)
- Mobile game development in 2018: best tools and advice<sup>103</sup> (publication data unavailable)
- The Most Recommended Game Development Tools and Engine of 2018 For Game Dev<sup>104</sup> (publication data unavailable)
- In-Depth Comparison of the Top Game Making Tools of 2018<sup>105</sup> (publication data unavailable)



**Exercise: History Lesson** — Read the development **history of Phaser from KiwiJS By Rich Davey**<sup>106</sup>



**Exercise:** Do your own “Google Search” — with the latest up-to-date information! — for this search term **“The Best Game Development Tools 2018”**; 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 game development?**

## What features are included?

This is the planning stage **where dreams are turned into the real tangible items**,<sup>107</sup> 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 won’t appear initially.

<sup>99</sup><https://www.capterra.com/game-development-software/>

<sup>100</sup>

<sup>101</sup><https://www.linkedin.com/pulse/7-best-game-development-tools-2018-revolutionize-henry-kundariya/>

<sup>102</sup><https://www.g2crowd.com/categories/game-development>

<sup>103</sup><https://thinkmobiles.com/blog/mobile-game-development-tools/>

<sup>104</sup><https://blog.sagipl.com/game-development-tools/>

<sup>105</sup><https://instabug.com/blog/game-making-tools/>

<sup>106</sup><http://www.html5gamedevs.com/topic/4281-kiwijs-vs-phaser/>

<sup>107</sup>[https://www.youtube.com/watch?v=ZXsQAXx\\_ao0](https://www.youtube.com/watch?v=ZXsQAXx_ao0)

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 between those scenes to show how one menu leads the player into another. Those lines help me understand what code I need in my game program to move between the various game stages.



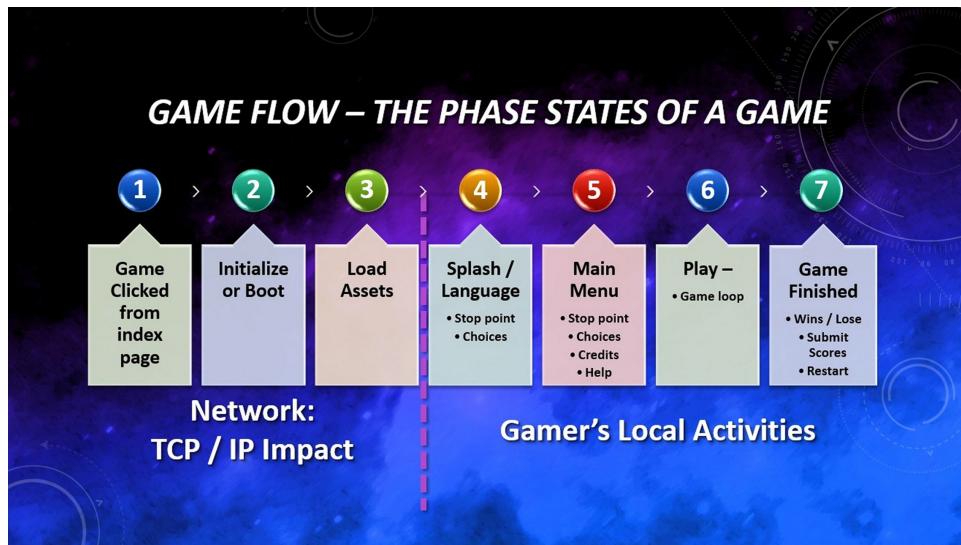
Simple Game Phase Flow for an RPG game

Here’s a more thorough illustration on my various game phases with a break-down of JavaScript recommendations..





**Exercise:** You downloaded this Phase Game Prototype starter kit already, right? (35 MB zipped)<sup>108</sup> <http://makingbrowsergames.com/p3gp-book/standalone.zip>



Typical Game Phases from Phaser III Game Design Workbook

Example from **Apple's Game Kit**<sup>109</sup> using a **Finite State Machine (FSM)**<sup>110</sup> to manage a game's navigation and user interfaces. I'm recommending "Apple" since they are following more what I'm suggesting in Game Design Systems development. (more on FSM in later chapters)

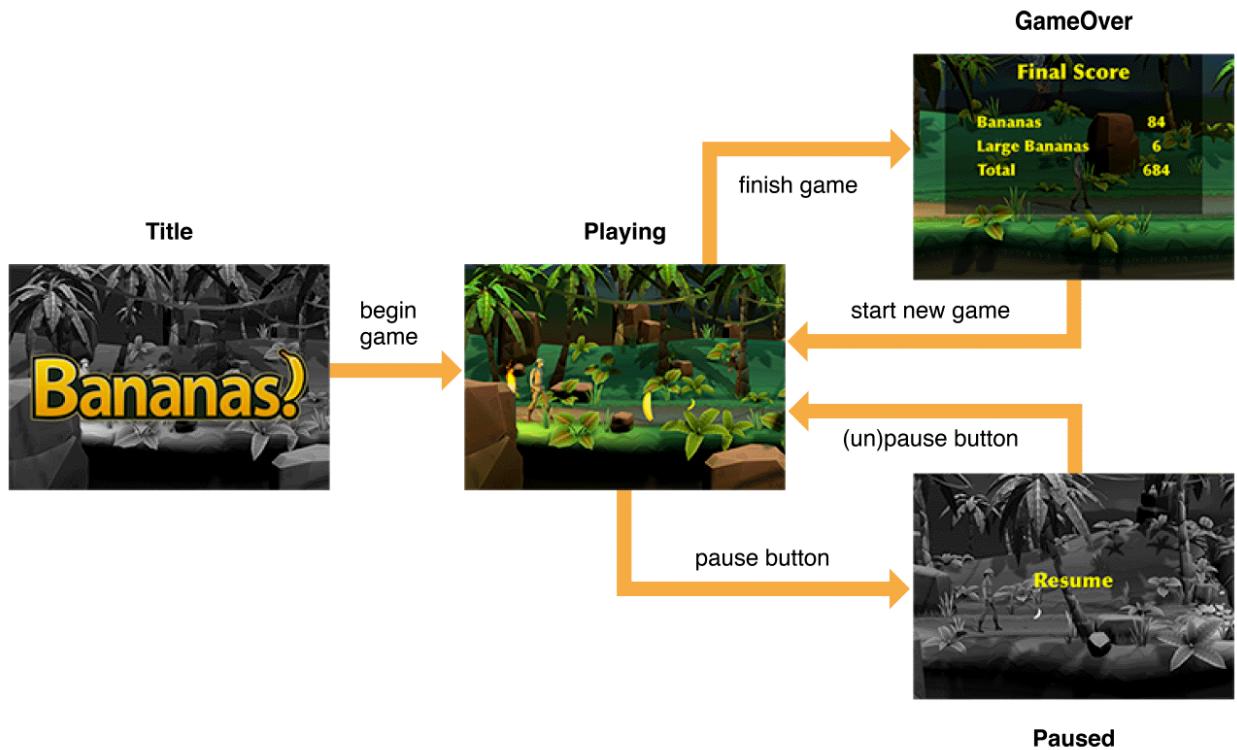


**Note: NOTE:** I've been an "apple" user since 1979 until the migration to "MacIntosh" in 1985. They lost me as a follower; but used their systems in the industry until 2001. I have developed iPhone course at the university level since 2007.

<sup>108</sup><http://makingbrowsergames.com/p3gp-book/standalone.zip>

<sup>109</sup><https://developer.apple.com/documentation/gamekit>

<sup>110</sup><https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867>



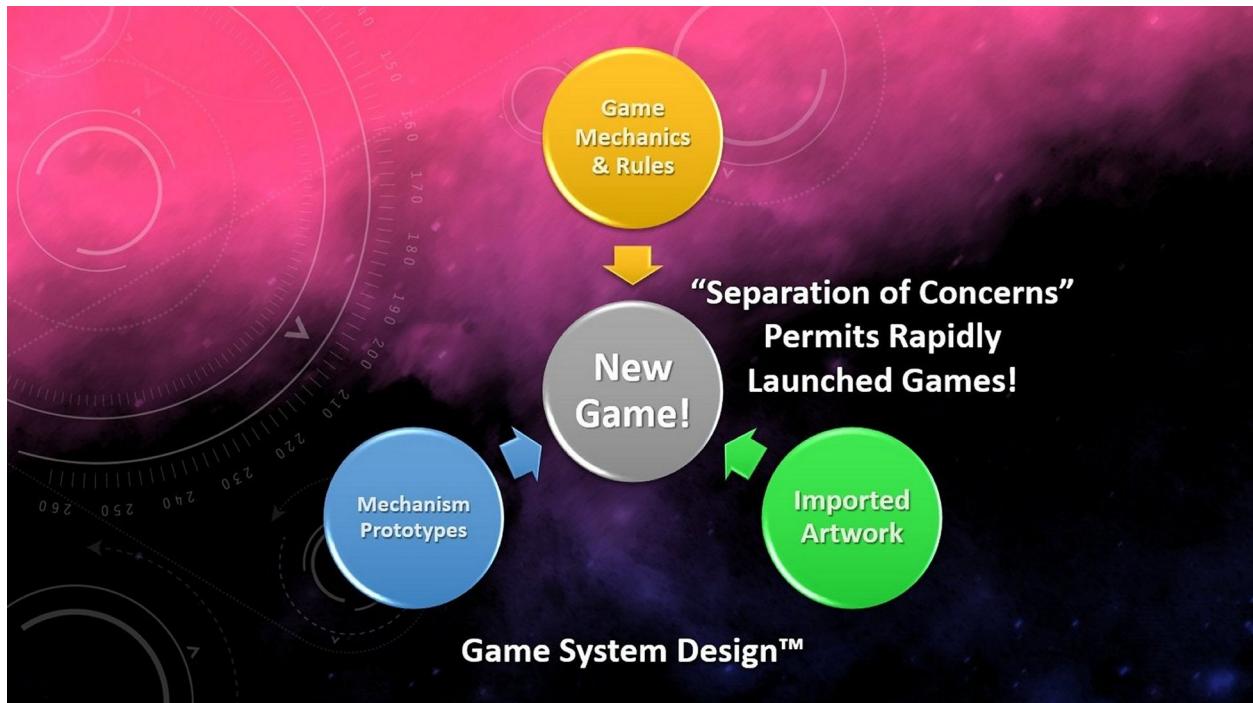
**Note:** Read more details on Apple's Game Kit at <https://developer.apple.com/documentation/gamekit>

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

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<sup>a</sup>** — 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 cover in Part II.

<sup>a</sup><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**.<sup>111</sup> Discover those **differences here ...**<sup>112</sup> "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**<sup>113</sup>)

By combining our game prototype mechanisms, our game mechanics and rules systems with **artwork**,<sup>114</sup> we're able to create various game genres quickly. It simply becomes a matter of exchanging any of those 3 components into new innovative combinations for a new game product release. **This is the secret in concocting a new game**<sup>115</sup> **every month or even every week!**



**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:

---

```

1   Game Play scene has following Game Mechanisms
2     - User-controlled paddle
3     - Multiple colored bricks
4     - Angled ball movement
5     - Collision detection
6     - Life display
7     - Score display
8     - Sound effects
9   Main Scene
10    - "Play" (button)
11    - "Help" (button)
12    - "Credits" (button)
13   Help Scene
14    - Headline: "Game Instructions"
15    - Text: "(explain how to play)"
16    - "Return" to Main Menu (button)
17   Credits Scene
18    - Headline: "Credits"
19    - Text: "(about me & partners)"

```

---

<sup>111</sup>[https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_quick\\_guide.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_quick_guide.htm)

<sup>112</sup><https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238>

<sup>113</sup>[https://en.wikipedia.org/wiki/Component-based\\_software\\_engineering](https://en.wikipedia.org/wiki/Component-based_software_engineering)

<sup>114</sup><https://www.gamedevmarket.net/?ally=GVgAVs0j>

<sup>115</sup><https://www.vocabulary.com/dictionary/concoct>

```

20      - "Return" to Main Menu (button)
21  Win Scene
22      - Headline "You're Awesome!"
23      - Text "(List of Scores)"
24      - [background Fireworks] (animation)
25      - "Return" to Main Menu (button)
26  Lose Scene
27      - Headline: "So Sorry!"
28      - 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!**<sup>116</sup>) 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 those later game releases. 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**?<sup>117</sup>
- 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**<sup>118</sup> is doing — **hiring additional staff**,<sup>119</sup> paying bounty

<sup>116</sup>[https://www.youtube.com/watch?v=kxUdFQ6N\\_OI](https://www.youtube.com/watch?v=kxUdFQ6N_OI)

<sup>117</sup>[https://en.wikipedia.org/wiki/Vertical\\_slice](https://en.wikipedia.org/wiki/Vertical_slice)

<sup>118</sup><https://blog.github.com/2016-04-12-meet-richard-davey-creator-of-phaser/>

<sup>119</sup><https://www.patreon.com/photonstorm>

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.**<sup>120</sup>

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.

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:

```

1      [1] Game Play scene has following Game Mechanisms
2          [1] User-controlled paddle (game object)
3              [1] animated ball (game object)
4                  [1] Multiple colored bricks (game object)
5                      [1] Angled ball movement (coded)
6                          [1] Collision detection (coded)
7                              [2] Life display (text)
8                                  [2] Score display (text)
9                                      [2] Sound effects (coded)
10         [2] Main Menu Scene (game phase/menu)
11             [2] "Play" (button)
12                 [3] "Help" (button)
13                     [3] "Credits" (button)
14         [3] Help Scene (game phase/menu)
15             [3] Headline: "Game Instructions" (text)
16                 [3] Text: "(explain how to play)" (text)
17                     [3] "Return" to Main Menu (button)
18         [3] Credits Scene (game phase/menu)
19             [3] Headline: "Credits" (text)
20                 [3] Text: "(about me & partners)" (text)
21                     [3] "Return" to Main Menu (button)
22         [2] Win Scene (game phase/menu)
23             [2] Headline: "You're Awesome!" (text)
24                 [3] Text: "(List of Scores)" (text)
25                     [3] [background Fireworks] (animation)
26                         [3] "Return" to Main Menu (button)
27         [2] Lose Scene (game phase/menu)

```

---

<sup>120</sup><https://www.indeed.com/jobs?q=Game+Design+Contractor&l=>

```

28 [2] Headline: "So Sorry!" (text)
29 [3] Text: "Boo Hoo! Play Again?" (text)
30 [3] "Return" to Main Menu (button)

```



**Hint:** 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<sup>121</sup>, and you can always stop after a particular iteration and just be happy with what you've made.

#### Sample Sprint Backlog by priority:

---

```

1 First Priority Listing:
2 Game has the following Game Mechanisms
3 [1] User-controlled paddle (game object)
4 [1] animated ball (game object)
5 [1] Multiple colored bricks (game object)
6 [1] Angled ball movement
7 [1] Collision detection
8 [2] Life display (text)
9 [2] Score display (text)
10 [2] Sound effects
11 [2] Main Menu Scene (game phase/menu)
12 [2] Play (button)
13 [2] Win Scene (game phase/menu)
14 [2] Headline: "You're Awesome!" (text)
15 [2] Lose Scene (game phase/menu)
16 [2] Headline: "So Sorry!" (text)
17 [3] Help (button)
18 [3] Credits (button)
19 [3] Help Scene (game phase/menu)
20 [3] Headline: "Game Instructions" (text)
21 [3] Text: "(explain how to play)" (text)
22 [3] Return to Main Menu (button)
23 [3] Credits Scene (game phase/menu)
24 [3] Headline: "Credits" (text)
25 [3] Text: "(about me & partners)" (text)
26 [3] Text: "(List of Scores)" (text)

```

---

<sup>121</sup><https://www.scrum.org/resources/what-is-a-sprint-in-scrum>

---

```

27      [3] [background Fireworks] (animation)
28      [3] Text: "Boo Hoo! Play Again?" (text)
29      [3] Restart (button)

```

---

### **Refer: Sprint Backlog by priority here<sup>122</sup>**

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:**

#### **2nd chart sorted alphabetically by Game Mechanisms**

---

```

1 // Second Chart: (Categories then original content)
2 (animation)      [background Fireworks]
3 (button)         Play
4 (button)         Help
5 (button)         Credits
6 (button)         Return to Main Menu
7 (button)         Restart
8 (coded)          Angled ball movement
9 (coded)          Collision detection
10 (coded)         Sound effects
11 (game object)   User-controlled paddle
12 (game object)   animated ball
13 (game object)   Multiple colored bricks
14 (game phase/menu) Main Menu Scene
15 (game phase/menu) Win Scene
16 (game phase/menu) Lose Scene
17 (game phase/menu) Help Scene
18 (game phase/menu) Credits Scene
19 (text)           Life display
20 (text)           Score display
21 (text)           Headline: "You're Awesome!"
22 (text)           Headline: "So Sorry!"
23 (text)           Headline: "Game Instructions"
24 (text)           Headline: "Credits"
25 (text)           Text: "(explain how to play)"
26 (text)           Text: "(about me & partners)"
27 (text)           Text: "(List of Scores)"
28 (text)           Text: "Boo Hoo! Play Again?"

```

---

<sup>122</sup><https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/sprint-backlog>



**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.**<sup>123</sup>

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 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) **Game Mechanisms** items to create (buttons, objects, menus and text), an animation effect, 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 names or functions should you use in your program? How will we design our game software? What is our game architecture?



**Exercise:** Refer to **Chapter 2: 4-Step Method**

## Design Architecture: “Oh! Oh!”

The answer comes from a phrase senior software engineers call “**high level architecture**” **design**.<sup>124</sup> Using **Object-Oriented Analysis Design (OOAD)**<sup>125</sup> 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” (also

<sup>123</sup><http://www.finitearts.com/pages/400page.html>

<sup>124</sup>[https://en.wikipedia.org/wiki/Architectural\\_pattern](https://en.wikipedia.org/wiki/Architectural_pattern)

<sup>125</sup>[https://en.wikipedia.org/wiki/Object-oriented\\_analysis\\_and\\_design](https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design)

known as<sup>126</sup> —aka—, objects), rules and metrics (aka, “logic”), “user interaction” (aka, UI), information and “user data”, and “scenes” (i.e., what is the player progress in the game) — then think about how you might write those items as JavaScript code, such as object types, functions, and variables. Here’s another example:

- Game Menus and Scenes (plural noun)
- Music Tiles (plural noun)
- Music sound files (plural noun)
- 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) method. From a different “Elevator Speech” and game description I wrote, I collected all the **nouns** from the game’s description. This is not executable JavaScript yet; it’s called `pseudo-code` and we have a lot more to do. “OOAD” should include some **adjectives** (properties of those **noun-things**<sup>127</sup> 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 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. Then 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.

### Download Comparison Chart of Fun to Human Emotions<sup>128</sup>

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

---

<sup>126</sup><https://www.freethesaurus.com/also+known+as>

<sup>127</sup><https://www.urbandictionary.com/define.php?term=thingie>

<sup>128</sup>[http://makingbrowergames.com/design/\\_p3-16HumanMotivations.pdf](http://makingbrowergames.com/design/_p3-16HumanMotivations.pdf)

- **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 "Chart sorted by Items" instead of this listing above.

## Design Architecture: “Top-down”

**Top-down design** (aka Step-wise refinement) is **another technique<sup>129</sup> — among many; (click here to see the 10 most commonly used)**<sup>130</sup> — that professional programmers use when they have to go beyond simply identifying items — as we 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.)

## Design Architecture: “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 v3.x.x library.**

### **References:**

---

<sup>129</sup>[https://en.wikipedia.org/wiki/List\\_of\\_software\\_architecture\\_styles\\_and\\_patterns](https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns)

<sup>130</sup><https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>

- 1) Programming like a Pro<sup>131</sup> Chapter 8, by Charles R. Hardnett
- 2) Google Analytics<sup>132</sup>

## “Oh! Oh!” vs. Top-Down vs. Bottom-Up

You might be wondering which design philosophy is the best one for you to adopt and use: “OOAD”, top-down or bottom-up. In reality, neither is better than the other. 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.

## What's your time-line?

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 your onto” 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

---

<sup>131</sup><http://amzn.to/2b8gvUr>

<sup>132</sup><https://developers.google.com/analytics/devguides/collection/>

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!



**Exercise:** Read why so many business fall victim to poor time management in this article: **Scrum makes you dumb**<sup>133</sup> ... (an excerpt)

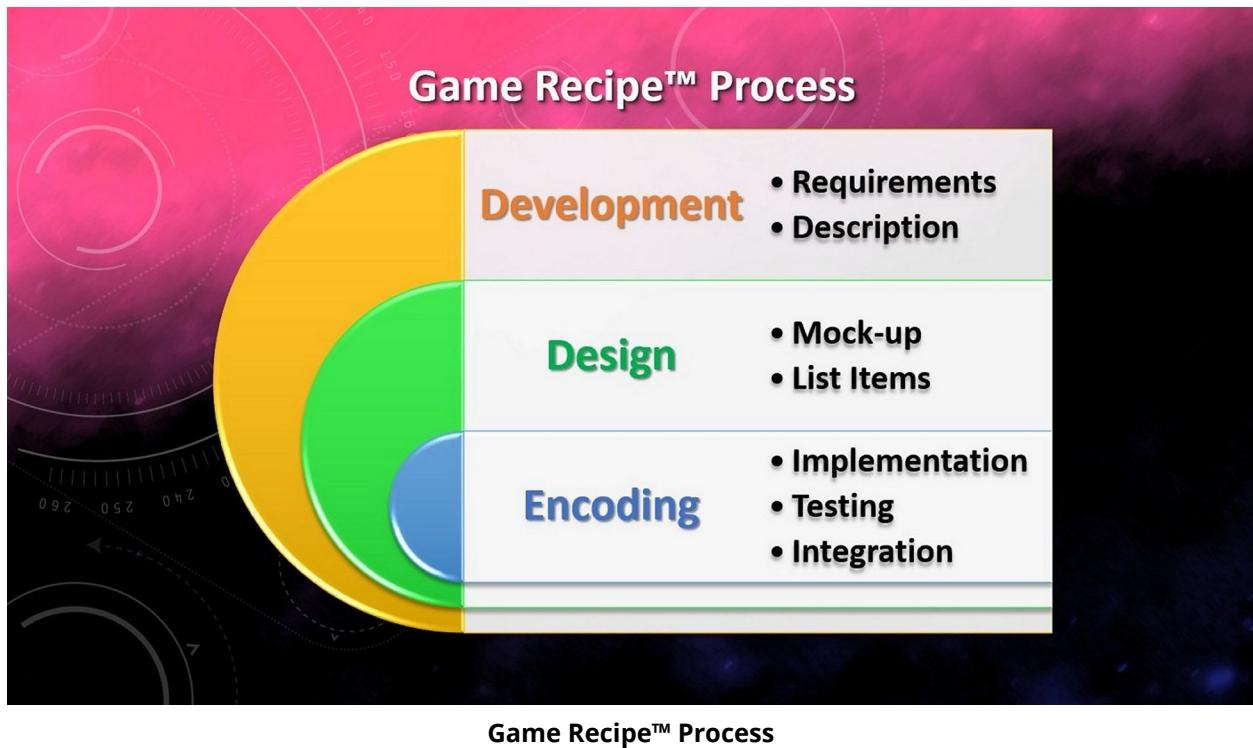
**If your software developers are able to accurately estimate how long something will take, you should fire them.** If they've done something so many times before that they know exactly how long it'll take them to do it again, then **they should have made a reusable solution by now. (ed.: a game prototype!)**

## Are you ready?

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, **because you're turning your dreams into reality.**

<sup>133</sup><https://www.linkedin.com/pulse/scrum-makes-you-dumb-daniel-jones/>

## 1.5 Game Recipe™ Summarized:



### Development:

1. Copy a fresh/new 'file structure' into a new or separate project directory (also known as — aka — a folder). A basic `index.html` file should be there already; just update the `<head>` metadata for search engine optimization (SEO). (See **Chapter 2 Worksheet #1-1**)
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**". (use an Excel spread sheet?)
5. Sort the items by either: 1) priorities, or 2) "**Catalog classifications**" (i.e., this is my favorite method and easiest for me to encode content.)

## Encoding:



**Note:** following the *Principles of Software Engineering*<sup>134</sup>

6. Use (aka “**implement!**” ) *software architecture design to break down*<sup>135</sup> (aka, **Deconstruct**) the various mechanisms and components into their logical elements. Use a combination of “OOAD”, top-down, and/or “bottom-up” design methods.
7. Find those **previously tested** prototype items **you’ve already created in other game products in the Game Recipes™ tool.**<sup>136</sup> 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!**

Download this FREE 400+ page ebook: “**Game Development for Human Beings**” from **GameDev Academy**.<sup>137</sup>



**Note:** Read *Getting Started Making Video Games by John Horton*<sup>138</sup> found in your Bonus Content.

## 1.6 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.

<sup>134</sup>[http://makingbrowergames.com/design/\\_PrinciplesofSoftwareEngineering.pdf](http://makingbrowergames.com/design/_PrinciplesofSoftwareEngineering.pdf)

<sup>135</sup><https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/>

<sup>136</sup><http://makingbrowergames.com/gameDesigner/>

<sup>137</sup><https://gamedevacademy.org/free-ebook-game-development-for-human-beings/?a=47>

<sup>138</sup>[http://makingbrowergames.com/design/\\_p3-GettingStartedMakingVideoGames.pdf](http://makingbrowergames.com/design/_p3-GettingStartedMakingVideoGames.pdf)

- 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.

## 1.7 Chapter References:

(*See more references*)

- Tools for Web Developers — Setting Up Your Dev Environment<sup>139</sup>
- How to use browserify<sup>140</sup> to build modular applications. Free handbook.
- Google Search for Text Editor for Source Code Development<sup>141</sup>
- “Using ECMAScript 6 today”<sup>142</sup> 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”<sup>143</sup> 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<sup>144</sup> is a quick intro to ECMAScript 6 modules available on Github<sup>145</sup>. Especially interesting is another

<sup>139</sup><https://developers.google.com/web/tools/setup/>

<sup>140</sup><https://github.com/substack/browserify-handbook>

<sup>141</sup><https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=text%20editor%20for%20source%20code>

<sup>142</sup><http://2ality.com/2014/08/es6-today.html>

<sup>143</sup><http://2ality.com/2013/11/es6-modules-browsers.html>

<sup>144</sup><http://jsmodules.io/>

<sup>145</sup><https://github.com/wycats/jsmodules>

page<sup>146</sup> where CommonJS modules are shown side by side with their ECMAScript 6 versions.

- Understanding ES6 Modules<sup>147</sup>
- How the Web works<sup>148</sup> Mozilla Developer's Network (MDN) provides this Learning Area designed to answer common questions that come up.

---

<sup>146</sup><http://jsmodules.io/cjs.html>

<sup>147</sup><https://www.sitepoint.com/understanding-es6-modules/>

<sup>148</sup>[https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions#How\\_the\\_Web\\_works](https://developer.mozilla.org/en-US/docs/Learn/Common_questions#How_the_Web_works)



## 2. Building a Game Launch Pad

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

**Answer:** It is an operational gaming foundation that can:

1. accept inputs;
2. move game components;
3. animate, and
4. react to internal game objects.

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

**Answer:** See the latest comments from various gaming experts [here](#)<sup>1</sup> and [here](#)<sup>2</sup>; and other software engineers' opinions about ***prototyping in general — here***.<sup>3</sup>

My game design concept is clearly echoed in both Apple's Game-Play Kit and ***Play Canvas***<sup>4</sup> as "***Entities and Components***".<sup>5</sup> 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.



**Exercise:** Read "***Apple's Game-Play Kit: Designing with Entities and Components***".<sup>6</sup> and learn how ***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: (quote!) ...

---

<sup>1</sup><https://www.quora.com/What-is-the-benefit-from-creating-the-prototype-of-a-game-first#>

<sup>2</sup><https://www.quora.com/Do-game-developers-create-prototypes-first-before-programming-the-actual-game>

<sup>3</sup>[https://www.sqa.org.uk/e-learning/IMAuthoring01CD/page\\_06.htm](https://www.sqa.org.uk/e-learning/IMAuthoring01CD/page_06.htm)

<sup>4</sup><https://playcanvas.com/>

<sup>5</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/EntityComponent.html#/apple\\_ref/doc/uid/TP40015172-CH6-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html#/apple_ref/doc/uid/TP40015172-CH6-SW1)

<sup>6</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/EntityComponent.html#/apple\\_ref/doc/uid/TP40015172-CH6-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/EntityComponent.html#/apple_ref/doc/uid/TP40015172-CH6-SW1)

## 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**,<sup>a</sup> 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. ...

...

### ***#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<sup>b</sup> before coding; these days I know that the tech specs and evolving game-play mechanics<sup>c</sup> will mean that much of what you make at the start won’t make it into the finished game.”***

Read more [here](#).<sup>d</sup>

<sup>a</sup><https://www.gamedevmarket.net/?ally=GVgAVsoj>

<sup>b</sup><https://www.gamedevmarket.net/?ally=GVgAVsoj>

<sup>c</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

<sup>d</sup><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<sup>7</sup>

## 2.1 Example: Box Graphics Prototypes

Since we are just 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 offers 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 swapping out these simple blocks for a variety of artwork themes, this will give us the opportunity to create 100s of games with similar game mechanics<sup>8</sup> in different theme settings.**



We’ll explore different game perspectives, mechanics, themes and modes later in this book and adjust these prototype mechanisms accordingly.



**Exercise:** Study the new v3.13.0 shapes features by **reading this DevLog**.<sup>6</sup><sup>9</sup>

---

<sup>7</sup><https://www.youtube.com/watch?v=yzeJ77RNcjs>

<sup>8</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

<sup>9</sup><https://phaser.io/phaser3/devlog/128>

**Sample 2.1: Prototyping Graphics**

```
// available in Phaser v3.13.0 and later.  
// 2D: this.add.rectangle(x, y, width, height, color)  
// AND even ...  
// 2.5 & 3D as the new 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 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: 0xFFFFF}  
    ) // call out to factory function  
  
); //End sprite  
); //new shiny avatar!  
  
//  
// ======  
//create a "box" on the HTML5 canvas for various game components.  
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:
```

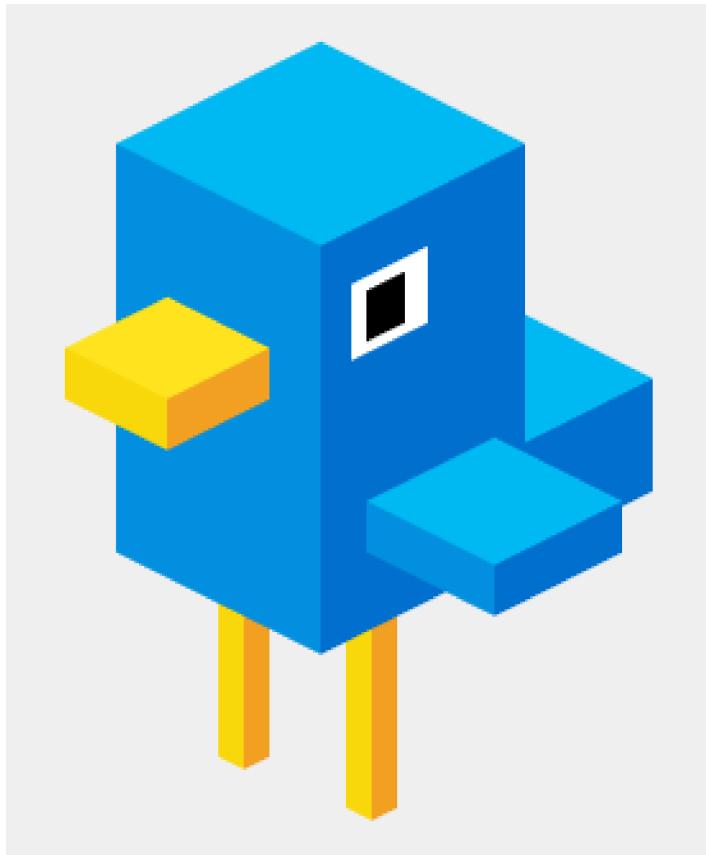
```
var bxImg = opt.who.add.rectangle(  
    opt.whereX,  
    opt.whereY,  
    opt.width,  
    opt.length);  
// decorate our shiny new "box"  
var bxColor = opt.who.add.graphics(  
    {fillStyle: {color: opt.color},  
     lineStyle: {color: opt.border} });  
  
bxColor.fillRectShape(bxImg); //fill box with color  
bxColor.strokeRectShape(bxImg); //draws a border around it.  
return bxImg;  
};
```

---

This new *shape feature*<sup>10</sup> 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.** You can substitute shape for images and sprites, and apply movement and physics reactions. It’s even possible to build 2.5D and 3D games with just isoboxes.

---

<sup>10</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/gameobjects/shape/Shape.js>



Phaser v3.16.1 isobox for 3D games.

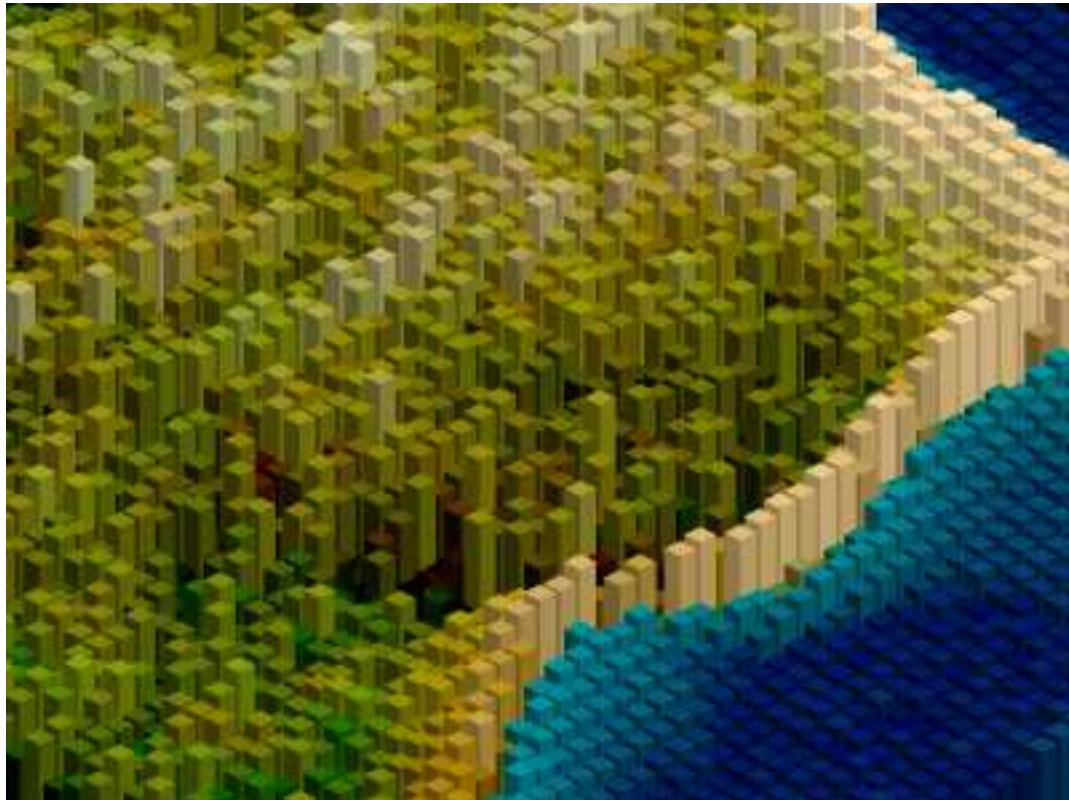
Quote about the **IsoBox Shape<sup>a</sup>** "... is a Game Object that can be added to a Scene, Group or Container — **WARNING: "some massive changes coming in v3.17+ about "containers.** It provides a quick and easy way for you to render this shape in your game without using a texture, while still 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.

<sup>a</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.IsoBox.html>



**ISO-terrain demonstration**

## **References:**

- **How to create Phaser v3.16.x Graphics**<sup>11</sup>
- **Turning static graphics into Sprites**<sup>12</sup>
- **Sample Stacker Game using shapes**<sup>13</sup>

**What time did all that take? a couple of seconds?** This way — using generic boxes or the **new Phaser v3.14+ “rectangle shape”**<sup>14</sup> — we can deploy them to represent any

<sup>11</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Graphics.html>

<sup>12</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Graphics.html#generateTexture>

<sup>13</sup><http://labs.phaser.io/view.html?src=src%5Cgame%20objects%5Cshapes%5Cstacker%20es6.js&v=128>

<sup>14</sup><http://labs.phaser.io/edit.html?src=src/game%20objects/shapes/rectangle%20with%20arcade%20physics.js>

game elements as well as player character(s), boundaries, walls, doors, treasures, and opponent(s) entities.



**Exercise:** Download the example above: [http://makingbrowsergames.com/p3gp-book/\\_p3demos/game.js](http://makingbrowsergames.com/p3gp-book/_p3demos/game.js)

**By swapping out these simple blocks for a variety of artwork themes settings,<sup>15</sup> it gives us the opportunity to create 100s of games along similar game mechanics<sup>16</sup>.**



**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.*<sup>17</sup>

## 2.2 “ToTo, ... we’re not in Kansas anymore” — Dorothy

*Welcome to OZ ... er! uHMM! “PHAZ3R”, Dorothy!*<sup>18</sup>

Phaser **pre-v3.16.x** is not for the “faint-hearted”. 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+)**<sup>19</sup> and **Zenva Online Game Academy (v3.12+)**<sup>20</sup> began to appear, **OOAD**<sup>21</sup> was then possible.

Phaser III is a sophisticated re-write that reminds me of **KiwiJS**<sup>22</sup> — the grand-father of Phaser v2.x.x. Phaser III replaces **PIXI** with native code, and **supplies many new features that transcends any of its lineage predecessors**. Quoting from the `dev` logs

<sup>15</sup><https://www.gamedevmarket.net/?ally=GVgAVsoj>

<sup>16</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

<sup>17</sup><http://www.lessmilk.com/tutorial/2d-platformer-phaser>

<sup>18</sup><https://www.youtube.com/watch?v=vQLNS3HWfCM>

<sup>19</sup><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>

<sup>20</sup>[https://academy.zenva.com/?a=47&s=phaser&submit=Search&post\\_type=product&campaign=Phaser3GamePrototyping](https://academy.zenva.com/?a=47&s=phaser&submit=Search&post_type=product&campaign=Phaser3GamePrototyping)

<sup>21</sup>[https://en.wikipedia.org/wiki/Object-oriented\\_analysis\\_and\\_design](https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design)

<sup>22</sup><http://www.kiwijs.org/>

"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"<sup>a</sup> in February 2017 until now (September 2018). 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**,<sup>b</sup> (ed.: Please read "JavaScript Module Systems Showdown: CommonJS vs AMD vs ES2015"<sup>c</sup>.) 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`.<sup>d</sup> (ed.: research what `webpack2` does<sup>e</sup> to raw JavaScript code and how it works.<sup>f</sup> ) All of the code is being updated (or has been updated) to **use CommonJS format modules**. (ed.: review Part III) 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.** (ed.: review Part III) 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>

<sup>a</sup><http://phaser.io/phaser3/history>

<sup>b</sup><https://webpack.js.org/concepts/modules/>

<sup>c</sup><https://auth0.com/blog/javascript-module-systems-showdown/>

<sup>d</sup><https://blog.madewithenvy.com/getting-started-with-webpack-2-ed2b86c68783#.fnuaum5tw>

<sup>e</sup><https://webpack.js.org/concepts/>

<sup>f</sup><http://kangax.github.io/compat-table/es5/>



Webpack2 outputs ES5 source code<sup>23</sup>! Do we need to know this? Not just yet. Bookmark these sites for later use in Part III:

- **Beginners guide to webpack—How to start a basic application with webpack 2**<sup>24</sup>
- **A Beginner's Guide to Webpack 4 and Module Bundling**<sup>25</sup>

<sup>23</sup><https://medium.com/@rajaraodv/webpack-the-confusing-parts-58712f8fcad9>

<sup>24</sup><https://medium.com/@ahsan.ayaz/beginners-guide-to-webpack-how-to-start-a-basic-application-with-webpack-2-ebed3172fa8c>

<sup>25</sup><https://www.sitepoint.com/beginners-guide-webpack-module-bundling/>

## 2.3 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 4-steps, from here on, whenever we need to generate ***new game prototypes and component mechanisms***. It'll become our ***regimen***<sup>26</sup>:

**1st RULE: Always be consistent in placement, programming paradigm<sup>27</sup>, JS coding style<sup>28</sup>, and naming schemes<sup>29</sup>. It might be worth reviewing what others are doing<sup>30</sup> with their JS Styling<sup>31</sup> and how they program JavaScript.<sup>32</sup>**

- **Step 0)** (Preparation) This is the fun part when developing a new game — so, enjoy! (***Boy Scouts: Earn your Merit Badge!***<sup>33</sup>)
  - Research and play 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.***
  - Follow the ***Game Recipe***™ **Steps #1 through 8.** or use the ***Game Recipe***™ **Tool**<sup>34</sup> from Chapter 4.
- **Step 1)** Generate game phases as needed. Once these are created, ***they should be “relatively” D.R.Y. (Don’t Repeat Yourself)***
- **Step 2)** Generate code for triggering events. (i.e., listeners, observers, sensors, sentinels, web worker, etc.).
- **Step 3)** Generate code that helps transition “into and out from” the new game phase. Once these are created, ***they should be “relatively” D.R.Y. (Don’t Repeat Yourself)***
- **Step 4)** Create your game’s core and auxiliary functions and objects, as required, for this new game’s phase.



***Hint:*** Watch for these ***4-Step numbers*** annotated inside the source code.

---

<sup>26</sup><https://www.merriam-webster.com/dictionary/regimen>

<sup>27</sup><https://github.com/getify/You-Dont-Know-JS/blob/master/this%20%26%20object%20prototypes/ch6.md>

<sup>28</sup><https://codeburst.io/5-javascript-style-guides-including-airbnb-github-google-88cbc6b2b7aa>

<sup>29</sup>[https://en.wikipedia.org/wiki/Computer\\_network\\_naming\\_scheme](https://en.wikipedia.org/wiki/Computer_network_naming_scheme)

<sup>30</sup><https://standardjs.com/>

<sup>31</sup><https://hackernoon.com/what-javascript-code-style-is-the-most-popular-5a3f5bec1f6f>

<sup>32</sup><https://github.com/getify/You-Dont-Know-JS>

<sup>33</sup>[http://makingbrowergames.com/book/Game\\_DesignMeritBadge.pdf](http://makingbrowergames.com/book/Game_DesignMeritBadge.pdf)

<sup>34</sup><http://makingbrowergames.com/gameDesigner/>

**Download this “template”<sup>35</sup> as a new project reference. Open and watch the Developer’s Console while running this template or simply click and watch this example inside the Developer’s Console:**

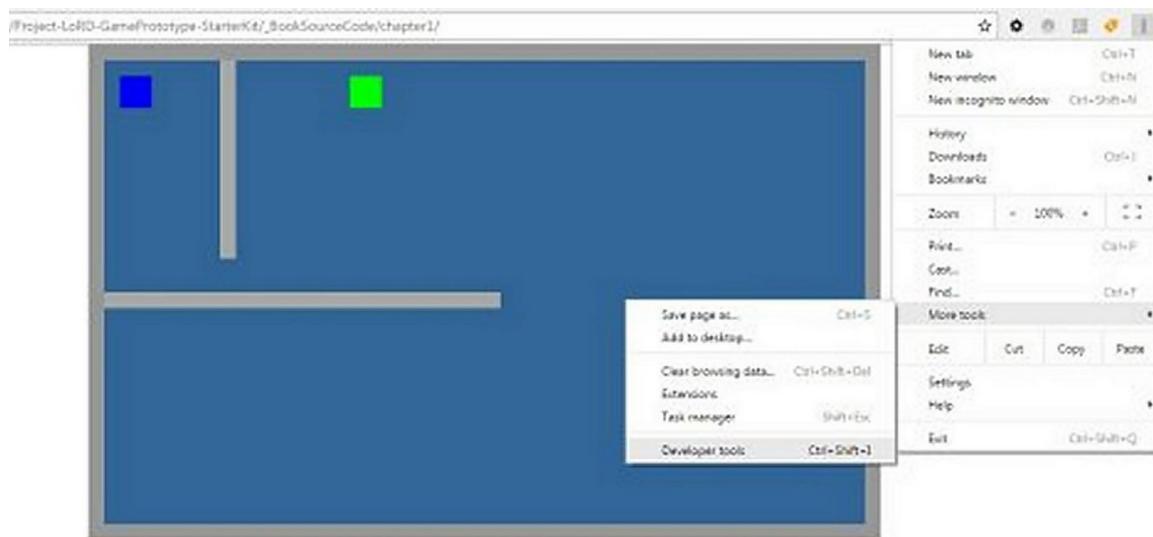
<http://makingbrowsergames.com/p3gp-book/standalone/>



**Warning: Avoid “Anti-patterns”<sup>36</sup>** when developing your game source code when integrating others’ supporting functions, document any **encroaching**<sup>37</sup> “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**<sup>38</sup>. 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 it alive**.<sup>39</sup> 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 develop tools. The browser console in the “Developer Tools”, tells us a lot about our game’s performance.



**Access to Develop tools and console in Chrome**

<sup>35</sup>[http://makingbrowsergames.com/p3gp-book/\\_v3.x.x-p3gp-book.zip](http://makingbrowsergames.com/p3gp-book/_v3.x.x-p3gp-book.zip)

<sup>36</sup><https://addyosmani.com/resources/essentialjsdesignpatterns/book/#antipatterns>

<sup>37</sup><https://www.dictionary.com/browse/encroaching>

<sup>38</sup><https://addyosmani.com/resources/essentialjsdesignpatterns/book/>

<sup>39</sup><https://idioms.thefreedictionary.com/bring+it+alive>

But first, we need a web page to hold our game . . .

## 2.4 Game Recipe™ Step #1) 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 game. 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 title and metadata inside each new project's index.html.***



***Hint: Both methods are in the [http://makingbrowsergames.com/p3gp-book/\\_v3.x.x-p3gp-book.zip](http://makingbrowsergames.com/p3gp-book/_v3.x.x-p3gp-book.zip)***

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**)<sup>40</sup> or **Progressive Web App (PWA)**.<sup>41</sup> I recommend using a "**mobile first, responsive design**"<sup>42</sup> for all `HTML` pages. There are two ways we can proceed:

1. the "*traditional method*" — Task #1-1 (below) or ***See the Appendix***<sup>43</sup>
2. the "*unorthodox method*" for mobile devices — 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.



***Note:*** If you're curious as to why we must have an `index.html`? I recommend a side-trip to this article on "**Why is it important that we name the main file index.html?**"<sup>44</sup>

<sup>40</sup>[https://en.wikipedia.org/wiki/Single-page\\_application](https://en.wikipedia.org/wiki/Single-page_application)

<sup>41</sup><https://developers.google.com/web/progressive-web-apps/>

<sup>42</sup><http://fredericgonzalo.com/en/2017/03/01/understanding-the-difference-between-mobile-first-adaptive-and-responsive-design/>

<sup>43</sup><http://makingbrowsergames.com/book/Appendix-buildHTML5webPage.pdf>

<sup>44</sup><https://teamtreehouse.com/community/why-is-it-important-that-we-name-the-main-file-indexhtml>

## 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***)
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. ***Download the 6-page worksheet #1-1 here***<sup>45</sup> or ***See Appendix: How to create an HTML5 web page***<sup>46</sup>

### Instructional `Index.html` – Traditional Method

---

```

<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" > <!-- MUST BE WITHIN FIRST 1024 BYTES -->
<title>Phaser Game Prototyping - (Your Game Title Here)</title>
<meta name="description" content="Phaser Game Prototyping Template" />
<link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
<!-- CSS must appear before JS -->
<style>
    body{margin:0}
    canvas {margin: 0 auto;}
</style>
</head>
<body itemscope itemtype="http://schema.org/CreativeWork/WebApplication">
<div id="orientation"></div>
<div id="gContent"></div>

<!--
NOTE: Phaser library must be loaded prior to any game logic.
We load script files here to avoid window.onload call.
Window.onload is rarely used for many reasons, and because
    Phaser doesn't wait until all resources are loaded.
The DOMContentLoaded event triggers when the page is ready.
Phaser waits for the full HTML and scripts and then starts.
This is explained in greater detail in Amazon edition of
    "Phaser III Design Guide workbook"
    https://leanpub.com/phaser3gamedesignworkbook/
-->
```

---

<sup>45</sup><http://makingbrowergames.com/book/ProjectIndex.pdf>

<sup>46</sup><http://makingbrowergames.com/book/Appendix-buildHTML5webPage.pdf>

```

<!-- REMEMBER Phaser v2.x.x IS NOT COMPATIBLE WITH PHASER v3.x.x
Phaser Official version
<script src="https://cdn.jsdelivr.net/npm/phaser@3.16.2/dist/phaser.js">
</script>

Phaser Community Edition active (UPDATE TO LATEST VERSION )
<script src="https://cdnjs.cloudflare.com/ajax/libs/phaser-ce/2.11.1/phaser.min.\
js">
</script>

Phaser v3.x.x (20180322); REMOVE ANTI-PATTERN
CORS: Remove anti-pattern!
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
- https://enable-cors.org/
Avoid this anti-pattern of omitting the protocols schemes in script "src";
Cross origin requests are only supported for protocol schemes:
  http, data, chrome, chrome-extension, https.
- ***the anti-pattern is this: (missing protocol)***

<script src="//cdn.jsdelivr.net/... .
- use this: <script src="https://cdn.jsdelivr.net/... .

REMINDER Phaser v2.x.x IS NOT COMPATIBLE WITH PHASER v3.x.x -->
<script src="https://cdn.jsdelivr.net/npm/phaser@3.16.2/dist/phaser.js">
</script>

<!--
NOTE: per the Phaser.JS Design Guide workbook, you may
place the following script externally as the last header
script using defer. OR you may write in-line.
See Chapter 2
-->
<script defer id="launch" src="js/main.js"></script>

<!-- The unorthodox method would place all the main.js content
inside this index page -->
</body>
</html>

```

---



**Exercise: See a “Bare-bones” Index Page live here.**<sup>47</sup> This is an example of an index.html used in development only.

<sup>47</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowsergames.com/p3gp-book/_p3demos/bareBonesIndex.html)



**Note:** You might like to try the “**15 seconds**” **HTML page creation tool**.<sup>48</sup> This responsive template comes with “**Golden Ratio**” already pre-configured. You can read about the “**Golden Ratio**” **here**.<sup>49</sup> The Golden Ratio is a weird mathematical proportions that our visual perception prefers. Learn even more about cutting-edge web design **using the Golden Ratio here**.<sup>50</sup> Otherwise, if you’ve previously worked with Bootstrap, you might like using their new **Drop-n-drag Layout Builder**<sup>51</sup>.

You shouldn’t have to change too much in this `index` 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)**<sup>52</sup>, the absolute barest essentials for a properly formatted `index.html` page. In our **Phaser III Design Guide workbook**,<sup>53</sup> 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.<sup>54</sup>



**Warning: Google AMP**<sup>55</sup> `index.html` page **requires the head and body tags** in a browser documents. Read about it **here**.<sup>56</sup>

Inside your `index.html` you need to choose which Phaser version to use in your game. I have set the default to v3.16.1 (released: ~20181002). The other Phaser v2.xx are commented out. Notice that the Phaser scripts are minified and already come from the appropriate content delivery networks (CDN). **ALWAYS use the CDN versions for fastest load times.**



**Exercise: Study which CDNs are the fastest (click here)**<sup>57</sup>



**Hint:** Always use the minified CDN version of Phaser v3.16+ — **never your own** — because, the files are moved closer to the Internet gamer and reduces their download time. It further increases the chances that the gamer may have Phaser III **already** in their browser cache which results in **0 download time!** Read Yahoo’s analysis on “empty cache” vs. “full cache” the **Surprising Results (excerpt from Yahoo blog)**<sup>58</sup>

<sup>48</sup><http://www.initializr.com/>

<sup>49</sup><https://www.goldennumber.net/>

<sup>50</sup><https://code.tutsplus.com/tutorials/the-golden-ratio-in-web-design--net-2272>

<sup>51</sup><http://www.layoutit.com/build>

<sup>52</sup><https://stackoverflow.com/questions/9797046/whats-a-valid-html5-document>

<sup>53</sup><http://leanpub.com/phaser3gamedesignworkbook>

<sup>54</sup><http://makingbrowergames.com/p3gp-book/#pricing>

<sup>55</sup><https://www.ampproject.org/learn/overview/>

<sup>56</sup>[https://www.ampproject.org/docs/getting\\_started/create/basic\\_markup](https://www.ampproject.org/docs/getting_started/create/basic_markup)

<sup>57</sup><https://www.cdnperf.com/>

<sup>58</sup><https://yuiblog.com/blog/2007/01/04/performance-research-part-2/>

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**<sup>59</sup>
- **AMP MOBILE INDEX.HTML ANALYSIS**<sup>60</sup>
- These do **NOT** use `window.onload!` **Refer to this article for more details on WHY!**<sup>61</sup>

## Compare your code

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

**<https://leanpub.com/c/p3gdc>**

## Mobile Single Web Page Applications

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 the 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`'s `<div>` tags. **The single web page application** is divided into `<div>` sections. Each `<div>` section represents a single phase menu and the `game.js` is placed into the "play game" `<div>`. 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.

---

<sup>59</sup><http://makingbrowsergames.com/book/ProjectIndex.pdf>

<sup>60</sup><http://makingbrowsergames.com/book/ProjectIndex-Mobile.pdf>

<sup>61</sup><https://javascript.info/onload-onDOMContentLoaded>

## Single Web Page Application (SWPA mobile)

---

```

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

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

&lt;div id="game-area" data-role="main" class="ui-content"&gt;
&lt;/div&gt;

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

&lt;div class="ui-content center footer" data-role="footer"&gt;
&lt;hr class="center" style="width: 60%" /&gt;
&lt;nav class="menu"&gt;&lt;a href=
'http://www.copyright.gov/fls/fl108.pdf' target='_blank'&gt; Copyright &lt;/a&gt;
    &amp;copy; 1978-2016, &lt;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'&gt; Stephen Gose LLC
&lt;/a&gt;. &lt;br /&gt; All Rights Reserved. &lt;br /&gt;
Questions or comments?
&lt;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/"&gt; Please Contact &lt;/a&gt;
&lt;br /&gt;
&lt;hr class="center" style="width: 60%" /&gt;
&lt;/nav&gt;
&lt;/div&gt;
&lt;/div&gt;
&lt;!-- End Game Page --&gt;
</pre>


---



```

You can see a “live example” at:



**Exercise:** Worksheet #1-2 Mobile index.html <http://makingbrowsergames.com/starterkits/quiz/game3/index-mobile-SWPA.html>

Sometimes, I like everything in one place; it depends on the size of the game I plan to deploy. If the game is small, I put all the raw game.js code into a <script> tag hosted inside the index.html. All that remains is a method to bind all these into a **single web page application (SWPA)**<sup>62</sup>. Using a single monolithic file has advantages per Google's new(?) **Accelerated Mobile Pages Project (AMP)**<sup>63</sup>. We'll do this through our game's index.html page. Many authors create yet another script file, but I prefer to use an inline scripting for mobile devices.

## 2.5 Task #2: Launching a Game

Phaser III, v2.6.2 and CE versions **are all launched from within a web page** as an inline JavaScript script or from an external file using JS modules. What happens next differentiates each Phaser version. **The official examples**<sup>64</sup> puts 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.

### Sample: Phaser v3.x.x 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?
 *
 */
```

---

<sup>62</sup><https://www.seguetech.com/what-is-a-single-page-application/>

<sup>63</sup><https://www.ampproject.org/>

<sup>64</sup><http://labs.phaser.io>



**Warning: Avoid “Anti-patterns”<sup>65</sup>** when developing your game source code. Bookmark the following **FREE** online book: **Essential JS Design Patterns by Addy Osmani<sup>66</sup>**. It shows what to look for and how to resolve “Anti-patterns” sneaking into the Phaser Libraries.

## Launching a Phaser III Game

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**.<sup>67</sup>

### 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));
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;

```

---

<sup>65</sup><https://addyosmani.com/resources/essentialjsdesignpatterns/book/#antipatterns>

<sup>66</sup><https://addyosmani.com/resources/essentialjsdesignpatterns/book/>

<sup>67</sup><http://makingbrowergames.com/p3gp-book/index12.html#12.3>

```
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     }
93
94     ; //Closes any previous scripts
95     //
96     // =====
97     // creates our Phaser Game configurations.
98     //    dozens of configurations parameters; see book
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: 'http://makingbrowergames.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: {
122             default: 'arcade',
123             arcade: {
124                 // Debug turned on for arcade physics
125                 debug: true
126             }
127         }
128     }
129
130     // Create a new Phaser.Game object
131     game = new Phaser.Game(gWidth, gHeight, Phaser.AUTO, 'game',
132     {
133         config
134     });
135
136     // Add a preloader
137     game.load.image('background', 'img/background.png');
138     game.load.image('player', 'img/player.png');
139     game.load.image('bullet', 'img/bullet.png');
140     game.load.image('explosion', 'img/explosion.png');
141
142     // Add a preloader
143     game.load.audio('explosion', 'audio/explosion.mp3');
144     game.load.audio('bullet', 'audio/bullet.mp3');
145     game.load.audio('background', 'audio/background.mp3');
```

```

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 // =====

```

---



**Exercise:** Download this example from: [http://makingbrowergames.com/p3gp-book/\\_p3demos/game.js](http://makingbrowergames.com/p3gp-book/_p3demos/game.js)



**Advanced Exercise:** Review advanced game setup using `_index.html` [http://makingbrowergames.com/p3gp-book/\\_indexp3.pdf](http://makingbrowergames.com/p3gp-book/_indexp3.pdf)



**Advanced Exercise:** Review advanced game setup using `_index-mobilep3.html` in the browser's console [http://makingbrowergames.com/p3gp-book/\\_index-mobilep3.pdf](http://makingbrowergames.com/p3gp-book/_index-mobilep3.pdf)

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?



**Advanced Exercise:** Compare your work **to another example.**<sup>68</sup> Open the Developer's Console and study what I've done with the "Game Object" name-space and "Phaser.Game" object in the console's drop-down menus. **This example also provides a timing test between `window.onload` VS. `document.onload`.**

<sup>68</sup><http://makingbrowergames.com/p3gp-book/standalone/>

I'm ***building a unique name-space***<sup>69</sup> for my game prototype in this second example. In ***Bob Nystrom's book, "Game Design Patterns"***,<sup>70</sup> 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. ..."



**Exercise:** Read about using the Singleton pattern in game design and development from ***Bob Nystrom's book, "Game Design Patterns"***.<sup>71</sup>



**Exercise:** Download this 3-page `main.js` example file I use as my standard prototype foundation: [http://makingbrowsergames.com/p3gp-book/\\_mainp3.pdf](http://makingbrowsergames.com/p3gp-book/_mainp3.pdf) and Refer to lines 112 to 150 in the file (you just downloaded? Right?).

## Deeper Dive: Launching the Game.

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

<sup>a</sup><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>

## Deeper Dive: Optional Game Config

Game Config has been around since before v2.4.2.<sup>72</sup> It's a JavaScript object that holds all the initial game configurations.

<sup>69</sup><https://javascriptweblog.wordpress.com/2010/12/07/namespacing-in-javascript/>

<sup>70</sup><http://www.gameprogrammingpatterns.com/singleton.html>

<sup>71</sup><http://www.gameprogrammingpatterns.com/singleton.html>

<sup>72</sup><https://labs.phaser.io/index.html?dir=game%20config&q=>

Open `main.pdf` (3-pages you downloaded above) and read lines 160 to 163. This waits for the browser to finish the Document Object Module (DOM) and then calls `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.

#### **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 ([http://makingbrowsergames.com/p3gp-book/\\_p3demos/](http://makingbrowsergames.com/p3gp-book/_p3demos/)) to the "**Official Phaser v3.x.x tutorial**"<sup>73</sup>

## Deeper Dive: To Infinity and Beyond!

Notice how Phaser v3.x.x uses a **configuration object**.<sup>74</sup> *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 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**<sup>75</sup>

---

<sup>73</sup><http://labs.phaser.io/>

<sup>74</sup><https://github.com/photonstorm/phaser/blob/dd39f9ab08d57fa1bacd1287ccadb03fb3151267/src/core/Game.js#L25>

<sup>75</sup><https://github.com/photonstorm/phaser/blob/dd39f9ab08d57fa1bacd1287ccadb03fb3151267/src/core/Game.js#L25>



**Warning:** There is a limit of **255 arguments passed into a JavaScript function per MDN.**<sup>76</sup>

## 2.6 Summary

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

Examples: [http://makingbrowsergames.com/p3gp-book/\\_p3demos/](http://makingbrowsergames.com/p3gp-book/_p3demos/)

- 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 from 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 client's enjoyment.
- Discovered which **CDN** have 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.

## 2.7 Chapter References:

(See more references)

- **MDN - Implementing game control mechanisms**<sup>77</sup>

<sup>76</sup><https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions>

<sup>77</sup>[https://developer.mozilla.org/en-US/docs/Games/Techniques/Control\\_mechanisms](https://developer.mozilla.org/en-US/docs/Games/Techniques/Control_mechanisms)

## 3. Building 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.<sup>1</sup> and

*"What's in a name? That which we call a rose, By any other name would smell as sweet."*

— William Shakespeare.<sup>2</sup>

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

### 3.1 Step #1 of 4: Generate a Game’s Phase

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 game code and then the ***“Gaming Loop’s event logic”***.

We also have a couple of choices in this construction Step #1 of 4. 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. Examples are:

- Our ***Breakout sample game***<sup>4</sup> we started in Chapter 1 — and will continue referring to it throughout this book.
- ***All the Official Phaser III Games examples***<sup>5</sup>

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Gertrude\\_Stein](https://en.wikipedia.org/wiki/Gertrude_Stein)

<sup>2</sup>[https://en.wikipedia.org/wiki/William\\_Shakespeare](https://en.wikipedia.org/wiki/William_Shakespeare)

<sup>3</sup>[https://en.wikipedia.org/wiki/Finite-state\\_machine](https://en.wikipedia.org/wiki/Finite-state_machine)

<sup>4</sup><http://makingbrowsergames.com/p3devcourse/standard/lesson15.html>

<sup>5</sup><https://labs.phaser.io/index.html?dir=games/&q=>

**Example 3.1: Creating Game Phase (traditional 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 this called this "init"
134     initialize: function(){
135         //stuff to generate this function
136         // debug header information
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:

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/game.js](http://makingbrowsergames.com/p3gp-book/_p3demos/game.js)

### 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.**<sup>6</sup>

---

<sup>6</sup><http://makingbrowsergames.com/book/index12.html#12.3>

**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,
    // Phaser v2.x.x this called this "init"; you must it ...
    initialize: function <GamePhaseName> () {
        Phaser.Scene.call(this, { key: '<GamePhaseName>' });
    },

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

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

---

**Deeper Dive: Creating Scenes using Phaser.Class**

Using the `Phaser.Class` is an interesting options. If you study the Phaser source code — <https://github.com/photonstorm/phaser/blob/v3.16.0/src/utils/Class.js> — on lines 166 to 192, 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.

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

You'll recall that I said in chapter 2, "I do not submit my game phases inside the `config` object. You can see how I inform Phaser about my game's phases in `main.pdf` lines 119 to 126. I add the game phases directly into my `this.game` variable. 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 phases file to load from one place — the index.html — and those phases will appear in my game without touching any code.***

#### 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
232     // Example 3.7: ends
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   // =====

```

---



**Exercise:** The example above refers to:  
[https://makingbrowsergames.com/p3gp-book/\\_p3demos/game.js](https://makingbrowsergames.com/p3gp-book/_p3demos/game.js) Lines 203 to 242

## Deeper Dive: Scene Transitions

DevLog #120 <https://phaser.io/phaser3/devlog/120>

### Sameple from v3.5.0

```

1  this.scene.transition({
2    //allowInput: false, // set true to enable input system
3    // of current scene and target scene
4    data: {x: x, y: y}, // an object containing any data you wish
5    //passed into target scene init methods.
6    duration: 1000, // in milliseconds
7    //moveAbove: true, // move the target Scene above this current
8    // scene before the transition starts
9    //moveBelow: true, // move the target Scene below this current
10   // scene before the transition starts
11   //onUpdate: null,
12   //onUpdateScope: scene,
13   //sleep: false, // set true to sleep this scene,

```

```

14          // set false to stop this scene
15      target: 'nextScene' //, the scene key name to transition into
16  });

```

---

## Deeper Dive: The CMS Game Shell

A **Content Management System (CMS)** game shell is merely a simple method toward building a **Progressive Web App (PWA)**<sup>7</sup> that reliably and instantly loads game content, similar to what you would see in native mobile applications.

The game shell is the minimal required technologies of HTML, CSS and JS to display the game's 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**”.<sup>8</sup> The game shell approach relies on aggressively caching the shell content using a **service worker**<sup>9</sup> to get the game up and running. Next, the dynamic game content and assets loads for each **game phase (aka roses)**. The secret sauce a game shell provides is: **Getting the initial HTML to the user’s device and displayed without the anything from the network!**

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 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 “data logic.”



**Note:** Try the **First Progressive Web App (PWA)**<sup>10</sup> to learn how to architect and implement your first generic mobile application shell. The “**Instant Loading with the App Shell model**”<sup>11</sup> video also walks you through this design pattern.

## Deeper Dive: When to use the game shell model

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

<sup>7</sup><https://developers.google.com/web/progressive-web-apps/>

<sup>8</sup><https://hbr.org/2016/06/the-go-to-market-approach-startups-need-to-adopt>

<sup>9</sup><https://developers.google.com/web/fundamentals/primers/service-worker/>

<sup>10</sup><https://codelabs.developers.google.com/codelabs/your-first-pwapp/#0>

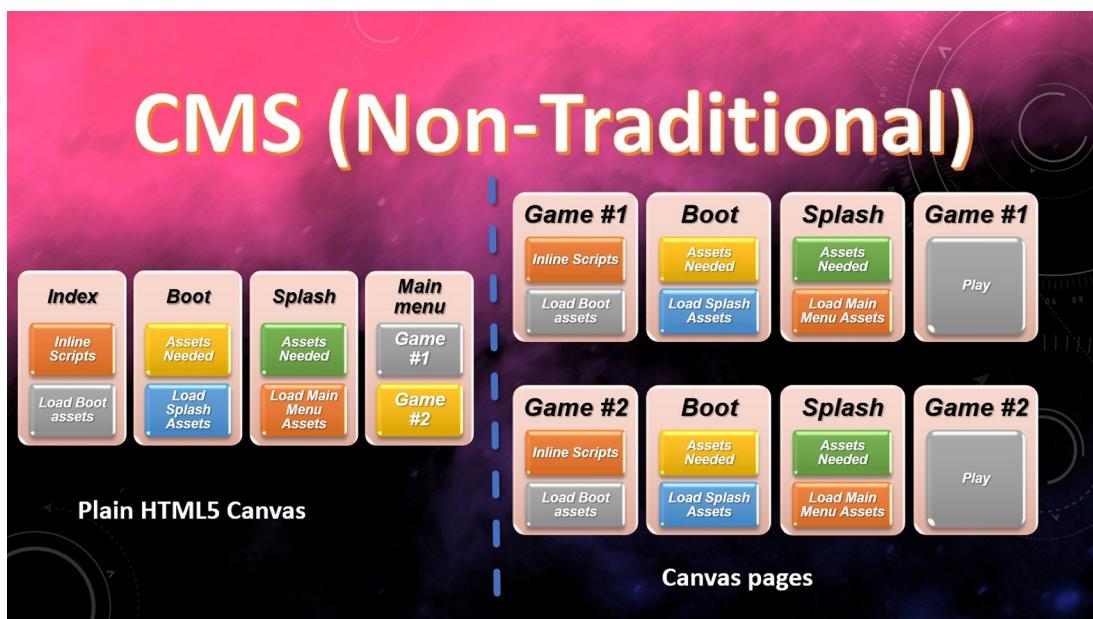
<sup>11</sup><https://www.youtube.com/watch?v=QhUzmR8eZAo>

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?!*** A number of modern 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 the majority of published **Phaser v2.x.x & v3.15+** games as single-page games described earlier; they use a single static canvas tag and within it are the various game ***“roses”*** (aka menus, movieClips, parts, phases, sections, stages, states, scenes, screens, ***thingies, dumaflache***,<sup>12</sup> or ***Ardvarks!***<sup>13</sup>) one is accustom to expect.



**Exercise:** Let’s see how Google builds regular mobile app shells. Take a look at ***Building the Google I/O 2016 Progressive Web App.***<sup>14</sup> This real-world mobile app started with a SPA to create a PWA that pre-caches content using a service worker, dynamically loads new pages, gracefully transitions between views, and reuses content after the first load.



One potential use of Phaser III CMS as separate game canvas.

<sup>12</sup><https://www.urbandictionary.com/define.php?term=dumaflache>

<sup>13</sup><https://en.wikipedia.org/wiki/Aardvark>

<sup>14</sup><https://developers.google.com/web/showcase/2016/iowa2016>

The benefits of using this game shell architecture and a **service worker**<sup>15</sup> are:

- 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 a 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.

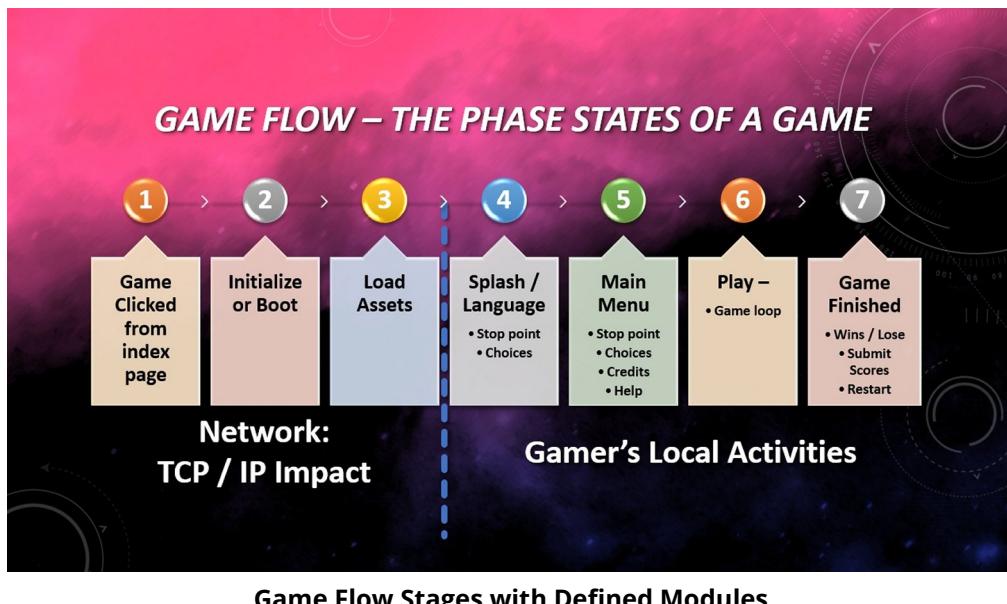
## 3.2 Bare Bones Prototypes

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

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

---

<sup>15</sup><https://developers.google.com/web/fundamentals/primers/service-worker/>



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 — ***in spite of which Phaser version used or whether we're using another Gaming Framework altogether!*** As the gamers migrate through our games, they have options; it doesn't matter **what we call** these "**rose**" **sections**<sup>16</sup> — phases, states, menus, scenes, screens, **thingies**, **dumaflache**,<sup>17</sup> or "**Ardvarks!**"<sup>18</sup> 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

<sup>16</sup><https://www.thefreedictionary.com/rose-like>

<sup>17</sup><https://www.urbandictionary.com/define.php?term=dumaflache>

<sup>18</sup><https://en.wikipedia.org/wiki/Aardvark>

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***<sup>19</sup> your game's 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.x.x reminds me more of a Adobe Flash MovieClip on its main time-line.

## Inside a Game Phase

Inside each of these phases, Phaser uses its internal "***Essential Functions***" (referred to as states in v2.x.x, scenes in v3.x.x; and Roses by everyone else ***with literary talents***). ***Many Phaser game developers, at this point, will create a new separate JavaScript file (i.e., a module) for each game phase and include these "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 agile software development when we begin to mix-and-match and re-use our "D.R.Y. code" (Don't Repeat Yourself code).

Each of these phases (aka, 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 within a game phase. For example, booting the game, loading assets, main menu, playing levels, winning, losing, ***all have their individually unique 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.

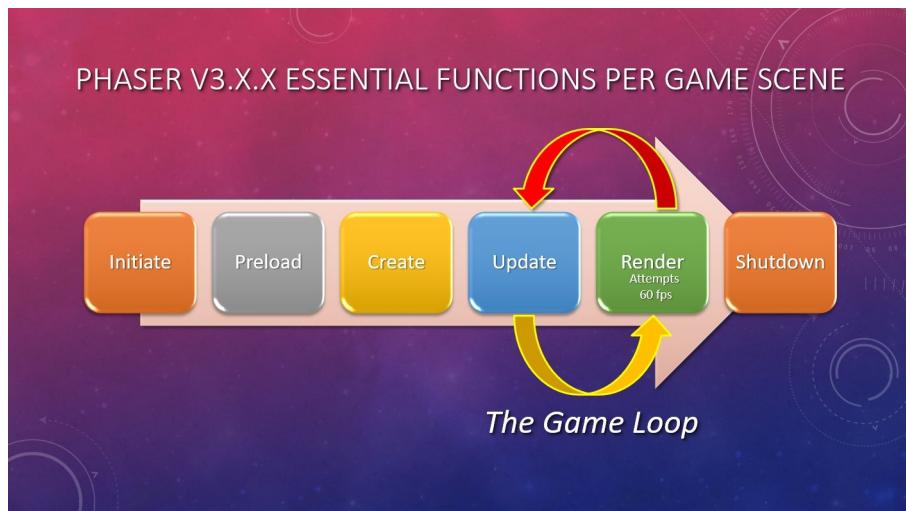
---

<sup>19</sup><https://www.vocabulary.com/dictionary/concoct>

### 3.3 Deeper Dive: “Phaser Essential Functions”



**Note:** “... Essential Functions ...” is not a term I invented! ***It comes directly from Official Phaser tutorials!***<sup>20</sup> So folks, ***let's call a spade a spade, and not a gardening tool!***<sup>21</sup> shall we?!?!



Phaser Essential Function in a single game phase

Phaser JavaScript Game Framework —***all versions!***— uses several code functions to process each game phase. ***Quoted from “Making Your First Phaser Game”***<sup>22</sup> — “Phaser supports a ***full State system*** allowing you to break your code into much cleaner ***JavaScript single-objects. (ed.: illustrated above.)*** 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 phase’s life-cycle***. The `create()` function places all the game prototype entities and components inside an HTML5 canvas; the `update()` function attempts to refresh the game’s display 60 times per second.

***Phaser Scene Constants:***<sup>23</sup> Each Game Phase also has these additional functions (***sorted alphabetically with its sequence of execution***):

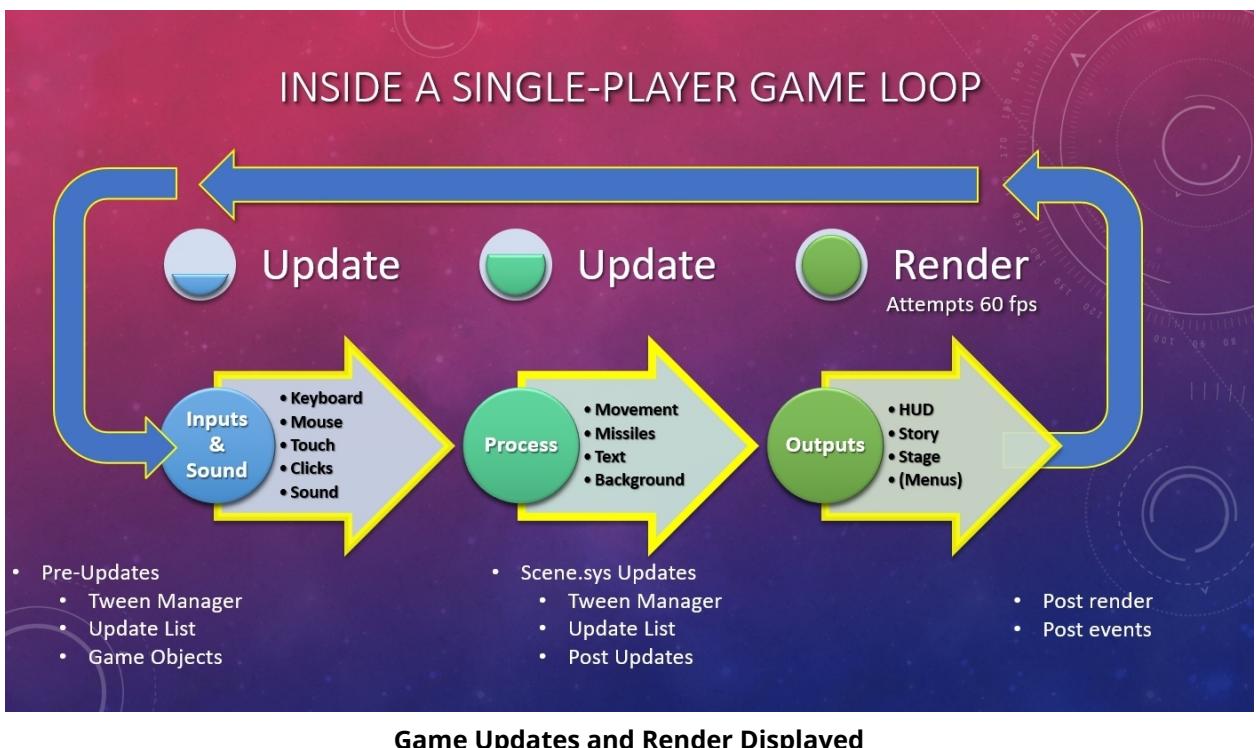
<sup>20</sup><https://phaser.io/tutorials/making-your-first-phaser-2-game>

<sup>21</sup>[https://en.wikipedia.org/wiki/Call\\_a\\_spade\\_a\\_spade](https://en.wikipedia.org/wiki/Call_a_spade_a_spade)

<sup>22</sup><http://phaser.io/tutorials/making-your-first-phaser-2-game>

<sup>23</sup><https://github.com/photonstorm/phaser/blob/v3.12.0/src/scene/const.js>

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



Flow chart of **Phaser III Game Loop**.<sup>24</sup>

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

- any player's inputs from the keyboard, touch and/or mouse clicks;

<sup>24</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/mainloop/#flow-chart>

- 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.<sup>25</sup>

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

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

<https://phaser.io/phaser3/devlog/121> describes the new capabilities that `Phaser.Scene` offers. In my mind and coming from an Adobe Flash background (as Davey has too!), ***the new behavior of Phaser III Scenes reminds me of Flash MovieClips on the timeline.*** Other commonly used “Essential Functions” inside of a `Phaser.Scene` are ***charted here***<sup>26</sup>:

1. ***initialize***<sup>27</sup> – A method called when any state starts. It is passed as an argument variable to facilitate data-sharing among the states. ***It must be called initialize or you risk adopting the default Phaser III object.***
2. ***preload***<sup>28</sup> – A method called whenever any state begins. It is used for loading your game resources and assets prior to their use. Normally, in Phaser III, you’d load your game assets just for the current phase. If you call any `this.load` methods from outside of `Scene.preload` function, then you will 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 preload. Each scene in Phaser III now has its own Load manager.*** Since the load scene may take up to 20 seconds, it a good time to use some “Splash scene ideas”. A Splash scene show the game’s title, sponsors, legal copyright notices. You should build the gamer

<sup>25</sup><https://www.youtube.com/watch?v=GibiNy4d4gc>

<sup>26</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/scene/#flow-chart>

<sup>27</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/Systems.js#L297>

<sup>28</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/SceneManager.js#L747>

anticipation for exciting game play. 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.

3. **create**<sup>29</sup> – A method called automatically after preload finished; it is used for generating game objects (See Ch 5 Game mechanics). If you didn’t actually load any game assets at all or didn’t use the preload function, create is the first function executed by the engine. Use create to set-up the bulk of your downloaded game assets.
4. **update**<sup>30</sup> – 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 (See Ch 5 Game mechanics). This method is often used to capture game events, such as key presses, button clicks, mouse movement etc, and to then update the game state variables as a result of those inputs. It is called on every frame. The engine attempts to execute, at best efforts, 60 times per second, but that is not guaranteed. It is called BEFORE Stage, Tweens, Sounds, Input, Physics, Particles, and Plugins have had their “post-Update” methods called.
5. **render**<sup>31</sup> 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.
6. **shutdown**<sup>32</sup> – A method called when a state shuts down (i.e. you switch to another game state menu from current one). You could consider this a “garbage collection” routine which cleans up orphaned game objects.



**Warning:** Remember JavaScript is a prototype-based language. If you try to store these states 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 state for each object.<sup>33</sup>

<sup>29</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/SceneManager.js#L295>

<sup>30</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/SceneManager.js#L537>

<sup>31</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/SceneManager.js#L584>

<sup>32</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/scene/SceneManager.js#L1149>

<sup>33</sup><https://medium.com/javascript-scene/3-different-kinds-of-prototypal-inheritance-es6-edition-32d777fa16c9#.iy7efb917>

## 3.4 Game Phase Prototypes

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 write a game phase for our game shell, we're done! We never have to touch it again — unless you want muck around with some tweaks***<sup>34</sup> — such as small unique modifications or features we might want to include inside the core "Game Play" phase. Each game phase could become ***siloed***<sup>35</sup> as a separate JS module file and prototype! We can ***mix, match and arrange our "rose" bouquet***<sup>36</sup> ***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***<sup>37</sup> — ***this is the secret sauce for cranking out a game per week!*** (Refer back to chapter 1)

Vanilla, Strawberry, or Chocolate Creme-filled?

### Sample: Step 3) Vanilla Game Phase 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 this 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);
    }
}
```

<sup>34</sup><https://www.collinsdictionary.com/us/dictionary/english/muck-around>

<sup>35</sup><https://en.oxforddictionaries.com/definition/siloed>

<sup>36</sup><https://www.merriam-webster.com/dictionary/bouquet>

<sup>37</sup><https://www.gamedevmarket.net/?ally=GVgAVsoJ>

```

}, //comma is very important

/** Create 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/

```

---

## Deeper Dive: Overriding Scenes from Phaser.Scene

```

1   // NOTE: this style could be applied to ANYTHING inside Phaser!
2   var demo = new Phaser.Scene('Demo');
3
4   // Phaser v2.x.x this called this "init"
5   demo.initialize = function initialize (data){ ... };
6   demo.preload     = function preload (){ ... };
7   demo.create      = function create (data){ ... };
8   demo.update      = function update (time, delta){ ... };

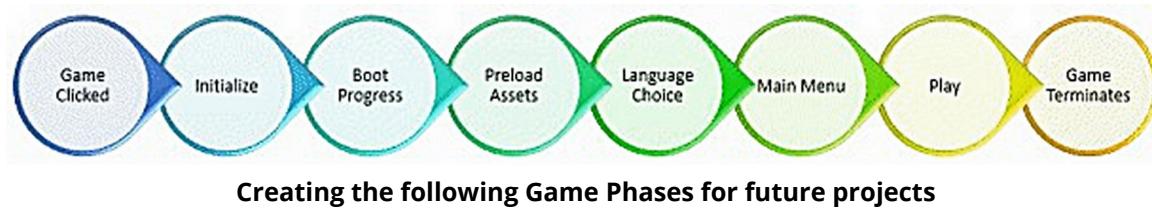
```

## Deeper Dive: Creating Scenes using ES5 Prototypes

```

1  var MyGame = { ... };
2
3  MyGame.Boot = function () {
4      //stuff boot game phase performs.
5  };
6
7  MyGame.Boot.prototype.constructor = MyGame.Boot;
8
9  MyGame.Boot.prototype = {
10     // Phaser v2.x.x this called this "init"
11     initialize: function initialize (data) { ... },
12     preload:    function preload () { ... },
13     create:     function create (data) { ... },
14     update:     function update (time, delta) { ... }
15 };

```



**Creating the following Game Phases for future projects**



**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.<sup>38</sup>



**Exercise:** Study the various combinations of creating JavaScript **Object creation patterns**<sup>39</sup> and styles from **Code reuse patterns**<sup>40</sup>

<sup>38</sup><http://makingbrowsergames.com/p3gp-book/index9.html>

<sup>39</sup><https://www.jspatterns.com/category/patterns/object-creation/>

<sup>40</sup><https://www.jspatterns.com/category/patterns/code-reuse/>

### Sample: Step 3) Chocolate Game Phase in TypeScript

---

```
//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:**<sup>41</sup> "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 development of large applications and **transcompiles to JavaScript (ed.: ES6!).**"<sup>42</sup> Per **this website**<sup>43</sup> and after 6 years from initial release, only 24% of web developers use Typescript.

## Deeper Dive: 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 });

```

## ES6 Considerations: Strawberry

---

<sup>41</sup><https://en.wikipedia.org/wiki>TypeScript>

<sup>42</sup><https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/>

<sup>43</sup><https://www.jetbrains.com/research/devecosystem-2018/javascript/>

### Sample: Step 3) ES6 Strawberry 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*<sup>44</sup>

Compare the “4-year-old 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**.<sup>45</sup> 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*.<sup>46</sup>



**Hint:** The current JavaScript standard is ES9 (as of 2018).

---

<sup>44</sup>[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object\\_prototypes](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes)

<sup>45</sup><https://kangax.github.io/compat-table/es6/>

<sup>46</sup><http://brianchang.info/2016/01/23/how-to-future-proof-your-code.html>

## Deeper Dive: Separate 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:

```

1  var config = {
2      key: '',
3      // active: false,
4      // visible: true,
5      // pack: false,      //see JSON file pack below
6      // cameras: null,
7      // map: {},
8      // physics: {},
9      // loader: {},
10     // plugins: false,
11     // input: {}
12 };
13
14 // JSON file pack example
15 // use: scene.load.pack(key, url, dataKey);
16
17 {
18     'dataKey': {
19         // "prefix": "...",    // optional, extend key by prefix
20         // "path": "...",     // optional, extend url by path
21         // "defaultType": "image", // optional, default file type
22         'files': [
23             {
24                 'type': 'image',
25                 'key': '...',
26                 'url': '...'
27             },
28             {
29                 'type': 'image',
30                 'key': '...',
31                 'url': '...'
32             }
33             // ...
34         ],
35     },
36
37     'node0': {

```

```

38     'node1': {
39         'node2': {
40             'files': [
41                 // ....
42             ]
43         }
44     }
45 }
46 // dataKey: 'node0.node1.node2'
47 }
```

## Deeper Dive: Defining Other Game 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 Phaser v2.x.x, Game Objects (such as Sprites) can 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.x.x, 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 have their own instance of a 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 Phaser III Game Design Workbook<sup>47</sup> 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:

### 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(
218     'http://makingbrowsergames.com/p3devcourse/_p3demos/imgages/');
219 console.log("Additional Phaser Properties set in preload!");
220 //Example 3.4: ends
```

---

<sup>47</sup><https://leanpub.com/phaser3gamedesignworkbook>

```

221 // =====
222
223 //Set a neutral background color
224 //notice we used the shorthand version; instead of #FF0000
225 //photonstorm/phaser/blob/v3.14.0/src/boot/Config.js#L532
226 //This sets up Phaser's Arcade physics engine,
227 // which are simple but effective for arcade-style games.
228 //this.physics.startSystem(Phaser.Physics.ARCADE);
229 //this.renderer.renderSession.roundPixels = true;
230
231 //this applies physics to every item.
232 //enable(object, [bodyType array or group])
233 //enableBody(object, DYNAMIC_BODY) or
234 //enableBody(object, STATIC_BODY)
235 //this.physics.world.enableBody(key, CONST.DYNAMIC_BODY);
236
237 //OR . .
238 this.scene.physics.world.enable(this);
239
240 console.log("Additional Phaser Properties set in preload!");
241 //Example 3.4: ends
242 // =====
243 },
244
245 //Recommended scaling for Phaser III
246 // managed by CSS.
247 function resize() {
248     var canvas = document.querySelector("gameCanvas");
249     var gWidth = window.innerWidth;
250     var gHeight = window.innerHeight;
251     var gRatio = gWidth / gHeight;
252     var gameRatio = config.width / config.height;
253
254     if (gRatio < gameRatio) {
255         canvas.style.width = gWidth + "px";
256         canvas.style.height = (gWidth / gameRatio) + "px";
257     } else {
258         canvas.style.width = (gHeight * gameRatio) + "px";
259         canvas.style.height = gHeight + "px";
260     }
261 };

```

---



**Note:** Example above comes from: [http://makingbrowergames.com/p3gp-book/\\_p3demos/game.js](http://makingbrowergames.com/p3gp-book/_p3demos/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)**<sup>48</sup> — “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**<sup>49</sup> — “... 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**<sup>50</sup> — “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`.”



**Exercise:** Review completed examples in the **Phaser III Game Prototype Library on the book's website.**<sup>51</sup>



**Exercise:** Review the Phaser’s Official examples.<sup>52</sup>



**Exercise:** Learn more about the various Phaser physics engines in the documentation.<sup>53</sup>



**Exercise:** Research these references on using multiple physics engines at the same time. Which Physics System To Chose?<sup>54</sup>

---

<sup>48</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Arcade.html>

<sup>49</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Impact.html>

<sup>50</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Physics.Matter.html>

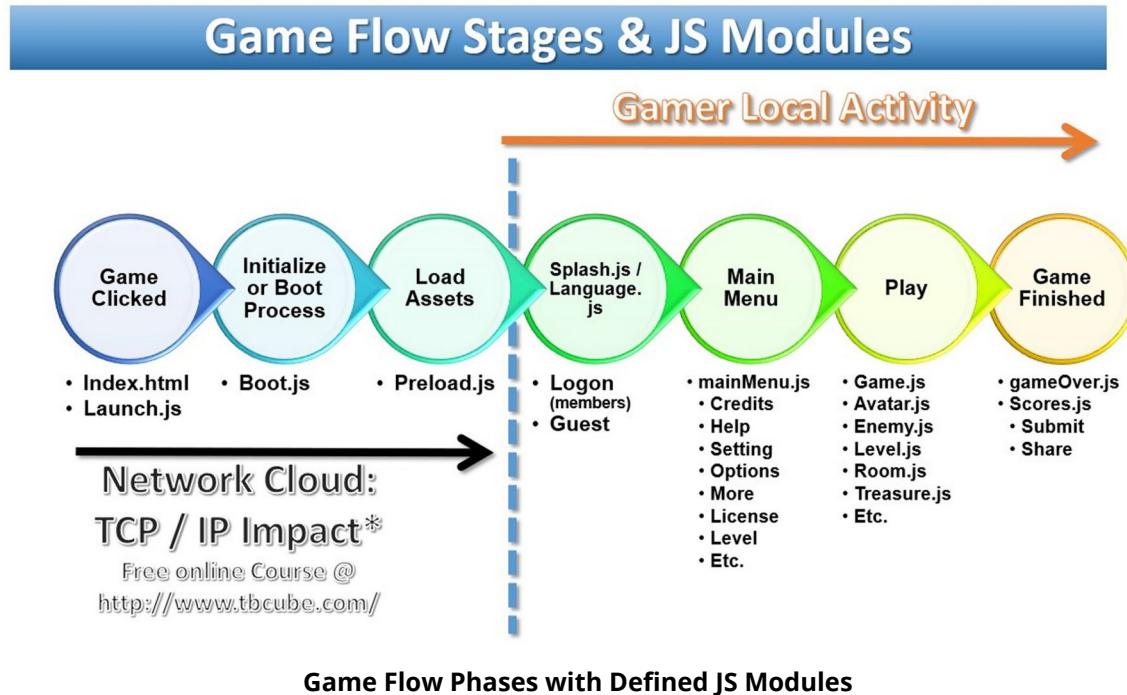
<sup>51</sup><http://makingbrowergames.com/p3gp-book/>

<sup>52</sup><http://labs.phaser.io/index.html?dir=physics/&q=>

<sup>53</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Physics.html>

<sup>54</sup><http://www.html5gamedevs.com/topic/4503-which-physics-system-to-choose/>

### 3.5 Game Phases as Modules



We'll follow this chart in the order of appearance. The only phases we need to revisit, **tweak**<sup>55</sup> and validate are these:

- **main.js (aka launch):**<sup>56</sup> This holds all the particular configurations for our game(s). Inside the `index.html`, it is the next file loaded immediately after Phaser. However, if we build a similar game mechanics, this file should be relatively **D.R.Y**
- **boot.js:**<sup>57</sup> will inspect the display dimensions and adjust sizing. This game phase could be integrated with the HTML index page. It lists all the game assets needed immediately. You can “control” its behavior from within the `config` object — which we placed inside either the `index.html` or `main.js` file.
- **preload.js:**<sup>58</sup> will download listed game assets into a queue for parallel downloading. It will place the game assets into the `cache` for immediate use for the current game phase.

<sup>55</sup><https://www.urbandictionary.com/define.php?term=tweak>

<sup>56</sup><http://makingbrowsergames.com/p3gp-book/index3.html#9.1>

<sup>57</sup><http://makingbrowsergames.com/p3gp-book/index3.html#9.2>

<sup>58</sup><http://makingbrowsergames.com/p3gp-book/index3.html#9.3>

- **splash.js or language.js:**<sup>59</sup> will adapt text information to the gamer's chosen language. The more standardized these screens are ... the better! — resulting in a saving in our development time. (**NOTE:** this is a "stop point" to force our gamers to make a decision. The amount of time they take provides us with more time to delivery game assets.)
- **mainMenu.js:**<sup>60</sup> central decision point for the gamer. The more standardized this screen is ... the better. (**NOTE:** Again, another "stop point")
- **play.js:** and finally, of course, all our specific game mechanics, core and supporting functions for any specific game project.
- **gameOver.js:**<sup>61</sup> administrative and record keeping functions.



**Hint:** Here's a **game programming flow chart**<sup>62</sup> located in your bonus download content or from here: [http://makingbrowergames.com/starterkits/\\_GameFlowChart.pdf](http://makingbrowergames.com/starterkits/_GameFlowChart.pdf); it demonstrates the same concepts as pictured above. **Notice it has nothing to do with any Phaser version we're using!!**

## Deeper Dive: ES9 Modules

### **Why mentions this??**

Because is the most recent release of Phaser III v3.16.2 (newsletter #139 20190211), we have "**scenefiles**". "**Scene Files**"<sup>63</sup> 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');
```

It's important that the key given is 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.

<sup>59</sup><http://makingbrowergames.com/p3gp-book/index3.html#9.4>

<sup>60</sup><http://makingbrowergames.com/p3gp-book/index3.html#9.5>

<sup>61</sup><http://makingbrowergames.com/p3gp-book/index3.html#9.6>

<sup>62</sup>[http://makingbrowergames.com/starterkits/\\_GameFlowChart.pdf](http://makingbrowergames.com/starterkits/_GameFlowChart.pdf)

<sup>63</sup><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;
});
```

---

“Phaser.Game” – One File to Rule them all ...

***Ash nazg durbatulûk, ash nazg gimbatul, ash nazg thrakatulûk, agh burzum-ishi krimpatul<sup>64</sup>***

(**Note:** Do NOT utter these words aloud .... you've been warned!)

### Yes, Gandalf got it wrong!

Actual literal translation from the **grimoire — “Lore of Phaser v3.x.x”<sup>a</sup>** is:

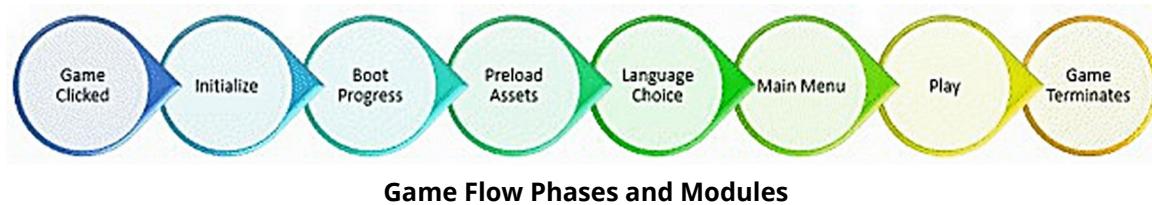
*One File to rule them all — (Phaser.Game the God-class!),*

*One File to find them — (Phaser.Boot),*

*One File to bring them all — (Phaser.Load),*

*and in the darkness bind them! — (Phaser.Scenes)*

<sup>a</sup><https://en.wikipedia.org/wiki/Grimoire>



We'll follow this chart in the order of appearance.

---

<sup>64</sup><https://www.youtube.com/watch?v=IMSLM33PQDM>

## Main.js (aka “launch” or index.js)

Let’s review each JavaScript file in the skeleton header. The `main.js` file “IS” the game’s foundation and configuration information. It is the first external JS game file loaded, after the Phaser III framework, in the `index.html`. It holds all the game settings used during the upcoming `play` phase. 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 `main` source code; it is thoroughly annotated and documented to reduce the price of this book.

C>[http://makingbrowergames.com/p3gp-book/\\_p3demos/js/mainp3.pdf](http://makingbrowergames.com/p3gp-book/_p3demos/js/mainp3.pdf)



**Hint:** Review the two demonstration games at <http://makingbrowergames.com/starterkits/> and open the browser console to watch the Phaser Game Framework in the Developer Console.

## Boot.js

Our `boot` 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. (See Ch. 5 Game mechanics) As its name suggests, its purpose prepares the web browser and sets the game dimensions — loading various game assets and storing them in the `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.

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, button 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 works. 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 a example “11-page file” from  
[http://makingbrowsergames.com/p3gp-book/\\_p3demos/js/state/boot.js](http://makingbrowsergames.com/p3gp-book/_p3demos/js/state/boot.js)



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



**Hint:** To further preserve this files integrity **and keep it D.R.Y**, you might consider having the `boot.js` simply upload a JSON data file with all the game resource unique to this project.

## **Preload.js**

This Game Phase manages our number of files to download; you should optimize this process with the fewest downloads immediately required by your game. In a normal CodeIgniter CMS game, I “inline” the normal `boot.js` into the `index.html` and consolidated everything else into a `game.js`; by doing so, I have deferred several potential downloaded files with this single combined file.

Many developers use **Browserify**<sup>65</sup> to the same effect. The formal and separate `preload.js` now becomes a simple JavaScript object in a single web page application (SWPA) Illustrated below is the “preload” process inside the “game.js”.

<sup>65</sup><http://browserify.org/>



**Note:** You can download an example “1-page file” from  
[http://makingbrowsergames.com/p3gp-book/\\_p3demos/js/state/load.js](http://makingbrowsergames.com/p3gp-book/_p3demos/js/state/load.js)

## Deeper Dive: Security

The most exciting thing is the game assets protection now available by using the “set Base URL”. Look at the `setBaseURL`. You should see inside the `preload` function a statement such as

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

When you run this `html` page all the assets are loaded directly from your domain’s designated directories. This is game asset security at its **BEST!** You maintain control of all gaming assets while the games are in the “wild”. Simply modifying your game assets dynamically 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 CDN). Naturally, you must have **Cross-Origin Resource Sharing (CORS)**<sup>66</sup> enabled.

## Deeper Dive: 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.

```

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();
```

---

<sup>66</sup><https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

You can manage `cache` contents 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.

## 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 compartments (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*<sup>67</sup> is a **free software package** specifically for creating tiled maps. Another **licensed application** is Texture Packer<sup>68</sup> that will also help you create sprite sheets and their atlases. Texture Packer could create tile maps also. Shoebox<sup>69</sup> is similar to Texture Packer except that it is **FREE**.

<sup>67</sup><http://www.mapeditor.org>

<sup>68</sup><https://www.codeandweb.com/texturepacker>

<sup>69</sup><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 the loading process. It is responsible for queue management, dispatching events, and load management. The `BaseLoader` class handles the following 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



**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 `\\"` — are ignored.

### Samples from [phaser.io/phaser/api/loader](https://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 phase — the first stopping point 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(?), offer language selections and present your own logo 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 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. Again, a word of caution here, all “app store” require your game to be “live and active” within 20 seconds.

What I do is present a “language menu” to our gamers and let them select — ***dynamically on demand!!*** — their native language for continued game-play and interaction. We will not download every language lexicon known to mankind at this point; and, of course, ***we'll steer clear of “Enochian”***<sup>70</sup> ... we'll have none of that here; they can go “somewhere else and play”.

The natural choice for “language selection” is a 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, a tool-tip text changes into

---

<sup>70</sup><https://en.wikipedia.org/wiki/Enochian>

that nation's predominant language. Mesmerized by the sudden display of various languages and spellings, our gamers — **doing what they do best** (i.e.: to 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")***<sup>71</sup>. 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 a visual clue. On that click-event, the internal game functions will send a request to download that specific JSON language file that will 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!***<sup>72</sup>



**Note:** More about international targeting in the Marketing & Distribution chapter.



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

Live language demonstrations: ***Ruins of Able-Wyvern™ (as pictured above)***<sup>73</sup>

<sup>71</sup><http://makingbrowergames.com/p3gp-book/standalone/index.html>

<sup>72</sup><http://www.internetworldstats.com/stats.htm>

<sup>73</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos-arrav15/index.html](http://makingbrowergames.com/p3gp-book/_p3demos-arrav15/index.html)

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 merely `<div>` tags using Bootstrap CSS!

Modifications for the `Splash.js` or `Language.js` should be minimal as long as you use a consistent information and ***menuing system***<sup>74</sup> across all your games. Loading standard game phase menus, images, buttons are already inside your ***Bonus Content/standalone***<sup>75</sup>. 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`.<sup>76</sup>

## Menu.js

Our next phase is the game's "main menu" — the central hub of all game activity. It is during this game phase we now load the language text for all tool-tips, feedback and menus.

### Reference

- |   |  |
|---|--|
| 1 | preload function — not used; everything was downloaded in the <code>boot.js</code>         |
| 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**

C>[http://makingbrowergames.com/p3gp-book/\\_p3demos/js/state/menu.js](http://makingbrowergames.com/p3gp-book/_p3demos/js/state/menu.js)

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 ***in a content management system (CMS)*** rather than stuffing everything inside a single game `canvas`. That would merely bloat your final gaming file. Remember the HTML5 `canvas` tag is merely a graphical display; in

---

<sup>74</sup>[https://en.wikipedia.org/wiki/Menu\\_\(computing\)](https://en.wikipedia.org/wiki/Menu_(computing))

<sup>75</sup><http://makingbrowergames.com/p3gp-book/standalone.zip>

<sup>76</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/js/state/language.js](http://makingbrowergames.com/p3gp-book/_p3demos/js/state/language.js)

essence, it replaces the former Adobe Flash plugin. (**NOTE:** End of life for Flash plugins is 20201231) Visit **other games on the book's website** for examples of a **content management system (CMS)**. Visit as examples of a Content Management System (CMS):

- *Rulers of Renown™: The Emancipation*<sup>77</sup>
- *Adventurers of Renown™: The Ruins of Able-Wyvern*<sup>78</sup>
- *Adventurers of Renown™: The Blood Pit*<sup>79</sup>
- *Adventurers of Renown™: The Rogue Prince*<sup>80</sup>
- *Adventurers of Renown™: The Rescue of NCC Pandora*<sup>81</sup>



Simple CMS or PWA game-shell

<sup>77</sup><http://www.adventurers-of-renown.com/quests/rrte.php>

<sup>78</sup><http://www.adventurers-of-renown.com/quests/arra.php>

<sup>79</sup><http://www.adventurers-of-renown.com/quests/arbp.php>

<sup>80</sup><http://www.adventurers-of-renown.com/quests/arrp.php>

<sup>81</sup><http://www.adventurers-of-renown.com/quests/arnp.php>

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 the game and develop customer loyalty and community. So, why should we bloat and clutter our Phaser game with ***superfluous information***.<sup>82</sup> Example of such pages 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 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.
- *MoreGame.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. Live example at <http://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 viral distribution of your game or announcements within the game. ***See the twitter enhancement***<sup>83</sup>.
- *SubmitScores.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.

---

<sup>82</sup><https://www.merriam-webster.com/dictionary/superfluous>

<sup>83</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3twitterEnhancementJS.pdf](http://makingbrowsergames.com/p3gp-book/_p3twitterEnhancementJS.pdf)



## Play.js

Finally, we arrive at what this book is all about — ***the game phase!*** This phase dives straight into creating the game's **entities and components**. 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"!***

**NOTE:** You can refer to the "Skeleton Game Phase". Download from  
C>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/js/state/play.js](http://makingbrowsergames.com/p3gp-book/_p3demos/js/state/play.js)

## Deeper Dive: JS Modules

Nearly every programming language has a concepts 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**<sup>84</sup> 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



**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**"<sup>85</sup> by Addy Osmani.

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

---

<sup>84</sup><https://www.openmymind.net/2012/2/3/Node-Require-and-Exports/>

<sup>85</sup><http://addyosmani.com/writing-modular-js/>

**Browser JavaScript modules verses 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	<b>local to module</b> <sup>86</sup>	global



**Note:** If you're not familiar with "JavaScript modular files", read this chapter about "**Modules**"<sup>87</sup> from the **FREE online book "Eloquent JavaScript"** by Marijn Haverbeke<sup>88</sup> or you might review **Preeti Kasireddy's superior article**.<sup>89</sup>

**QUOTE from Phaser III Game Design Workbook<sup>a</sup>**

"Development source code is what you read and write, and "check-in" to your source control system such as GitHub.<sup>b</sup> 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 —see **Google Developer: Setup Your Build Tools**<sup>c</sup> and later chapters — is a step in which you apply these transformations; many developers use the automated "Grunt".<sup>d</sup> Finally, your web server should deliver the machine code with gzip compression for extra speediness." Read more tips **here**.<sup>e</sup>

<sup>a</sup><https://leanpub.com/phaser3gamedesignworkbook>

<sup>b</sup><https://github.com/>

<sup>c</sup><https://developers.google.com/web/tools/setup/setup-buildtools>

<sup>d</sup><https://24ways.org/2013/grunt-is-not-weird-and-hard/>

<sup>e</sup><https://sunpig.com/martin/2008/10/07/maintainable-css-modular-to-the-max/>

## 3.6 Summary

### Examples:

<sup>86</sup><https://www.openmymind.net/2012/2/3/Node-Require-and-Exports/>

<sup>87</sup>[https://eloquentjavascript.net/10\\_modules.html](https://eloquentjavascript.net/10_modules.html)

<sup>88</sup><https://eloquentjavascript.net/index.html>

<sup>89</sup><https://medium.freecodecamp.org/javascript-modules-a-beginner-s-guide-783f7d7a5fcc>

- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowsergames.com/p3gp-book/_p3demos/bareBonesIndex.html)
- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/index.html](http://makingbrowsergames.com/p3gp-book/_p3demos/index.html)

Chapter 3 completed! Here's what we 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.

### 3.7 Chapter References:

(*See more references*)

- **Plain English Guide to JavaScript Prototypes**<sup>90</sup>
- **JavaScript Classes**<sup>91</sup>

---

<sup>90</sup><http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/>

<sup>91</sup><https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

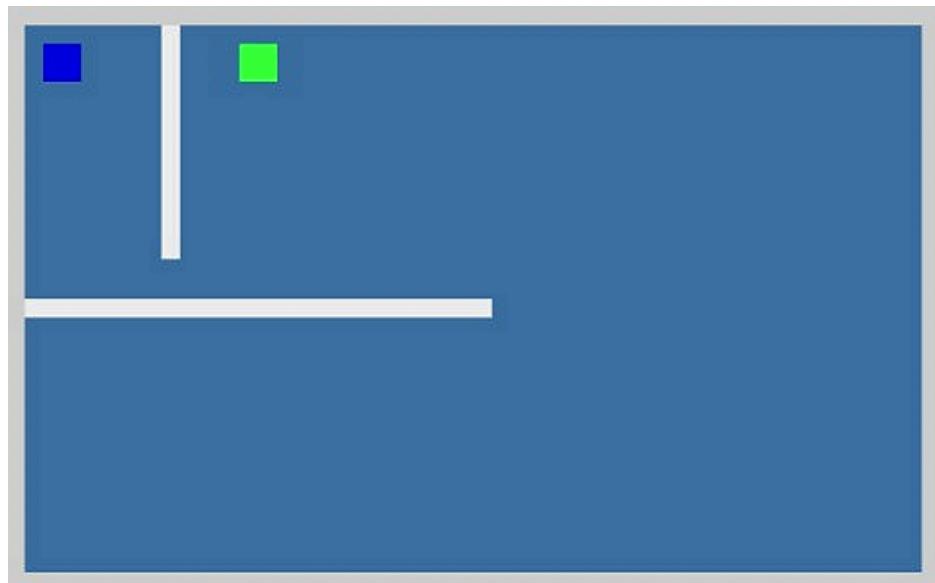


## 4. Building Game Prototypes, Mechanisms & Tools

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.***<sup>1</sup>

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 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

---

<sup>1</sup><http://leanpub.com/p3gskc/>



**Exercise:** Download the resources from [http://makingbrowergames.com/p3gp-book/\\_p3demos.zip](http://makingbrowergames.com/p3gp-book/_p3demos.zip) or start a new project following Task #1 & #2 from previous chapters.

## 4.1 Task #3: Mini-Me

To represent a player in our game, we need to define two things:

1. a **visual** sprite image in the game which represents our gamer and their location in relation to things on the stage; and
2. a **separate data storage object** describing that avatar's characteristics and skills.



**Note:** We've already discussed the "**Principles of Software Engineering**"<sup>2</sup> and the "**Separation of Concerns**"<sup>3</sup> as effective software engineering principles in Chapter 1.

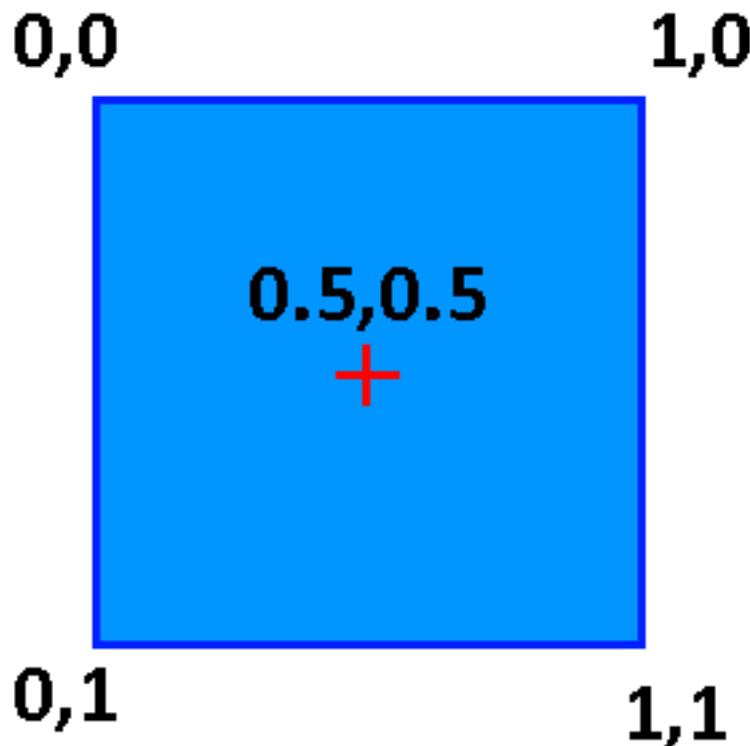
### Creating an Avatar (the visual display)

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 and in Chapter 2 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 Game Object (GO) in Phaser III, you must remember that the visual element is "centered" by default. In Phaser v2.x.x, we used setting `anchor(0.5,0.5)` to have the "anchor" (aka origin point) in the center. If you do not want all visual objects "centered", then use:

<sup>2</sup>[http://makingbrowergames.com/design/\\_PrinciplesofSoftwareEngineering.pdf](http://makingbrowergames.com/design/_PrinciplesofSoftwareEngineering.pdf)

<sup>3</sup><https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/>



`.setOrigin(x, y);`

### Review Example 2.1: 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: 0xFFFFFFF}));
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: 0xFFFFF}
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 // =====

```

---



**Note:** Refer to these resource files: [http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson04.html](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/lesson04.html)

The `main >> create >> this.player = this.add.sprite` experiment — commented out — placed a box shape off screen from the top and left. We drew this image using the `box` function with using the parameters delivered to it with the starting coordinates `(150,100, ...)` and its dimensions `(... options.length, options.width)`. The “Blue avatar” used just the `box` function; the moving “Red avatar” is another method. The Phaser III game uses a coordinate system just like the ones used in CSS positioning; the upper left corner of the game world is `(x=0,y=0)`. Finally, we filled the box image with

our designated colors. Refresh the browser and you should see both the player and another avatar box.



**Note:** Review completed examples in this chapter's resource file: [http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/)

There are other clever v3.x.x methods at Phaser.io<sup>4</sup> we could have used; those are found in the Phaser.io v3 examples.<sup>5</sup> We can reuse our new `box` function to define any colored rectangle as a game object prototype.



Sample image from [text-image.com](http://text-image.com). See! Base64 Retro is TOO SEXY!

I use these two favorite resources to build sprites and spriteSheets for my games.

- **Charas** — the online resource generators <http://www.charas-project.net/charas2/index.php> and review their sample games at <https://charas-project.itch.io>. Charas Project is a community founded in 2003 by Alex. It's a community for 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<sup>6</sup> they share their community games and projects.

<sup>4</sup><http://labs.phaser.io/index.html?dir=game%20objects/&q=>

<sup>5</sup><http://labs.phaser.io/index.html?dir=game%20objects/shapes/&q=>

<sup>6</sup><https://charas-project.itch.io>

- **Universal LPC Sprite Sheet** — Create a character sprite sheet for your games using 100% open art. <http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/>. I'm using LPC sprites for my own game and enhancing the universal spriteSheet with new changes and fixes. **GitHub repository**<sup>7</sup> modified for our game prototypes.

Deeper Dive: Display selected sprite from sprite-sheet.

## Displaying a particular sprite from sprite-sheet

**Quote:**<sup>a</sup> Phaser III has support for two types of sprite sheet: “**classic**” ones, where every frame is the exact same size, and “**texture atlases**” which are created with the help of a third party app like Texture Packer, Shoebox or Flash CC and come with **an associated JSON file**.

You load the “**classic**” ones with `game.load.spritesheet` where you must specify the width and height of the frames, and optionally the quantity, i.e.:

```
game.load.spritesheet('uniqueKey', 'sexygirl.png', 37, 45, 18);
```

To load a **texture atlas** you can use `game.load.atlas`. You'll find plenty of examples of this in the Phaser Examples repo.

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 will jump to `frame 0` of the sprite sheet, but you can change it with `sprite.frame` to any other valid number.

If the sprite was using a texture atlas, it's easier to change frame based on the frame name: `sprite.frameName = 'card4'` where the frame name is exactly as specified in the **texture atlas JSON file** (open it up and look at it to see!).

---

<sup>a</sup>[http://labs.phaser.io/edit.html?src=src/loader sprite%20sheet/load%20sprite%20sheet.js](http://labs.phaser.io/edit.html?src=src/loader	sprite%20sheet/load%20sprite%20sheet.js)

---

<sup>7</sup><https://github.com/MakingBrowserGames/Universal-LPC-spritesheet>

## Deeper Dive: Using Base64 Images

Another consideration is using base64 images in Phaser. Many image formats can be converted into “**base64**”<sup>8</sup>. If you’re unfamiliar with what “base64” is or why it exists ***take a look here***<sup>9</sup> and ***here***.<sup>10</sup>

### How to use Base64 as an image

---

```

1   function create () {
2       this.textures.once('addtexture', function () {
3           this.add.image(400, 300, 'brain');
4       }, this);
5       this.textures.addBase64('brain', imageData);
6   }

```

---

See the entire Base64 example at ***labs.phaser.io***<sup>11</sup>.

## 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***.<sup>12</sup>

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.

---

<sup>8</sup><https://www.base64decode.org/>

<sup>9</sup><https://en.wikipedia.org/wiki/Base64>

<sup>10</sup><https://tools.ietf.org/html/rfc4648>

<sup>11</sup><http://labs.phaser.io/edit.html?src=src/textures/texture%20from%20base64.js>

<sup>12</sup><http://makingbrowsergames.com/p3gp-book/index12.html>

**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
    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.Movement-(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™<sup>13</sup>** Watch the developer's console.

## 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:

---

<sup>13</sup>[makingbrowsergames.com/p3gp-book/\\_p3demos-arrav15/](http://makingbrowsergames.com/p3gp-book/_p3demos-arrav15/)

```
// Line 69 - NEW Input Manager v3.16+
// 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)

---

```

1   Game.step();
2     └ MainLoop.step
3       ├ All Active Scenes:
4         ├ Scene.sys.begin (called once per active state)
5           |   └ Iterates Scene.children,
6             |   if child exists, call child.preUpdate
7           ├ While (frameDelta within step range)
8             |   ├ Scene.sys.update
9               |   └ Scene.update
10            ├ Renderer.preRender
11            ├ Scene.sys.render
12              ├ Update Manager Start (Scene.sys.updates)
13              ├ Game.renderer.render (if Scene is visible)
14                |   ├ Renderer set-up (blend mode, clear canvas, etc)
15                |   └ Batch Manager Start
16                  |   └ SceneManager.renderChildren
17                    |   └ Iterates all children, calling child.render on each
18                  └ Update Manager Stop (Scene.sys.updates)
19                    └ 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**”<sup>14</sup> with this number until it “feels” correct

<sup>14</sup><http://dictionary.cambridge.org/us/dictionary/english/tinker>

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

---

```

64 // =====
65 // Example 4.2: Prototyping Movement Properties
66 //   frame refresh and display updates
67 // =====
68 cursors = this.input.keyboard.createCursorKeys();
69 speed = 250;
70 player.setBounce(0.2); // our player will bounce from items
71 player.setCollideWorldBounds(true); // don't go out of the map
72 player.body.velocity.x = 0;
73 player.body.velocity.y = 0;
74 console.log("Movable Black character avatar");
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: [http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson05.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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 player's character inside the visible game stage. You'll discover the **"black" avatar box** cannot penetrate the room's walls — yet all the others can, and **"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 ***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+<sup>a</sup>***

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.

At the moment we have completed 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.

<sup>a</sup><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 key boards layout 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<sup>a</sup>**

**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<sup>b</sup> 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<sup>c</sup>) or Windows 1252 code corresponding to the key. If the key can't be identified, this value is 0.

The value of keypress event is different between browsers. IE and Google Chrome set the `KeyboardEvent.charCodeAt` value<sup>d</sup>. 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`<sup>e</sup>, 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`<sup>f</sup>, 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<sup>g</sup>.

<sup>a</sup><https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent keyCode>

<sup>b</sup>[https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent keyCode#Browser\\_compatibility](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent keyCode#Browser_compatibility)

<sup>c</sup><http://tools.ietf.org/html/20>

<sup>d</sup><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](https://developer.mozilla.org/en-US/docs/Gecko_Keypress_Event)

**Exercise:** More examples on how to “Future Proof” your **game source code in this article.**<sup>15</sup>



**Exercise:** Read more about the differences **between US and UK keyboards here**<sup>16</sup>



**Exercise:** Investigate various **International keyboard layouts used by your gamers.**<sup>17</sup>

## 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.

<sup>15</sup><http://brianchang.info/2016/01/23/how-to-future-proof-your-code.html>

<sup>16</sup>[https://en.wikipedia.org/wiki/British\\_and\\_American\\_keyboards](https://en.wikipedia.org/wiki/British_and_American_keyboards)

<sup>17</sup>[https://en.wikipedia.org/wiki/Keyboard\\_layout](https://en.wikipedia.org/wiki/Keyboard_layout)

"New in 3.16 (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<sup>a</sup>***

<sup>a</sup><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 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 rationale 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:

[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson06.html](http://makingbrowergames.com/p3gp-book/_p3demos/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”**)<sup>18</sup> “... **group together similar objects and control them all as one single unit.** **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:**<sup>a</sup> “... 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.

<sup>a</sup><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

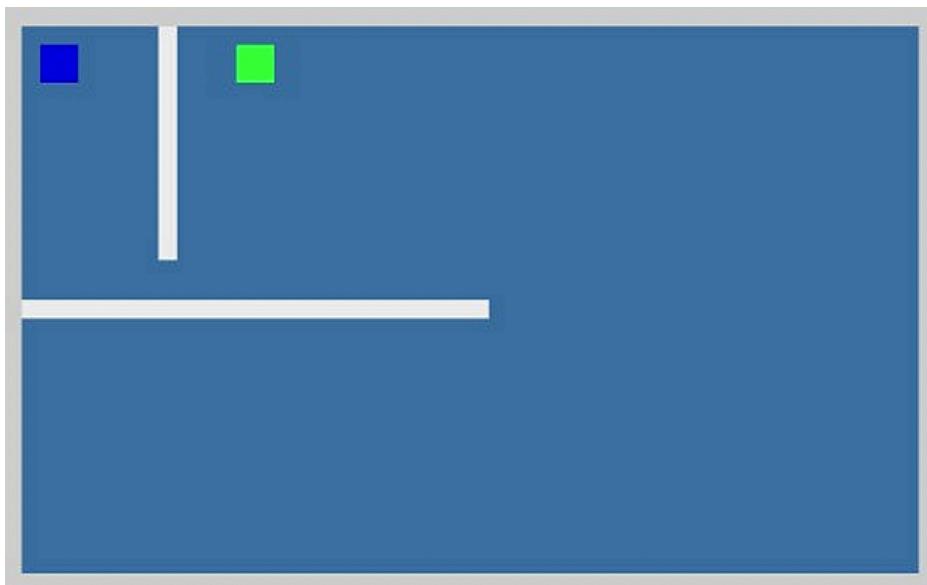
<sup>18</sup><http://phaser.io/tutorials/making-your-first-phaser-3-game/part4>

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 North nor South walls since each of those are 16 pixels wide.

## Interior Decoration



Studio Room

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 `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:

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson07.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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 game play, 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. 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**,<sup>19</sup> there're 5 sprites each of which have 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 using `Shapes` as "hit areas". Firstly, more than one Game Object can share the same "shape hit areas". **This example**<sup>20</sup> demonstrates 400 sprites aligned within a grid, however each of them all share the same `Rectangle` shape hit area. Each sprite does not create their own unique `rectangle` hit area. This preserves memory usage in your game — in other words, the less unique objects created, the better!

---

<sup>19</sup><http://labs.phaser.io/edit.html?src=src/input/mouse/shape%20hit%20tests.js>

<sup>20</sup><http://labs.phaser.io/edit.html?src=src/input/mouse/mass%20sprite%20test.js>

### Sample: One Shared Hit Area

---

```

1   function create () {
2       // Create a little 32x32 texture to use to show where the mouse is
3       var graphics = this.make.graphics(
4           { x: 0, y: 0, add: false,
5               fillStyle: { color: 0xff00ff, alpha: 1 }
6           });
7
8       graphics.fillRect(0, 0, 32, 32);
9       graphics.generateTexture('block', 32, 32);
10      var highlighted = this.add.image(16, 16, 'block');
11      var hitArea = new Phaser.Geom.Rectangle(0, 0, 32, 32);
12      var hitAreaCallback = Phaser.Geom.Rectangle.Contains;
13
14      // Create 400 sprites aligned in a grid
15      group = this.make.group({
16          classType: Phaser.GameObjects.Image,
17          key: 'bobs',
18          frame: Phaser.Utils.Array.NumberArray(0, 399),
19          randomFrame: true,
20          hitArea: hitArea,
21          hitAreaCallback: hitAreaCallback,
22          gridAlign: {
23              width: 25,
24              height: 25,
25              cellWidth: 32,
26              cellHeight: 32
27          }
28      });
29      this.input.on('gameobjectover', function (pointer, gameObject) {
30          highlighted.setPosition(gameObject.x, gameObject.y);
31      });
32  }

```

---

## Doors, Knobs and Buttons

There are two ways to create doorways: 1) Doors are immovable “static” objects, when collided with, the avatar moves 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()
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').setInteractive();
149     this.doorN.setFrame(1);
150     this.doorN.setOrigin(0.5,0);
151     this.doorN.setScale(0.7,0.7);
152     this.doorN.name = "north";
153
154     this.doorN.on('pointerover', function (pointer){
155         console.info(this.doorN.name + " over. ");
156         this._toolTip.setText(GAMEAPP._toolTip);
157         this.doorN.setFrame(2);
158     }, this);
159     this.doorN.on('pointerout', function (pointer){
160         console.info(this.doorN.name + " out. ");
161         this._toolTip.setText("");
162         this.doorN.setFrame(1);
163     }, this);
164     this.doorN.on('pointerdown', function (pointer){
165         console.info(this.doorN.name + " clicked down. ");
166         this.doorN.setFrame(0);
167     }, this);
168     this.doorN.on('pointerup', function (pointer){
169         console.log(this.doorN.name + " click released.");
170         changeRooms(this);

```

---

```

171         console.log("north door entered");
172         this.doorN.setFrame(1);
173     }, this);
174 }
175 */
176 // check for collision
177 this.physics.add.collider(player, doorN, changeRooms, null, this);
178
179 console.log("Northern Door created.");
180 // Example 4.6: ends
181 }
182 /**
183
184 function changeRooms(){
185     console.log("Leaving Room via Northern Door.");
186     // change scene to new room
187     // Refer to Part IV - Project Walk-through - Ruins of Able-Wyvern™
188 };

```

---



**Note:** Refer to this resource file:  
[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson08.html](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/lesson08.html)

The special “Button sprite” does not exist in Phaser III as it does in Phaser v2.x.x; we must take a sprite, image or graphic and simply append the `.setInteractive()`. 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??!!** — the gamer could simply “click” on the door, as any regular button, and enter the next room as a “scene” change.

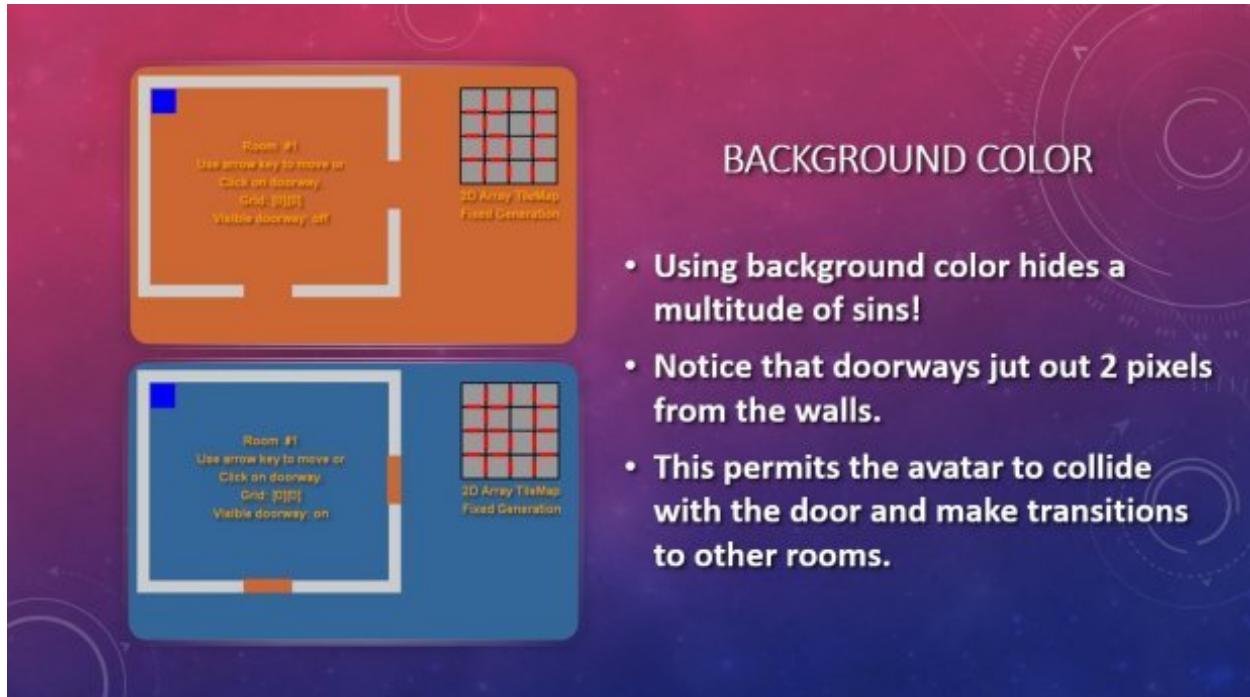
- We should assign a “name” for the door for further processing such as animations.
- We’ll eventually assign a spriteSheet frame to display; but for now we’ll just rectangles.
- We assign physics to stop the avatar on collision and make the door immovable.
- Finally, we assign “the clickable” (when 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 textures, frames or activated sound effects. ***This also reminds me of 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** — when the Pointer moves over the Button game object. This is also commonly known as ‘hover’. Mobile devices use the “down state” below.
- **pointerout** — when the Pointer that was previously over the Button “moves out” away from it. Mobile devices use the “up state” below.
- **pointerdown** — when the Pointer is pressed down while over the Button game object or “touched” on a touch-enabled device or clicked with the mouse.
- **pointerup** — when the Pointer that was pressed down on the Button is released again.



**Note:** I highly recommend **this button plugin**<sup>21</sup> from “rexRainBow” or **William Clarkson’s style for buttons**.<sup>22</sup>



Sample from Part III: 2D Array and Door Placement

<sup>21</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/button/>

<sup>22</sup><https://phasergames.com/how-to-make-buttons-in-phaser-3/>



**Exercise:** Try the live v2.x.x demonstration [here<sup>23</sup>](#) or v3.16+ demonstration [here<sup>24</sup>](#)

### 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
            Rooms2D.CrntRoom = Rooms2D.LastRoom - 4;
            Rooms2D.CrntRoomY -= 1;
            //Leave via North; enter new room from South-side
            Rooms2D.pPosX = config.width / 3;
            Rooms2D.pPosY = 320;
            break;

        case "East":
            Rooms2D.CrntRoom += 1;
            Rooms2D.CrntRoomX += 1;
            //Leave via East; Enters new room from the west-side
            Rooms2D.pPosX = 50;
            Rooms2D.pPosY = config.height / 2;
            break;

        case "South":
```

---

<sup>23</sup><http://makingbrowergames.com/book/ch6/index.html>

<sup>24</sup>[http://makingbrowergames.com/p3gp-book/\\_p3-2DRooms/](http://makingbrowergames.com/p3gp-book/_p3-2DRooms/)

```

Rooms2D.CrntRoom += 4; // or GRID_ROWS or MT.length
Rooms2D.CrntRoomY += 1;
//Leave via South; enter new room from North-side
Rooms2D.pPosX = config.width / 3;
Rooms2D.pPosY = 50;
break;

case "West":
    Rooms2D.CrntRoom -= 1;
    Rooms2D.CrntRoomX -= 1;
    //Leave via West; enters new room from 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);

false;
};

```

---

**Try the live v2.x.x demonstration here<sup>25</sup> or v3.16+ demonstration here<sup>26</sup>**

Download the Phaser III source code from:

[https://makingbrowergames.com/p3gp-book/p3\\_2DRoomsDemo.pdf](https://makingbrowergames.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".<sup>27</sup> 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

<sup>25</sup><http://makingbrowergames.com/book/ch6/index.html>

<sup>26</sup>[http://makingbrowergames.com/p3gp-book/\\_p3-2DRooms/](http://makingbrowergames.com/p3gp-book/_p3-2DRooms/)

<sup>27</sup><https://en.wikipedia.org/wiki/Azimuth>

code, and then perform some “magic special effects” — similar to slide transition in a business meeting power point 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 game at a rapid pace, is learning to write simple, modularized code. Here's some guidelines for doing so; they are:

- ***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 descriptive method, function and variable names so if someone else were to read your source code, they would easily be able to tell what what your intent is.
- ***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 let's you call it from wherever it is needed. Having 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***<sup>28</sup> into a separate functions or file modules for easy reuse.



**Exercise:** Read “**Clean Code**”<sup>29</sup> by Robert Martin converted in 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 abuses it.” Now, let me show you a “ninja trick” on buttons.***

Obviously, a game phase will have numerous buttons in the heads up display (HUD) and menus. It only becomes natural to make a “**Button Class**” ***cookie-cutter*** and “**stamp**” ***out***<sup>30</sup> out all our pretty buttons for our user’s interface(s). Well, you could

---

<sup>28</sup>[https://en.wikipedia.org/wiki/Abstraction\\_principle\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Abstraction_principle_(computer_programming))

<sup>29</sup><https://github.com/ryanmcdermott/clean-code-javascript>

<sup>30</sup><https://medium.com/javascript-scene/introducing-the-stamp-specification-77f8911c2fee>

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<sup>31</sup> of buttons) their very own Phaser.Scene?!?*** Think of the possibilities . . . ninja!

## Deeper Dive: Button size considerations

**Quote:**<sup>a</sup> Apple’s ***iPhone Human Interface Guidelines***<sup>b</sup> 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***,<sup>c</sup> 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<sup>d</sup> 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.

<sup>a</sup><https://www.smashingmagazine.com/2012/02/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/>

<sup>b</sup><https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/>

<sup>c</sup>[https://en.wikipedia.org/wiki/Fitts%27s\\_law](https://en.wikipedia.org/wiki/Fitts%27s_law)

<sup>d</sup>[http://makingbrowergames.com/p3gp-book/\\_Touch\\_key\\_design\\_for\\_target\\_selection\\_on\\_a\\_mobile\\_.pdf](http://makingbrowergames.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

<sup>31</sup>[https://en.wiktionary.org/wiki/chock\\_full](https://en.wiktionary.org/wiki/chock_full)

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**<sup>32</sup> 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:**<sup>33</sup> For iOS, you’ll activate the minimize app gesture as soon as you use the 6th finger — and there’s nothing Phaser can do to stop it.

## 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:

---

<sup>32</sup><https://madmimi.com/p/6f870d>

<sup>33</sup><http://labs.phaser.io/index.html?dir=input/multitouch/&q=>

```
//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, player's characters only have one life — so to illustrate this upcoming 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 must change when final artwork is
146     // 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: 0x00ffff,
184         // maxEntries: 16,
185         // set false if amount of dynamic bodies > 5000
186         // useTree: true
187     }
188 }
189 // ...

```

---



**Note:** Refer to this resource file:  
[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson09.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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<sup>34</sup>**

So, we have thingies checking when the player is in contact with the monster. When the player moves into the monster and whether the player is “over lapping” the monster. Which is better? Do we need both? Well, if you’ll remember, we wanted to include 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 (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 the 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();
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:

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson10.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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

<sup>34</sup><https://codepen.io/samme/pen/yqjoym?editors=0010>

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***<sup>35</sup>, our avatar died; the game is over. I'm starting to ***“tear-up”***<sup>36</sup> .... 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

<sup>35</sup><https://forum.wordreference.com/threads/its-curtains-for-you.1509930/>

<sup>36</sup>[http://www.macmillandictionary.com/us/dictionary/american/tear-up\\_2](http://www.macmillandictionary.com/us/dictionary/american/tear-up_2)

“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 };

```

---

In order for Phaser to recognize this new game-transition, we must add it to the list of game scenes in the game’s config. We have a couple of development path options:

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 file for this game phase and tell the index.html to load this external script.

Which is better? You learned that we automated the “rose bouquet” process; anytime an external game phase (i.e., module file) is discovered, it is automatically added to the game.scenes. This is the “D.R.Y.”-est approach and is already in place. The first two options “**mucks around**”<sup>37</sup> with our development regimen. But to prove the point of **adding “cruft”**<sup>38</sup> into our game add the following code and conduct some experiments:

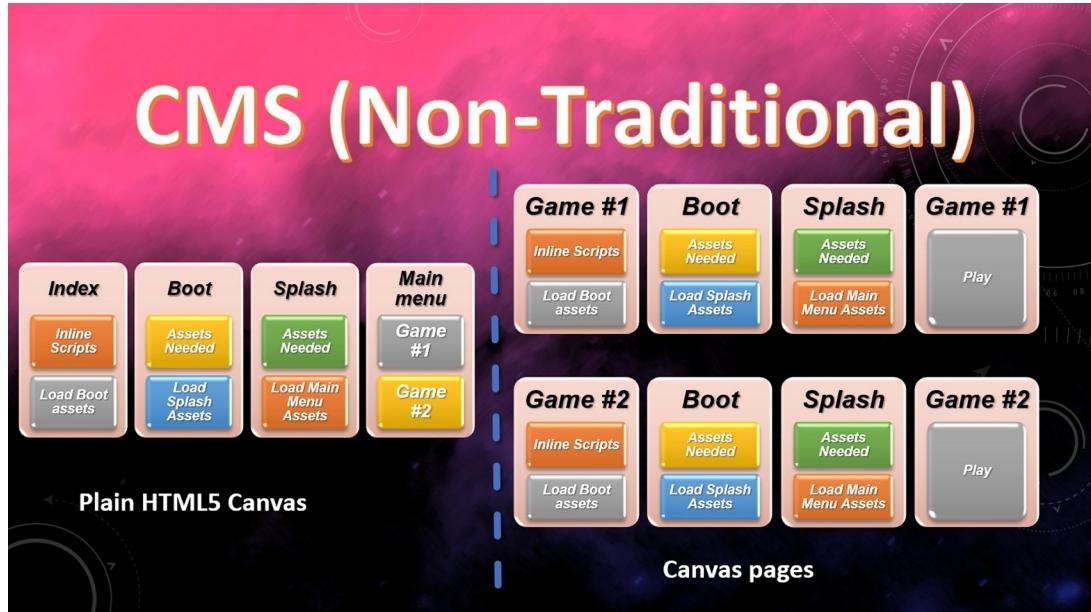
```
// See Line 50
game.scene.add("gameOver", gameOverState); //mucking around!
```

**OR**

<sup>37</sup><https://idioms.thefreedictionary.com/muck+around>

<sup>38</sup><https://en.oxforddictionaries.com/definition/cruft>

```
add gameOver: gameOver into config.scene array. //mucking around!
```



Launching a New Game Phase as a separate HTML5 page.

Inside this `gameOver create() function`,<sup>39</sup> 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:

#### Example 4.10: Elementary HUD Creation

```

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 gameOver = {
28   create: function(){
29     // -----
30     //Example 4.10: Elementary HUD Creation begins
31     // -----
32     var label = this.add.text(
33       config.width/2,           //centering HUD horizontally
34       config.height/2,          //centering HUD vertically
35       "Game Over \n Press the SPACE bar to start again",

```

<sup>39</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson11a.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch4-examples/lesson11a.html)

```

36         {font: "22px Arial", fill: "#FFF", align:"center"}));
37     label.setOrigin(0.5,0.5);
38     //Example 4.10: ends
39     // -----
40     // . . .
41 }, //comma very important here

```

---



**Note:** Refer to this resource file:

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson11.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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 concepts 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 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 automate this hoisting process and have forgotten to pay attention to this as a results.

Similarly to our graphics placements, we are able to 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:  
[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson11.html](http://makingbrowsergames.com/p3gp-book/_p3demos/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 Mechanism Categories

Other Game Mechanisms (i.e., software routines and input controls) fall into 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 Mechanisms (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**,<sup>40</sup> and **sentinel variables**.<sup>41</sup> Yes, by mixing and matching various game mechanics into a game, it changes a game into a new project release.

**Need Proof?** Return to **this file**<sup>42</sup> and change Line 209 `handlePlayerDeath();` to `combatEncounter` and watch how you **have simply entered a new game rule to modify game play!** Yes, these rules could be in external JavaScript Modules **that are loading dynamically on demand during play. More about that process in the coming chapters in Part II.**

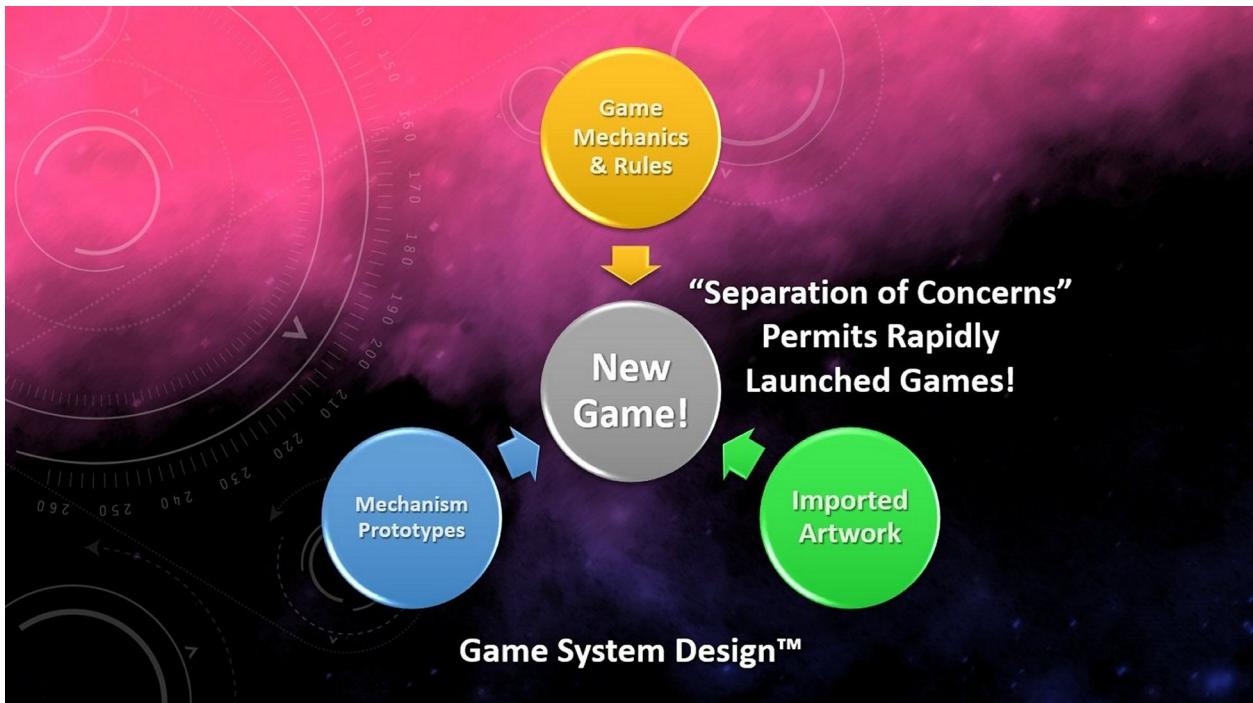


**Hint:** You must use my file (not yours!) because you don't have the function `combatEncounter` nor the new game phase `combat` created yet.

<sup>40</sup>[https://computersciencewiki.org/index.php/Abstract\\_data\\_structures](https://computersciencewiki.org/index.php/Abstract_data_structures)

<sup>41</sup>[https://en.wikipedia.org/wiki/Sentinel\\_value](https://en.wikipedia.org/wiki/Sentinel_value)

<sup>42</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson11.html](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/lesson11.html)



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

Here are a few suggestions from Chapter 5 **Game Mechanics & Rules** (aka: **Heuristic**<sup>43</sup>). 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 from the following mechanisms. **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**<sup>44</sup>) Use

<sup>43</sup><https://en.wikipedia.org/wiki/Heuristic>

<sup>44</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/Agent.html#/apple\\_ref/doc/uid/TP40015172-CH8-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Agent.html#/apple_ref/doc/uid/TP40015172-CH8-SW1)

this simulation to let game characters move themselves based on high-level goals and react to their surroundings.

- **Artificial Intelligence Strategist:** (See AI chapter and *Apple's Game-Play Kit*<sup>45</sup>) 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 rights 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 mechanism using movement into the same area (immediate elimination or deterministic combat), jumping across an opponent as in checkers, producing a “checkmate” event from which the opponent has no movement options. Many online games define the capture mechanism as a “kill count” that reflects the sum of opponent tokens eliminated during the game.
- **Catch-up:** This mechanism 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*<sup>46</sup>) The most common use is to randomly determine an outcome of a game interaction. This is deeper problem than most folks will admit. You are often looking for a specific statistical distribution (**NOTE:** if you haven't played **AD&D**<sup>47</sup>, then think about the bell curve derived from rolling two six-sided dice, the most frequently appearing results (**68% of the time**)<sup>48</sup> 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;
```

<sup>45</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/Minmax.html#/apple\\_ref/doc/uid/TP40015172-CH2-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Minmax.html#/apple_ref/doc/uid/TP40015172-CH2-SW1)

<sup>46</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/RandomSources.html#/apple\\_ref/doc/uid/TP40015172-CH9-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RandomSources.html#/apple_ref/doc/uid/TP40015172-CH9-SW1)

<sup>47</sup>[https://en.wikipedia.org/wiki/Editions\\_of\\_Dungeons\\_%26\\_Dragons](https://en.wikipedia.org/wiki/Editions_of_Dungeons_%26_Dragons)

<sup>48</sup><http://www.compensationcafe.com/2014/04/ding-dong-the-wicked-bell-curve-is-dead.html>

```
if (results <= to CharacterSkill){ ... Do something ... }
```

- **Game modes:** One of the most common examples of 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*<sup>49</sup>) Separating game mechanics from display code will optimize your gameplay rendering cycle. Implementing fuzzy logic reasoning 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 mechanisms. The current game area may effect movement (e.g., forest area 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*<sup>50</sup>) — 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 playing surface is displayed. The grid defines the movement rules; how the grid is drawn is the mechanism.

<sup>49</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/RuleSystems.html#/apple\\_ref/doc/uid/TP40015172-CH10-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RuleSystems.html#/apple_ref/doc/uid/TP40015172-CH10-SW1)

<sup>50</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/StateMachine.html#/apple\\_ref/doc/uid/TP40015172-CH7-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/StateMachine.html#/apple_ref/doc/uid/TP40015172-CH7-SW1)

- **Turns:** A game turn is an important fundamental concept; it could be an abstract representation to regulate gameplay or denote a passage of time in a game set aside for certain player actions to happen before moving on to another turn. In simulation games, turns represent time in a more concrete 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 which could be considered the surviving trace of the turn concept.

```
gameTurn += 1;
```

- **Worker Deployment:** is a game mechanism where players allocate a limited number of tokens ("workers") to multiple stations that provide various defined actions. This is commonly used in Tower Defense games.



**Note:** *Apple's Game-Play Kit*<sup>51</sup> 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*,<sup>52</sup> *SceneKit for 3D games*,<sup>53</sup> or a custom or third-party game engine using *Metal*<sup>54</sup> or *OpenGL ES*.<sup>55</sup> For games with less demanding graphics needs, you can even use **Apple's Game-Play Kit**<sup>56</sup> 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"**<sup>57</sup>, but remember to be humble and kind!<sup>58</sup> Celebrate! You have a fully function Phaser v3.x.x game prototype that:

<sup>51</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/index.html#/apple\\_ref/doc/uid/TP40015172-CH1-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/index.html#/apple_ref/doc/uid/TP40015172-CH1-SW1)

<sup>52</sup><https://developer.apple.com/documentation/spritekit>

<sup>53</sup><https://developer.apple.com/documentation/scenekit>

<sup>54</sup><https://developer.apple.com/documentation/metalkit>

<sup>55</sup><https://developer.apple.com/documentation/opengles>

<sup>56</sup><https://developer.apple.com/documentation/gameplaykit>

<sup>57</sup><http://www.urbandictionary.com/define.php?term=miller%20time>

<sup>58</sup><https://www.youtube.com/watch?v=awzNHuGqoMc&list=PLuvCpe8H09C8ViEKyO4Qzo25JUaziEpK>

- 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**"<sup>59</sup> 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: <http://makingbrowergames.com/p3gp-book/>

Complete Chapter ***Source Code in the online appendix.***<sup>60</sup>

***Play III Game Prototype Demo thus far***<sup>61</sup>

- ***Example 2.4 Bare-bones Index Page - Traditional Method***<sup>62</sup>
- ***Example 2.5: Starting the Game.js***<sup>63</sup>
- ***Example 3.1a: Creating State Objects in Game.js - traditional method***<sup>64</sup>
- ***Example 4.1: Prototyping a Visual Avatars***<sup>65</sup>
- ***Example 4.2: Prototyping Movement Properties in v3***<sup>66</sup>
- ***Example 4.3: Movement Arrows v3 Integration***<sup>67</sup>
- ***Example 4.4: World Boundaries Grouping***<sup>68</sup>
- ***Example 4.5: World Boundaries Integration***<sup>69</sup>
- ***Example 4.6: Interior Boundaries Integration***<sup>70</sup>
- ***Example 4.7: Collision Detection Integration***<sup>71</sup>
- ***Example 4.8: Collision Results Determination***<sup>72</sup>
- ***Example 4.9: New Game Over State***<sup>73</sup>

<sup>59</sup><http://www.urbandictionary.com/define.php?term=Awesomesauce>

<sup>60</sup><http://makingbrowergames.com/p3gp-book/tools.html>

<sup>61</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch4-examples/](http://makingbrowergames.com/p3gp-book/_p3demos/ch4-examples/)

<sup>62</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowergames.com/p3gp-book/_p3demos/bareBonesIndex.html)

<sup>63</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson02.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson02.html)

<sup>64</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson03.html)

<sup>65</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson04.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson04.html)

<sup>66</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson04.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson04.html)

<sup>67</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson05.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson05.html)

<sup>68</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson06.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson06.html)

<sup>69</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson07.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson07.html)

<sup>70</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson08.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson08.html)

<sup>71</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson09.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson09.html)

<sup>72</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson10.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson10.html)

<sup>73</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson11.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson11.html)

- **Example 4.10: Elementary HUD Creation**<sup>74</sup>
- **Example 4.11: Collecting User Input**<sup>75</sup>
- **Example 4.12: Responding to User Input**<sup>76</sup>

## 4.9 Summary

Examples:

- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowsergames.com/p3gp-book/_p3demos/bareBonesIndex.html)
- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/index.html](http://makingbrowsergames.com/p3gp-book/_p3demos/index.html)
- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/index-OLOO.html](http://makingbrowsergames.com/p3gp-book/_p3demos/index-OLOO.html)

Here's an inventory of what we've learned thus far.

- Game Prototyping uses simple graphics and focuses on game mechanics.<sup>77</sup>
- 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<sup>78</sup> our game.
- Learned each Phaser game state has separate functions of which the 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 physics 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.

<sup>74</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/lesson12.html](http://makingbrowsergames.com/p3gp-book/_p3demos/lesson12.html)

<sup>75</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/lesson11a.html](http://makingbrowsergames.com/p3gp-book/_p3demos/lesson11a.html)

<sup>76</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/lesson11a.html](http://makingbrowsergames.com/p3gp-book/_p3demos/lesson11a.html)

<sup>77</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

<sup>78</sup><http://www.dictionary.com/browse/modularize>

## 4.10 Chapter References

(See more references)

- ***How to Prototype a Game in Under 7 Days***<sup>79</sup>
- ***MDN Game development***<sup>80</sup>
- ***Game Design Concepts 5.1: Prototyping***<sup>81</sup>
- ***Plain English Guide to JavaScript Prototypes***<sup>82</sup>
- ***JavaScript Classes***<sup>83</sup>
- ***<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://en.wikipedia.org/wiki/Sui_generis_database_right)***
- ***<https://www.michalsons.com/blog/the-rights-to-a-database/2937>***

---

<sup>79</sup>[http://www.gamasutra.com/view/feature/130848/how\\_to\\_prototype\\_a\\_game\\_in\\_under\\_7\\_.php?print=1](http://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php?print=1)

<sup>80</sup><https://developer.mozilla.org/en-US/docs/Games>

<sup>81</sup><https://learn.canvas.net/courses/3/pages/level-5-dot-1-prototyping>

<sup>82</sup><http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/>

<sup>83</sup><https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>



## 5. Game Recipe™ Automation Tool

We've generated a bunch of **code snippets**<sup>1</sup> so far. This **codebase**<sup>2</sup> 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**<sup>3</sup> based upon your clients' requests. This process is a gentle **introduction into "cloud-based"**<sup>4</sup> services and eventual **MMoG development.**<sup>5</sup>



**Exercise:** Research applications that save "snippets". Here's a good starting point:

- *JetBrains WebStorm*<sup>6</sup> or,
- *Snippet Designer for Visual Studio*<sup>7</sup> or,
- *Snip2Code*<sup>8</sup> 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**.<sup>9</sup> AND, the newest **Phaz3r "documentation"** ... well, enough ranting for one sentence ;)

<sup>1</sup>[https://en.wikipedia.org/wiki/Snippet\\_\(programming\)](https://en.wikipedia.org/wiki/Snippet_(programming))

<sup>2</sup><https://en.wikipedia.org/wiki/Codebase>

<sup>3</sup><http://makingbrowergames.com/gameDesigner/index-randommechanic.html>

<sup>4</sup>[https://www.w3schools.com/appml/appml\\_architecture.asp](https://www.w3schools.com/appml/appml_architecture.asp)

<sup>5</sup><https://www.ics.uci.edu/~wscacchi/GameIndustry/Lecture06-MMORPG-Planning.html>

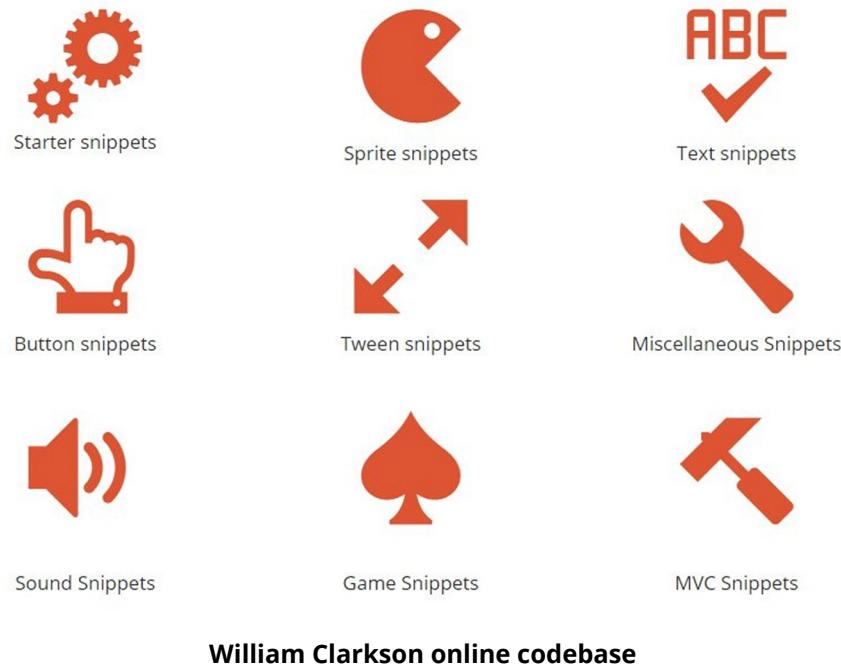
<sup>6</sup><https://blog.jetbrains.com/webstorm/2018/01/using-and-creating-code-snippets/>

<sup>7</sup><https://marketplace.visualstudio.com/items?itemName=vs-publisher-2795.SnippetDesigner>

<sup>8</sup><https://www.snip2code.com/>

<sup>9</sup><https://en.wikipedia.org/wiki/Taxonomy>

Let's learn something from one of my colleagues and ***how he organizes his code.***<sup>10</sup> Organization and correct labeling (in your own mind) is the foundation. ***It will greatly help when we begin to create our "Entity Relationship Diagrams" for our codebase.***



This is an excellent start into ***category classification;***<sup>11</sup> — software professional cannot hide their mastery in “**taxis**” (Greek for “arrangement”) and “**nomos**” (Greek for “law”) = **Taxonomy**.<sup>12</sup> 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**<sup>13</sup> and books. His focus is 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.***

## 5.1 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

<sup>10</sup><https://williamclarkson.net/code/phaser-snippets/>

<sup>11</sup><https://www.britannica.com/science/taxonomy>

<sup>12</sup><https://www.britannica.com/science/taxonomy>

<sup>13</sup><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>

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 (publicly or privately) so that its 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**<sup>14</sup> 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 provide protection for software used to create the database or for material contained in the database**. It is the scheme of the database that is protected."
- **Database Legal Protection**<sup>15</sup> 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)"

## 5.2 Database Schema Construction (Copyright-able!!)

Let's catalog our various game prototypes we've created thus far. You'll find in Chapter 1 we've begun<sup>16</sup> this process with "Chart 2" already. Here's other categories to include:

- **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.**"<sup>17</sup>
- **Objects** — Ch.1 "**buttons", "text" and "game object**" — a generic building block containing **metadata**.<sup>18</sup> Everything in JavaScript is an object, but we'll use this

<sup>14</sup>[https://www.esa.int/About\\_Us/Law\\_at\\_ESA/Intellectual\\_Property\\_Rights/Copyright\\_and\\_databases](https://www.esa.int/About_Us/Law_at_ESA/Intellectual_Property_Rights/Copyright_and_databases)

<sup>15</sup><https://www.bitlaw.com/copyright/database.html>

<sup>16</sup><https://brians.wsu.edu/2016/05/19/began-begun/>

<sup>17</sup><https://softwareengineering.stackexchange.com/questions/229479/how-did-separation-of-code-and-data-become-a-practice>

<sup>18</sup><https://cuahsi.zendesk.com/hc/en-us/articles/208639877-What-is-Metadata->

category to mean **non-visual items and source code**.

- **Events — Ch.1 “coded”** — will be our “messengers” that look and listen for some set of conditions and respond with an action notice.
- **Behaviors — Ch.1 “animation”** — we’ll place source code in this category to modify any objects actions. For example `physics` behavior makes objects react to events.
- **Scenes — Ch.1 “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.

## Database Record Construction

Suggested snippet record fields:

- record identification number (primary key as “`sID`”)
- category grouping (foreign key into category table as “`catID`”)
- “`framework`” ID grouping (foreign key into framework table; 1:1; )
- “`deployed`” on either “client”, “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**.<sup>19</sup> when supplementing this suggest record format.

### SQL sample table creation

---

```

1 //created with sqlite browser; visit http://sqlitetutorial.net/
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.

<sup>19</sup><https://www.guru99.com/database-normalization.html>

## Database structure

There are several technologies we might consider. For XML, I recommend Microsoft XML Notepad. It is open source and easy to use. XML Notepad 2007 is the latest release. Another option is the XAML tool integrated into Visual Studio. You can find it by searching for “xmlnotepad” or visit <https://www.microsoft.com/en-us/download/details.aspx?id=7973>



**Exercise:** Download the following XML or JSON databases for our codebase:



**Hint:** Once you have a spiffy XML database file, you might like to convert it into JSON. I use [this online tool<sup>21</sup>](#) or you might like to pick your own.

## 5.3 Remote Codebase Using AppML

A simple local snippet pool would have been sufficient for our games and similar to what we did in [\*\*Game Designer\*\*<sup>22</sup>](#) website. [\*\*Google Actions\*\*<sup>23</sup>](#) uses an “Excel 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**<sup>24</sup>. Also, I have decided to use the [\*\*W3School's version of AppML\*\*<sup>25</sup>](#) 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

---

<sup>20</sup><http://taffydb.com/>

<sup>21</sup><http://convertjson.com/xml-to-json.htm>

<sup>22</sup><http://makingbrowergames.com/gameDesigner/>

<sup>23</sup><https://developers.google.com/actions/>

<sup>24</sup><http://makingbrowergames.com/gameDesigner/index-randommechanic.html>

<sup>25</sup><https://www.w3schools.com/appml/default.asp>

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 **Game Designer**,<sup>26</sup> 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](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 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](https://www.w3schools.com/js/js_json_intro.asp)

**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.

---

<sup>26</sup><http://makingbrowsergames.com/gameDesigner/>

- 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.

## 5.4 Building an AppML application

AppML applications are simple to build. The AppML tutorial<sup>27</sup> could be summarized into this 4-step process.

1. Describe the elements of the snippets application with AppML. (style / XML conformance) Review 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*)<sup>28</sup> or *this page*.<sup>29</sup> or consider putting your snippets *in the “cloud”*<sup>30</sup> 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.

## 5.5 Sample AppML codebase (Public Access)

- <http://makingbrowergames.com/gameDesigner/>
- <http://makingbrowergames.com/gameDesigner/index-genre.html>

## 5.6 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>

---

<sup>27</sup><https://www.w3schools.com/appml/default.asp>

<sup>28</sup>[https://www.w3schools.com/appml/appml\\_data.asp](https://www.w3schools.com/appml/appml_data.asp)

<sup>29</sup>[https://www.w3schools.com/appml/appml\\_php.asp](https://www.w3schools.com/appml/appml_php.asp)

<sup>30</sup>[https://www.w3schools.com/appml/appml\\_google\\_cloud\\_sql.asp](https://www.w3schools.com/appml/appml_google_cloud_sql.asp)



**Exercise:** Review the following JSON databases for **Game Designer**<sup>31</sup>:

---

<sup>31</sup><http://makingbrowsergames.com/gameDesigner/>

## 5.7 Chapter Source Code & Demo

book website: <http://makingbrowergames.com/p3gp-book/>

Complete Chapter ***Source Code in the online appendix.***<sup>32</sup>

***Play III Game Prototype Demo thus far***<sup>33</sup>

- ***Example 2.4 Bare-bones Index Page - Traditional Method***<sup>34</sup>
- ***Example 2.5: Starting the Game.js***<sup>35</sup>
- ***Example 3.1a: Creating State Objects in Game.js - traditional method***<sup>36</sup>
- ***Example 4.1: Prototyping a Visual Avatars***<sup>37</sup>
- ***Example 4.2: Prototyping Movement Properties in v3***<sup>38</sup>
- ***Example 4.3: Movement Arrows v3 Integration***<sup>39</sup>
- ***Example 4.4: World Boundaries Grouping***<sup>40</sup>
- ***Example 4.5: World Boundaries Integration***<sup>41</sup>
- ***Example 4.6: Interior Boundaries Integration***<sup>42</sup>
- ***Example 4.7: Collision Detection Integration***<sup>43</sup>
- ***Example 4.8: Collision Results Determination***<sup>44</sup>
- ***Example 4.9: New Game Over State***<sup>45</sup>
- ***Example 4.10: Elementary HUD Creation***<sup>46</sup>
- ***Example 4.11: Collecting User Input***<sup>47</sup>
- ***Example 4.12: Responding to User Input***<sup>48</sup>

---

<sup>32</sup><http://makingbrowergames.com/p3gp-book/tools.html>

<sup>33</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch4-examples/](http://makingbrowergames.com/p3gp-book/_p3demos/ch4-examples/)

<sup>34</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowergames.com/p3gp-book/_p3demos/bareBonesIndex.html)

<sup>35</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson02.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson02.html)

<sup>36</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson03.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson03.html)

<sup>37</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson04.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson04.html)

<sup>38</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson04.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson04.html)

<sup>39</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson05.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson05.html)

<sup>40</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson06.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson06.html)

<sup>41</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson07.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson07.html)

<sup>42</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson08.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson08.html)

<sup>43</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson09.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson09.html)

<sup>44</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson10.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson10.html)

<sup>45</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson11.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson11.html)

<sup>46</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson12.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson12.html)

<sup>47</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson11a.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson11a.html)

<sup>48</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/lesson11a.html](http://makingbrowergames.com/p3gp-book/_p3demos/lesson11a.html)

## 5.8 Summary

Examples:

- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/bareBonesIndex.html](http://makingbrowsergames.com/p3gp-book/_p3demos/bareBonesIndex.html)
- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/index.html](http://makingbrowsergames.com/p3gp-book/_p3demos/index.html)
- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/index-OLOO.html](http://makingbrowsergames.com/p3gp-book/_p3demos/index-OLOO.html)

Here's an inventory of what we've learned thus far.

- Game Prototyping uses simple graphics and focuses on game mechanics.<sup>49</sup>
- 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<sup>50</sup> our game.
- Learned each Phaser game state has separate functions of which the 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.

---

<sup>49</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

<sup>50</sup><http://www.dictionary.com/browse/modularize>

## 5.9 Chapter References

(See more references)

- *How to Prototype a Game in Under 7 Days*<sup>51</sup>
- *MDN Game development*<sup>52</sup>
- *Game Design Concepts 5.1: Prototyping*<sup>53</sup>
- *Plain English Guide to JavaScript Prototypes*<sup>54</sup>
- *JavaScript Classes*<sup>55</sup>
- <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://en.wikipedia.org/wiki/Sui_generis_database_right)
- <https://www.michalsons.com/blog/the-rights-to-a-database/2937>

---

<sup>51</sup>[http://www.gamasutra.com/view/feature/130848/how\\_to\\_prototype\\_a\\_game\\_in\\_under\\_7\\_.php?print=1](http://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php?print=1)

<sup>52</sup><https://developer.mozilla.org/en-US/docs/Games>

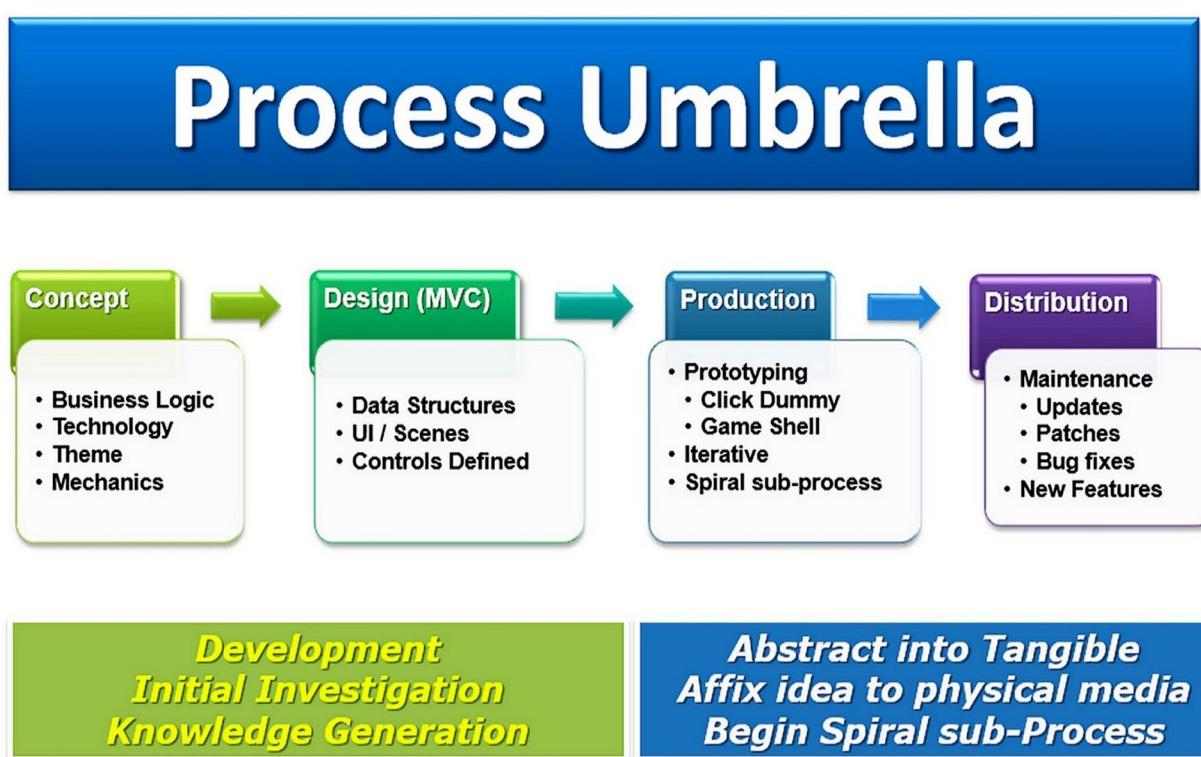
<sup>53</sup><https://learn.canvas.net/courses/3/pages/level-5-dot-1-prototyping>

<sup>54</sup><http://sporto.github.io/blog/2013/02/22/a-plain-english-guide-to-javascript-prototypes/>

<sup>55</sup><https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>



# Part II - Mechanisms into Mechanics



Part II Chapters 6 to 10 covers connecting **Game Recipes™** into the **Game Design System™**. It is the construction stage of my project management process. We created various common game components in Phaser v3.16.x and prepared our Part I game prototypes for Part II's game mechanics, rules and logic. In Part III, we'll take those various items we created from Parts I & II and learn to combine them in Part III Tutorial Walk-throughs as:

- RPG maze games,

- code 6 different combat systems,
- heads-up displays (HUD) both internal to and outside of the Phaser canvas,
- apply 6 different artificial intelligence systems,
- build grids and tiled-maps in Phaser v3.16.x rendering,
- create other actions with popular game perspective of 2D, 2.5D and 3D.

You'll discover how to develop multiple levels, isometric scenes using new features available in Phaser v3.16.x. All of these techniques and supporting source code are explained in an easy-to-understand manner for game designers to gain new skills in Phaser v3.16.x.

# 6. Game Mechanics & Systems

I once heard a game described as "... a series of interesting decisions." What are those decisions within your game? Whatever those are, they become the fundamental core of your game's mechanics. Remember how we "**deconstructed**" an **action-fighting** game in chapter 1? Any other decision points, such as how many dice to roll, in your game are only relevant within the context of these core decisions. Those identified core mechanics are similar to a building's infrastructure — the framing or **steel girders**<sup>1</sup> — they should carry the "weight" of the game's housing. If your game mechanics aren't captivating or **titlizing**<sup>2</sup>, it really doesn't matter what else you have going on during game-play; the game **will stink**<sup>3</sup>. Focus on the core decision points of your game mechanics before choosing a color to paint the walls.

## 6.1 Game-Play vs Game Mechanics vs Game Mechanism

Let's study **game-play and its connection to "fun"**,<sup>4</sup> "Human-Computer Interaction" (HCI), and understanding why humans have the desire to play in **Phaser III Game Design Workbook**<sup>5</sup>. In this section, we will define game mechanics and mechanism.



**Exercise:** Review **16 Human Motivations to the "FUN" factors**<sup>6</sup> from the Bonus Content.

Game mechanics are the rules, regulations, and methods that are designed to constrain a player's interactions using input and control mechanisms — which we created in Part I. The player's input, via these mechanisms, modifies the game's environment. All games use rules; all games have input control mechanisms for the player to mold game's environment toward their advantage. In general, ludology (aka "the study of game design") provides a way to create game rules (aka mechanics) that allow players an engaging and fun(?) experience. The interactions and limitations, imposed by these governing rules, create the "complexity" of gaming strategies. Some

<sup>1</sup><http://northern-weldarc.com/steel-girders-different-steel-beams/>

<sup>2</sup><https://www.urbandictionary.com/define.php?term=titilizing>

<sup>3</sup><https://www.dictionary.com/browse/stink>

<sup>4</sup><http://makingbrownergames.com/book/16HumanMotivations.pdf>

<sup>5</sup><https://leanpub.com/phaser3gamedesignworkbook>

<sup>6</sup><http://makingbrownergames.com/book/16HumanMotivations.pdf>

game mechanics have existed for centuries, as in the example of the chess game, while other mechanics are relatively new and prompted by the dawn of personal computing devices.

Quote from ***Phaser III Game Design Workbook***<sup>a</sup> "Let's define what game-play is. "It is the tactical aspects of a computer game, such as its plot and the way it is played, as distinct from the graphics and sound effects." Game-play is the specific way in which players interact with a game known as "Human-Computer Interface". We could add onto these definitions various modes of game-play such as: **Asymmetric, Cooperative, Emergent, and Nonlinear.**<sup>b</sup> However, this is all on the "science-side" of game design and overlooks the "human-side".

Game-play is all about "Human-Computer Interaction" (HCI) — how the player interacts with a game — and understanding why humans play. Whenever you're thinking about rules, plot, objectives, or challenges, and how a player is meant to interact with these, you're thinking about the gameplay. For example, kittens and puppies play. Their antics are cute and often-times humorous; but, it is their way of training for life and acquiring the necessary survival skills. Humans play for the same final results — survival! But more than survival, humans also seek happiness; no one intentionally practices "pain and suffering", or do they? from ***Phaser III Game Design Workbook pg 51***<sup>c</sup>

<sup>a</sup><https://leanpub.com/phaser3gamedesignworkbook/>

<sup>b</sup>[https://en.wikipedia.org/wiki/Game\\_theory](https://en.wikipedia.org/wiki/Game_theory)

<sup>c</sup><https://leanpub.com/phaser3gamedesignworkbook>

From a programming perspective, basic Game-Play can be deconstructed — revealing tactical components inside game mechanics' rules. For example, a **fps-shooting game** deconstructs into various tactics such as shooting (or reloading, exchanging weapons, and throwing), defensive moves, healing, and dodges. These tactics are assigned to game mechanisms — input keys, mouse clicks, buttons, and mobile screen interactions. These maneuvers could be further enhanced into "sniper headshots" or "strafing fire" from the other various input control combination.

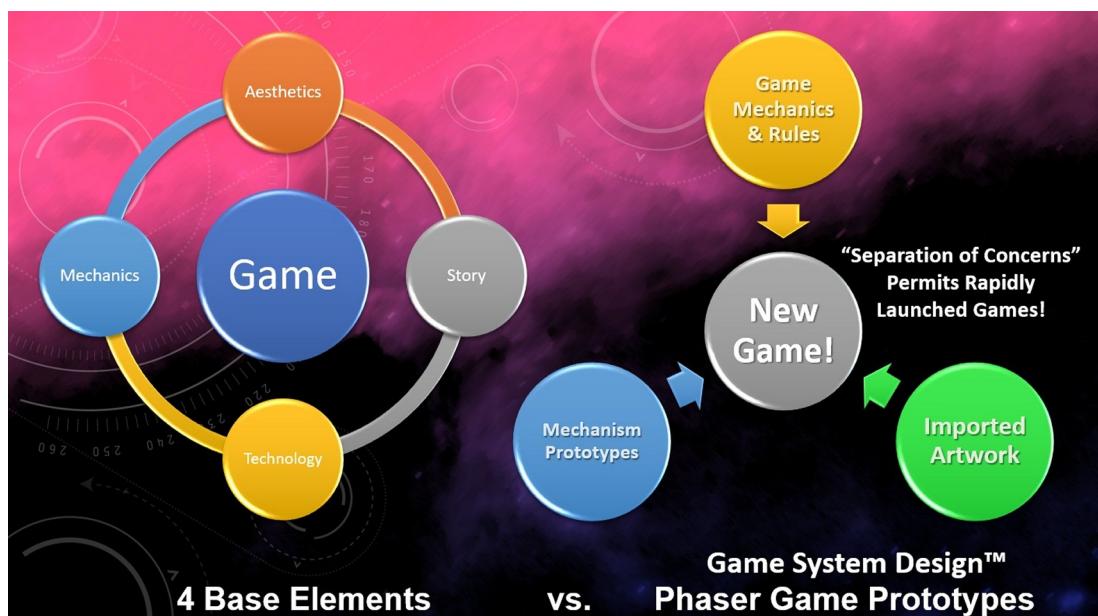
Therefore, **game control mechanisms** (e.g., buttons, mouse, touch-screen) are more of an engineering **programming concept** while **Game-Play** is more of a design **heuristic (aka rules-based) concept**.



**Note:** "Game Genres" are specific games categories related by **similar Game-Play characteristics**.

## 6.2 Game Mechanics (GM)

In the field of ***Iudology (aka, game studies)***,<sup>7</sup> the term “**Mechanics**” (11-pages Bonus Content)<sup>8</sup> refers to the inner workings of a game, that includes its rules, goals, and any other procedures. Other sources define Game Mechanics in this way — “... **game mechanics are methods invoked by agents, designed for interaction with the game state.**”<sup>9</sup> Mechanics set games apart from other media (i.e., stories, novels, theater, TV or cinema). Games mechanics create a dynamic, interactive, story-style experience for the gamer as a consequence of their decisions and actions. “**Game mechanics are the core of what a game truly is. They are the interactions and relationships that remain when all of the aesthetics, technology, and story are stripped away.**” (**Schell pg 130**) It is also one of the four elements within **Jesse Schell’s<sup>10</sup> Elemental Tetrad<sup>11</sup>** mentioned in his book — “**The Art of Game Design: A Book of Lenses**”<sup>12</sup>. (a mandatory read for any serious game developers)



*Jesse Schell Elemental Tetrad compared to our recommendations!*

Schell’s book pontificates many excellent theories and insights about ludology. Yet, reading through his 518 pages, you’ll not find “how” to implement his recommenda-

<sup>7</sup>[https://en.wikipedia.org/wiki/Game\\_studies](https://en.wikipedia.org/wiki/Game_studies)

<sup>8</sup><http://makingbrowergames.com/book/IntroducingAppliedLudology.pdf>

<sup>9</sup><http://gamestudies.org/0802/articles/sicart>

<sup>10</sup>[http://en.wikipedia.org/wiki/Jesse\\_Schell](http://en.wikipedia.org/wiki/Jesse_Schell)

<sup>11</sup><http://game-studies.wikia.com/wiki/Game>

<sup>12</sup><https://amzn.to/2CRTnM8>

tions into software. He offers his excuse as, "Since these exist largely in the darkness of the subconscious mind, it is hard for us come up with a well-defined analytical taxonomy of how they work." (Schell, pg 130) ***Mmmmm, uhm, 'nuf said? Nope!***

Schell	Phaser Game Prototypes in Game Design System™
Aesthetics =	Is the separated theme artwork
Technology =	game prototypes & entity components
Mechanics =	newly inserted rules as JSON governing game flow
Story =	gamer's self-directed adventures during game-play

## Game Mechanics Suggested by Schell



***Hint:*** I have rearranged Schell's Game Mechanics recommendations alphabetically because he provides no rational reason for them to follow the sequence he presents in his book.

### Actions Game Mechanics:

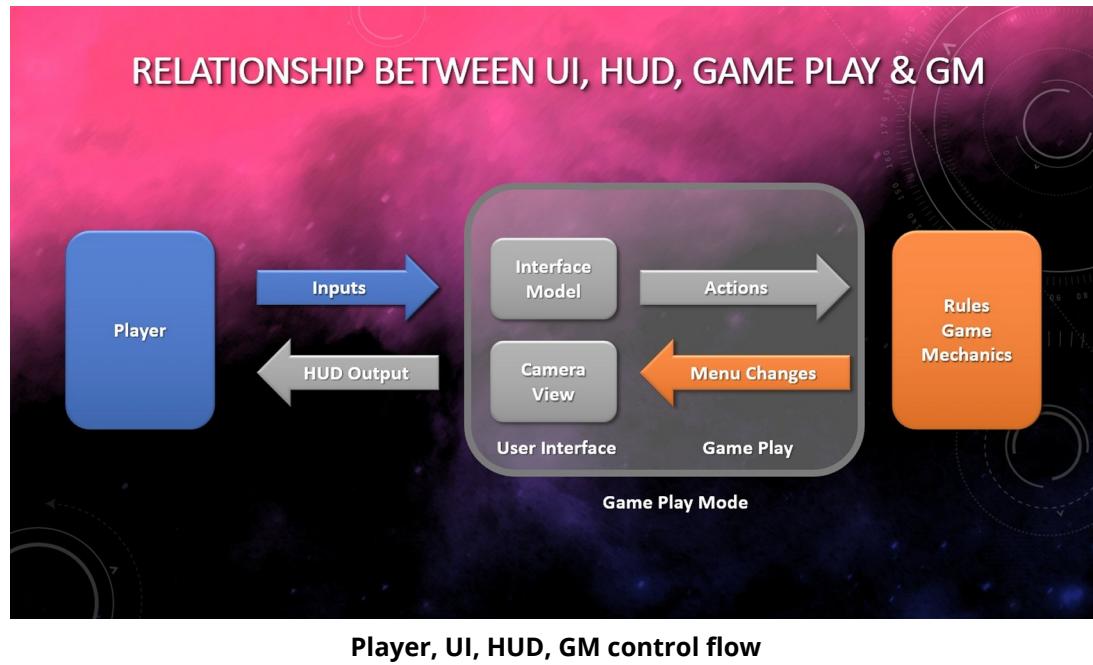
Gamers use ***action verbs*** in a game just as learned from Chapter 1 when dissecting a game's elevator speech. For example in a game of checkers the player can "move forward", "jump" an opponent's piece and a King can "move backwards".

A gamer's actions exposes their tactics and gaming strategies. The results solicit defensive counter-actions from their opponent and so the game continues with each one taking offensive or defensive moves.

We "deconstructed" games in previous chapters. The "actions" a gamer selects to perform are directly related to the "User Interface (UI)" displayed — the provided menus, buttons and draggable items — in short the game's mechanisms and components that we created in Chapter 4. Allowing a combination of actions could lead to "***emergent gameplay***".<sup>13</sup>

---

<sup>13</sup>[http://game-studies.wikia.com/wiki/Emergent\\_Systems](http://game-studies.wikia.com/wiki/Emergent_Systems)



## Game Mechanics as: Attributes, Objects, & States

Schell states, “... anything that can be seen or manipulated in your game ...” is a member of this Game Mechanics; they are the “nouns” — all the “entities” and components we’ve already constructed in Part I. We won’t follow his advice here; we want to keep our “Game Prototypes” (the visual elements) separate from our “Game Mechanics” (their data structures) — ***we’re using “separation of concerns”***. All these “dumaflaches” (i.e., technical terminology for objects) have attributes and states. Attributes are the descriptive information about an object — the “adjectives” or its “properties”. The current information held inside an attribute is the state of that object’s attribute — the data value the property contains. Attributes can be static or dynamic; for example, an avatar’s hit points during combat injuries or healing afterwards. Schell suggests using a finite state machine (FSM) to monitor the changes in attributes. (See the following Artificial Intelligence chapter.)

## Deeper Dive: Game Phases Revisited

### Apple's GameplayKit

Quote: “When you start building a game, it’s easy to put all the state-dependent

code in one place — **for example, in the per-frame update method** (ed.: ... sound familiar? ***It should; it's Phaser's approach!***) of a SpriteKit game. ***However, as your game grows and becomes more complex, that single method can become difficult to maintain or extend further.*** Read more from ***Apple Game Developers: State Machine***<sup>a</sup>

<sup>a</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/StateMachine.html#/apple\\_ref/doc/uid/TP40015172-CH7-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/StateMachine.html#/apple_ref/doc/uid/TP40015172-CH7-SW1)

## Deeper Dive: **StateManager**

Quote from Phaser III Dev Log #67 — ***StateManager***:<sup>14</sup> “The State Manager is responsible for loading, setting up and switching game states. ... You must give each State a unique key by which you'll identify it. The State can be either a Phaser.State object (or an object that extends it), ***a plain JavaScript object or a function***. If a function is given a new state object will be created by calling it. ... ***A State is considered valid if it has at least one of the core functions: preload, create, update or render.***” (***NOTE: written before Phaser III renamed “states” to “scenes”***)

## Deeper Dive: Object Manipulation in ES5/6/7/8/9

Objects had a major overhaul in ES6. Things like object deconstructing and rest/spread operators made working with objects very easy. Here's a comparison between ES5 code and trying to merge two objects.

```

1 //ES5 method
2 var obj1 = { a: 1, b: 2 }
3 var obj2 = { a: 2, c: 3, d: 4}
4 var obj3 = Object.assign(obj1, obj2)

```

ES5 used `Object.assign()` which takes both objects as input and outputs a newly merged object. Here's the same effect in ES6.

---

<sup>14</sup><https://phaser.io/phaser3/devlog/67>

```

1  //ES6 method
2  const obj1 = { a: 1, b: 2 }
3  const obj2 = { a: 2, c: 3, d: 4}
4  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 noticed by now that there are segregated `preload`, `config`, `create`, and `updates` functions throughout the global window. Why not simply write them once and "**Merge**" them like this.

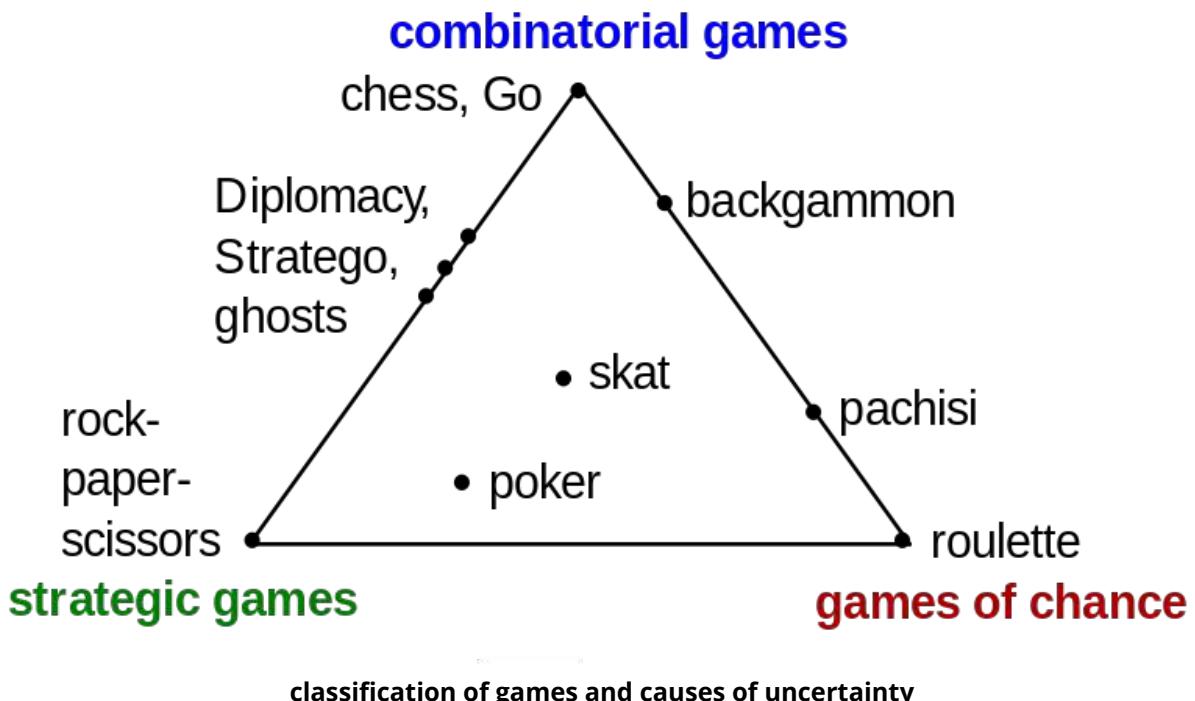
```

1  var preload = { loads all game assets };
2  var create = { assigns cached assets to scene }
3
4  //Game Mechanics is an external resource to dynamically change game-play
5  var GameMechanics = Object.assign(preload, create);
6  var scene1 = Object.assign(GameMechanics, Phaser.Scene);

```

### Chance Game Mechanics:

Chance (aka randomization) provides games with unpredictability, and as a results increases "replay value", and provides "surprises" — the secret sauce of "fun".





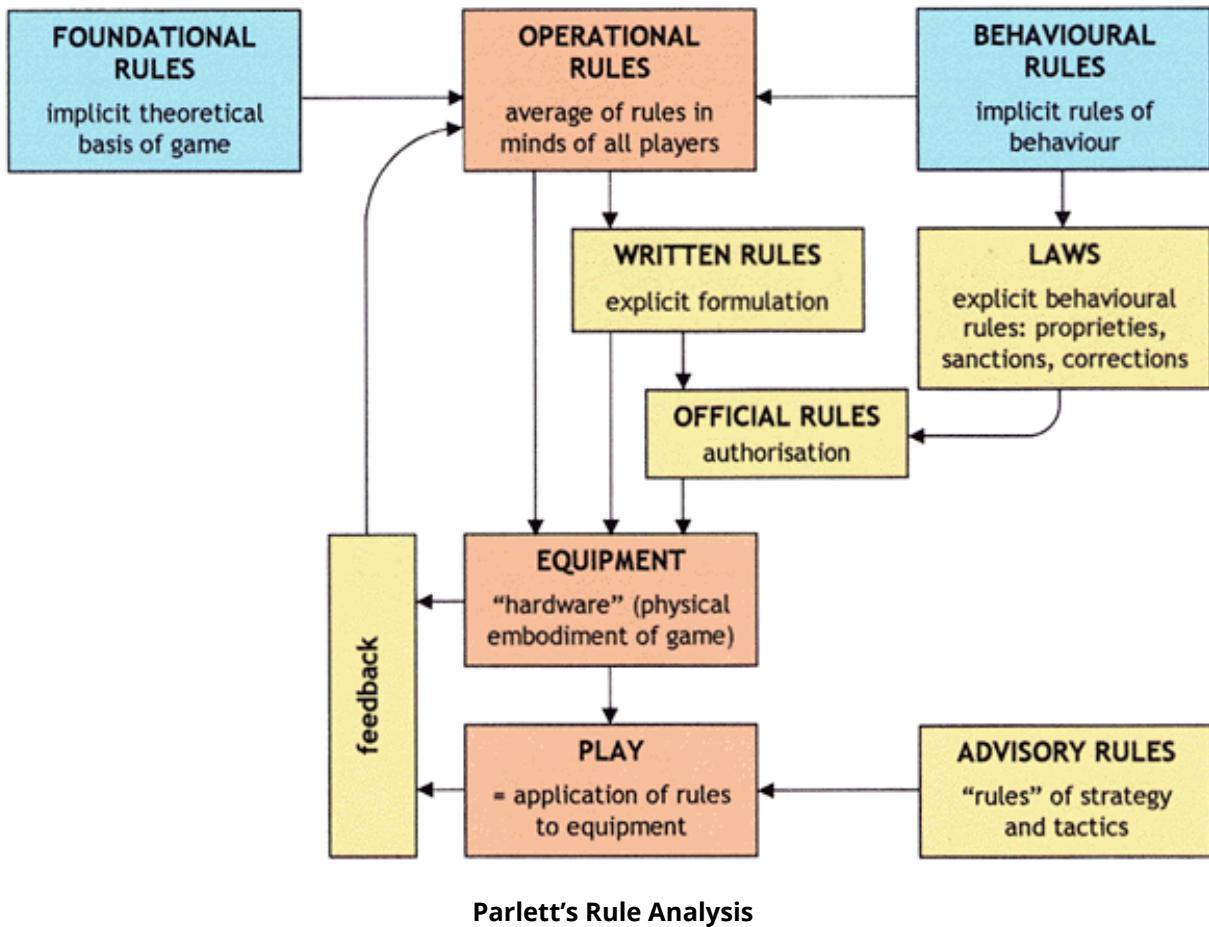
**Exercise:** Review Apple Game Developer's recommendations on **Randomization**. **QUOTE:** "To build robust pseudo-random behavior in a game, **you typically want some or all of the following traits . . .**"<sup>15</sup>

## Rules Game Mechanics:

Rules (**aka "heuristic"**) are the most fundamental of all game mechanics. They restrict the game's space (i.e., Phaser.Camera), define the location of entities and components (i.e., sprite's x and y coordinates), monitor and interpret the UI input actions (i.e., Phaser.InputManager) and adjust the data states of various game entities' attributes and properties. In short all other mechanics are built upon "rules" and provide what all **games genre must have**<sup>16</sup> — **goals**. "A game is not just defined by its rules, a game **is** its rules." (Schell, pg 148) Schell defers to the game historian, David Parlett and his analysis of "rules" illustrated below.

<sup>15</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/RandomSources.html#/apple\\_ref/doc/uid/TP40015172-CH9-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RandomSources.html#/apple_ref/doc/uid/TP40015172-CH9-SW1)

<sup>16</sup>[https://en.wikipedia.org/wiki/List\\_of\\_video\\_game\\_genres](https://en.wikipedia.org/wiki/List_of_video_game_genres)



**Exercise:** Read [this article](#)<sup>17</sup> in which Parlett explains his illustration above.



**Exercise:** Study how [Performing a Heuristic Analysis of Your Game](#)<sup>18</sup> can benefit your game design.

## Deeper Dive: Rules

For us, our rules will manage **a dynamic options menu (ui)**, respond to physics collision, overlaps, and intersections from various objects. Rules will take the respon-

<sup>17</sup><http://www.parlettgames.uk/gamester/rulesOK.html>

<sup>18</sup>[https://www.gamasutra.com/blogs/SuchaaverChahal/20151211/261574/Performing\\_a\\_Heuristic\\_Analysis\\_of\\_Your\\_Game.php](https://www.gamasutra.com/blogs/SuchaaverChahal/20151211/261574/Performing_a_Heuristic_Analysis_of_Your_Game.php)

sibility of adjusting the various objects attribute properties and states.

Consider this scenario: What if you retained all of a game's prototypes, mechanisms, components and artwork; **but, only replaced its Rules Game Mechanic???** What could we create?

Here's what I'm suggesting. Imagine a side-scrolling platform game which uses a main character spriteSheet. In the spriteSheet, we have soldiers, mages, clerics, thieves. We load all the game prototypes and artwork ***but selectively upload the Game Rules that apply to a gamer's specifically chosen character's skill set, with further considerations whether that gamer has a free or paid membership access!*** We have dynamically tailored our game based on the gamer's choice ***by simply exchanging the Rules.***



**Exercise:** Study ***Game Usability Heuristics (PLAY) For Evaluating and Designing Better Games: The Next Iteration***<sup>19</sup>



**Exercise:** Research Apple's recommendations on "***Designing Rule Systems***".<sup>20</sup>

## Deeper Dive: Rule Categories

- **Goals:** — are the general type of victory conditions a gamer must achieve to win.
- **Loss Avoidance:** — some game goal state a losing condition; for example, capture the flag.
- **Piece Elimination:** — is a goal found in capture game mechanics. The player wins by eliminating a specified opponent piece or all opponent's tokens.
- **Puzzle Solving:** — is a goal for the player to solve or reassemble. Jigsaw puzzle are the first game mechanic that comes to mind, but games such as "Clue" (in the UK "Cluedo") are included in this category.
- **Quests:** — are generally found role-playing games and is an adventure that a gamer's party must pursue and return.
- **Races:** — are goals to be the first player to reach a finish position.
- **Structure Building:** — The goal of a structure building game is to acquire and assemble a set of game resources into either a defined winning structure or into a structure that is somehow better than those of other players. In some games,

<sup>19</sup><http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.588.2260&rep=rep1&type=pdf>

<sup>20</sup>[https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit\\_Guide/RuleSystems.html#/apple\\_ref/doc/uid/TP40015172-CH10-SW1](https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/RuleSystems.html#/apple_ref/doc/uid/TP40015172-CH10-SW1)

the acquisition is of primary importance (e.g. concentration,)<sup>21</sup> while in others the resources are readily available and the interactions between them form more or less useful structures (e.g. poker).<sup>22</sup>

- **Territory Control:** — A winner may be decided by which player controls the most “territory” on the playing surface, or a specific piece of territory. This is common in wargames<sup>23</sup> but is also used in more abstract games such as Go.<sup>24</sup>
- **Victory Points:** — A player’s progress is often measured by an abstract quantity of **victory points** or simply known as **VP**, which accumulate as the game develops. Victory points or similar quantities need not be restricted to development games, but are most common in that type as they ensure sufficient reward for all aspects of development. For example, in a game involving the development of civilizations, there is usually no need to reward investments such as trade and military expenditures, which yield their own strategic benefits. However, a victory point system may be used to reward more subjective aspects of civilization-building, such as the arts. This mechanism is often used explicitly in German-style board games, but many other games are played for points that form a winning condition. The electoral college of the United States political system is also a well-publicized example of this type of victory condition. Victory points may be partially disguised in the role of game resources, with play money being a common example.



**Exercise:** Research board game mechanics at <https://www.boardgamegeek.com/browse/boardgamedemechanic>.

## “Skills” Game Mechanics:

This game mechanics is beyond the skill set of the avatar character played; it considers the physical, mental and social skills of the controlling gamer. Sampling the gamer’s responses, we can adjust the various rules, and interactions. The “... intrinsic skill required and stress ... required by a challenge is defined as the level of skill needed to surmount the challenge ***if you give the player an unlimited amount of time in which to do it.***” (pg 259 **Fundamentals of Game Design** 2nd Ed. by Ernest Adams)

---

<sup>21</sup>[https://en.wikipedia.org/wiki/Concentration\\_\(game\)](https://en.wikipedia.org/wiki/Concentration_(game))

<sup>22</sup><https://en.wikipedia.org/wiki/Poker>

<sup>23</sup><https://en.wikipedia.org/wiki/Wargaming>

<sup>24</sup>[https://en.wikipedia.org/wiki/Go\\_\(board\\_game\)](https://en.wikipedia.org/wiki/Go_(board_game))

## “Space” Game Mechanics:

Schell suggests, that after all visuals are stripped away, what remains is the “game space”. Schell relies heavily on the **15 properties of physical architecture**<sup>25</sup> to describe this game mechanic.

For us, the “game space mechanics” is the `camera view`<sup>26</sup> — the player’s immediate view into the larger world environment. We can control the `camera` using either `this.camera` from any game phase or from `game.camera` since it was globally defined. The camera will only display those objects within its “Field of View” (FoV).



**Exercise:** Review the various `camera` examples from <http://labs.phaser.io/index.html?dir=camera/&q=>

## 6.3 Phaser III API into Game Mechanics (GM)

Game mechanisms (i.e., software routines and input controls) fall into several more or less well-defined categories, along with basic Game-Play, game heuristics, mode, and genre. Game mechanisms help define game rules and players’ Game-Play strategies. This is the **“Bottom-up Design”** — understanding the various pieces found inside Phaser III and using them to build up the Game Mechanics (GM) components. Here are a few suggestions for game mechanisms in Phaser III that directly support Game Genre Mechanics:

**GM: Actions** — Deconstructed actions become the Game Prototypes and Components. They are the visual and abstract components of the User Interface (UI).

- **Input Manager**<sup>27</sup> — The Input Manager is responsible for handling all Input subsystems and native browser events — Scene Manager, Texture Manager and Sound Manager. These systems are global and are designed to interface through plugins. Do not directly access or adjust them. Also looks after Input enabled Game Objects. `Phaser.Input` is the Input Manager for all types of Input across Phaser, including mouse, keyboard, touch, and MSPointer. The Input manager is updated automatically by the core game loop. Use `game.input` to access the global Input Manager or preferably `this.input.on` from within any Scene.

<sup>25</sup><http://www.tkwa.com/fifteen-properties/>

<sup>26</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Cameras.Scene2D.CameraManager.html>

<sup>27</sup><http://labs.phaser.io/index.html?dir=input/&q=>

- **InputHandler**<sup>28</sup> — “If a Game Object is enabled for input this class controls all input related events, including clicks and drag. The Input Handler is bound to a specific Sprite and is responsible for managing all Input events on that Sprite.”
- **Gamepad Manager**<sup>29</sup> — “The Gamepad Manager looks after all connected Gamepads to the device. Creates SinglePad instances. The Gamepad class handles gamepad input and dispatches gamepad events.”

### **GM: Objects' Attributes & States**

- **Data loaded**<sup>30</sup> from JSON files and controlled by `Phaser.Loader`, maintained in `Phaser.Cache`. Automatically invoked by each game phase `preload` (Essential Function).
- Create (Essential Function) — “The Phaser.Create class is a collection of smaller helper methods that allow you to generate game content quickly and easily, without the need for any external files. You can create textures for sprites and in coming releases, we'll add dynamic sound effect generation support as well (like `sfxr`).”
- **GameObjectFactory**<sup>31</sup> — “The GameObjectFactory is a quick way to create many common game objects using `game.add`. Created objects are automatically added to the appropriate `Manager`, `World`, or manually specified parent group.”
- **GameObjectCreator**<sup>32</sup> — “The GameObjectCreator is a quick way to create common game objects without adding them to the game world. The object creator can be accessed with `game.make`.”
- **Game States Manager**<sup>33</sup> — “The State Manager is responsible for loading, setting up and switching game states.” `Phaser.State` “... provides **quick access** to common functions such as the camera, cache, input, match, sound and more.”

### **GM: Chance**

- Phaser.Math uses Raw JS Math library. Recommendation to preserve on memory and delegation processing, just use the JS Math library directly. Read through the offered Phaser.Math functions to build your own.

### **GM: Rules**

<sup>28</sup><https://github.com/photonstorm/phaser/blob/v3.14.0/src/input/InputManager.js#L352>

<sup>29</sup><https://photonstorm.github.io/phaser3-docs/Phaser.Input.Gamepad.Gamepad.html>

<sup>30</sup><http://labs.phaser.io/index.html?dir=components/data/&q=>

<sup>31</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObjectFactory.html>

<sup>32</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObjectCreator.html>

<sup>33</sup><https://phaser.io/phaser3/devlog/67>

- Physics Manager — **as Parlett's Behavioral (both implicit and explicit "Laws"), Equipment, and Play Rules** — “The core Physics Manager. Provides access to all of the physics sub-systems. The Physics Manager is responsible for looking after all of the running physics systems. Phaser III supports 3 physics systems: Arcade Physics, ImpactJS, and MatterJS. Game Objects (such as Sprites) can only belong to 1 physics system, and you can only have one systems active in a single game scene. For example, you could have Matter managing a polygon-built terrain landscape that a vehicle drives over, while it could be firing bullets that use the faster (due to being much simpler) Arcade Physics system found in a different scene.” If you were a Flash Developer, then using Phaser III Scenes is similar to using MovieClips on the timeline.

**GM: Skills — lacking in Phaser.Game.** It’s up to the game designer to use tools such as **SCORM**<sup>34</sup> and **TINCAN**<sup>35</sup> to track and evaluate the gameplay activity. Using these tools, the game difficulty could dynamically adapt to the player’s demonstrated skills.

### **GM: Space or “Area of Interest”**

- Camera — gamer’s view of the “world Area of Interest” (AOI). “A Camera is your view into the game world. It has a position and size and renders only those objects within its field of view. The game automatically creates a single Stage sized camera on boot. Move the camera around the world with Phaser.Camera.x or y”
- Game — The Game object is the heart of your game, providing quick access to common functions and handling the boot process.
- Game Scaling Manager — “Manages the sizing and scaling of your game across devices.”
- Stage — root display of gaming space area. It also handles browser visibility handling and the pausing due to loss of focus.
- System (Canvas) — “The Canvas class handles everything related to creating the canvas DOM tag that Phaser will use, including styles, offset and aspect ratio.”
- TileMaps — “Creates a new Phaser.Tilemap object. The map can either be populated with data from a JSON or a CSV file. A Tilemap consists of one or more TilemapLayers and associated tile data. Contains methods for tile data manipulation and TilemapLayer generation. A single layer within a Tilemap. Extends from Phaser.Sprite and is responsible for rendering itself.”
- World — “It is not bound by stage limits and can be any size. You look into the world via cameras. All game objects live within the world at world-based coordinates. By default a world is created the same size as your Stage.”

---

<sup>34</sup><https://scorm.com/scorm-explained/>

<sup>35</sup><https://elearningindustry.com/scorm-vs-tin-can-api-whats-difference>

**As you can see the Phaser III JavaScript Gaming Framework performs many of Schell's Game Mechanics with the exception of GM Rules and GM Skills.**

## Deeper Dive: Input Manager Event Horizon

### Quote Phaser Newsletter #118

Phaser 2 used what are known as Signals for most of its communication from systems to user-land code. If you've ever written code such as `onInputDown.add` then you've been interfacing with a Signal. Signals are like Events but have a different internal structure and approach. There's an *interesting article*<sup>a</sup> on the subject here and I used to really enjoy using Signals back in the Flash / AS3 days. However, for v3 we didn't carry this through. There are a few reasons for this but the primary one is that Phaser is a JavaScript based library, and on the most part, when in Rome you should do like the Romans do. For JavaScript, and the web in general, that means using events.

In v3 we are using the *EventEmitter3 library*,<sup>b</sup> which is proven to be both insanely fast and flexible enough for our needs during development and beyond. We use it internally for systems to communicate with each other and also lots of systems, and indeed the core GameObject class, all extend from it.

... as you can appreciate can be insanely powerful and opens up all kinds of opportunities for you. You no longer have to create masses of Signal instances and can instead always rely on the fact that a GameObject can both emit and receive events.

<sup>a</sup><http://blog.millermedeiros.com/callbacks-promises-signals-and-events/>

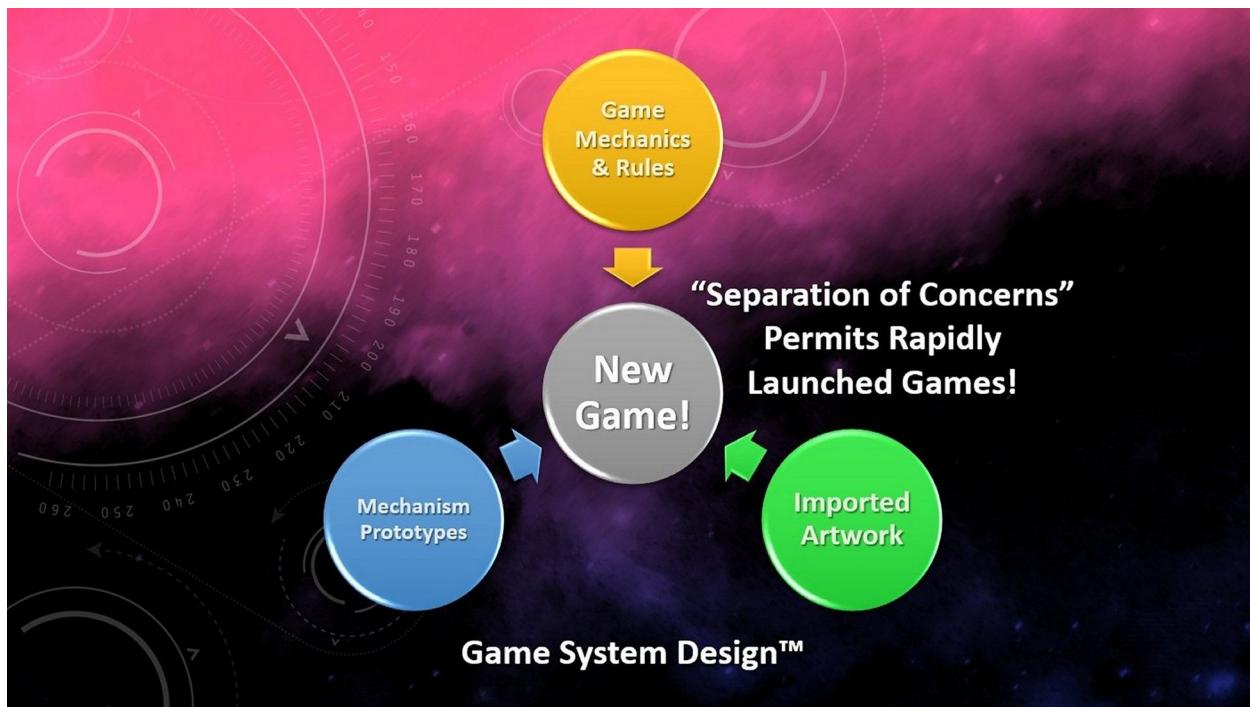
<sup>b</sup><https://github.com/primus/eventemitter3>

## 6.4 Game Design System™

**Generic programming** centers around the idea of abstracting from concrete, efficient algorithms to obtain **generic algorithms** that can be **combined with different data representations** to produce a **wide variety** of useful software.

— Musser, David R.; Stepanov, Alexander A., *Generic Programming*<sup>a</sup>

<sup>a</sup>[https://en.wikipedia.org/wiki/Generic\\_programming](https://en.wikipedia.org/wiki/Generic_programming)



Assembling the blocks

**Lost Garden's article<sup>36</sup>** has hinted about the **Game Design System™** proposed in this book since 2006.

<sup>36</sup><http://www.lostgarden.com/2006/10/what-are-game-mechanics.html>

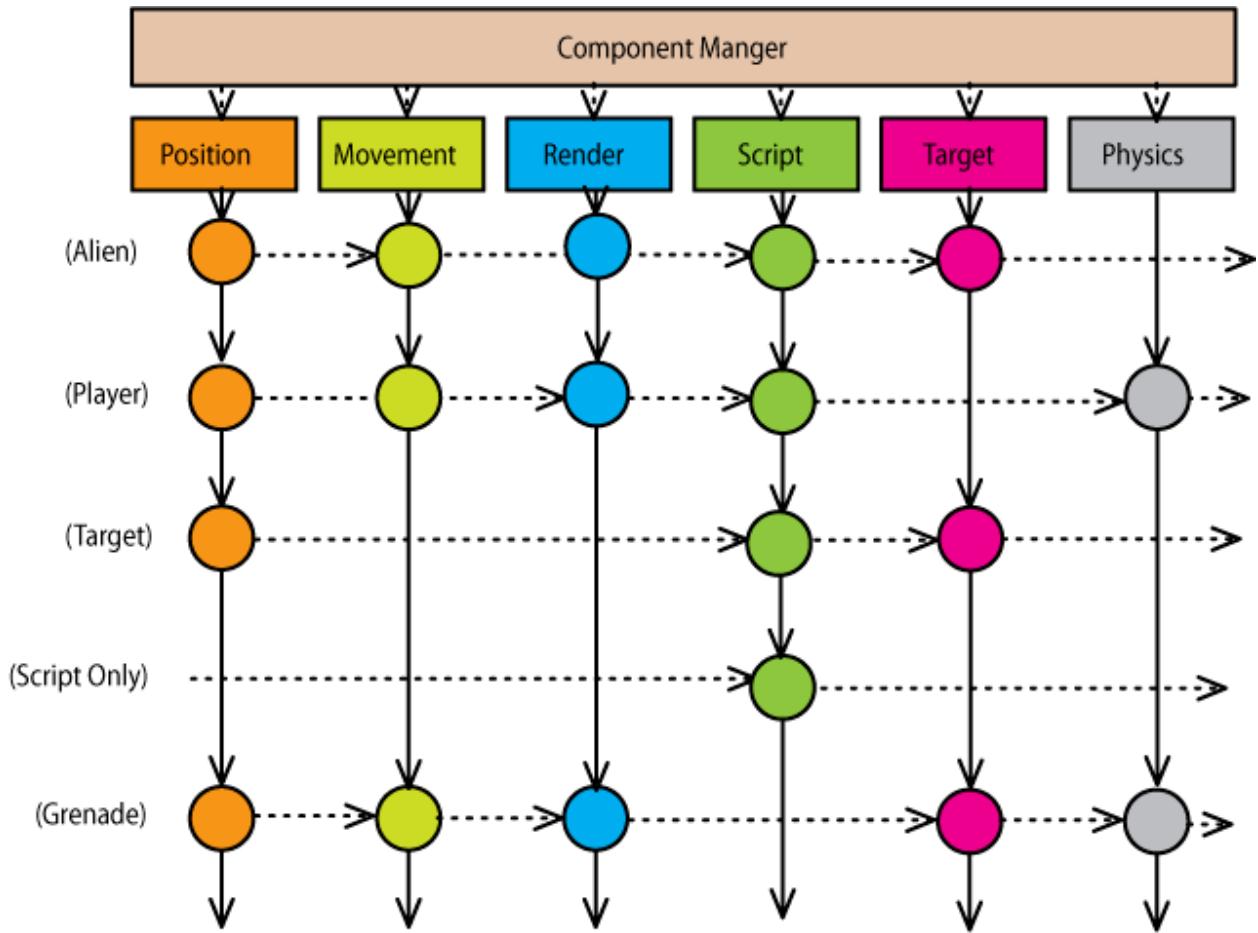


Figure 2 Object composition using components, viewed as a grid.

#### Entities and Component Game Design viewed as a cross reference

The goal I'm suggesting is building a codebase full of components that easily interlocks like "**Lego**" blocks.<sup>37</sup> The **Game Design System™** becomes a true skeletal framework/engine with "new muscles (**prototype components**),<sup>38</sup> organs (mechanics) and flesh (artwork)."



**Exercise:** Read <https://coronet.iicm.tugraz.at/sa/scripts/lesson01.htm>

<sup>37</sup>[https://en.wikipedia.org/wiki/The\\_Lego\\_Group](https://en.wikipedia.org/wiki/The_Lego_Group)

<sup>38</sup>[https://en.wikipedia.org/wiki/Component-based\\_software\\_engineering](https://en.wikipedia.org/wiki/Component-based_software_engineering)



**Note:** "Paradigm" is a Greek word meaning "example". It commonly refers to categories of things that have common characteristics. **Most Senior Software Engineers understand several programming paradigms<sup>39</sup> and can quickly adapt between them and write source code.**

## How it works

We have a couple of choices on how to build our game mechanics into our games.

1. **Traditional method:** by which we **sow<sup>40</sup>** our game mechanics code, throughout the game's "fertile fields", hoping for a bountiful harvest of fun and entertainment from such planting. Spreading our game mechanics code throughout allows us to **tightly couple<sup>41</sup>** them **inside each game phase and essential function. Everyone constructs their games this way now-a-days,<sup>42</sup> so, why should we be any different?** In other words by **following the crowd,<sup>43</sup>** we lose the flexibility I'm suggesting in **this new approach in the Game Design System™ — the flexibility** to exchange new game mechanics into existing game frameworks, components, and artwork themes.
2. **Game Design System™ method (for a new project)** by which we collect **all game mechanics (suggested above)** and consolidate them into separate JS Module(s) — just as we did for **our game phases in Chapter 3**. Doing so, provides the **ultimate "separation of concerns"<sup>44</sup>** and keeps our senior software engineers "happy" by following what they are so familiar with **in Java — The Principles of Good Programming.<sup>45</sup>**
3. **Migrating** former game project into the **Game Design System™ method** is our third (3rd) option. This method takes former game projects and reviews their code. When we find an embedded game mechanic, we tag it with a comment and note where the code is located and what it is doing. What do we hope to gain? The recapture of work invested, and the ability to expand that single game into a series of sister games by simply exchanging mechanism, components, artwork and/or game rule mechanics. Revisiting our "Breakout" game from chapter 1 and chapter 2, let's take that prototype and exchange the artwork and retain everything else. Using the "Break Out!" game components and GMs we've created **this new RPG Battle game<sup>46</sup>** (total conversion time to replace old artwork for new artwork: 5 minutes!! **A new game of similar genre in 5 minutes!**)

<sup>39</sup>[https://en.wikipedia.org/wiki/Comparison\\_of\\_programming\\_paradigms](https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms)

<sup>40</sup><https://www.vocabulary.com/dictionary/sow>

<sup>41</sup><https://blog.interfaceware.com/integration-best-practices-separation-of-concerns/>

<sup>42</sup><https://www.grammarly.com/blog/nowadays-or-now-days/>

<sup>43</sup><https://www.psychologytoday.com/us/blog/after-service/201705/the-science-behind-why-people-follow-the-crowd>

<sup>44</sup><https://hackernoon.com/separation-of-concerns-by-files-dan-abramov-56d0d6da842e>

<sup>45</sup><https://java-design-patterns.com/principles/>

<sup>46</sup><http://makingbrowsergames.com/p3devcourse/standard/argh.html>

**Example 1: Compare what we did with what is possible in Game Design System™ using Game Recipes™. Play Chapter 1 Break Out!**<sup>47</sup>



**Note:** This third method is the one I'm personally using to migrate my released Adobe Flash game collection — **88 total!** How I do this is detailed in the **Phaser III Game Starter Kit Collection — 16+ Classic Game Mechanics from popular Phaser III Gaming Framework**<sup>48</sup>

**Sample Game Mechanics with inserted Migration Comments.**

---

```

<script>
    //GM: Attributes, Objects, & States Game Mechanics:
    //standard Phaser III launch method
    game = new Phaser.Game(config);

    function preload() {
        //GM: "Space" Game Mechanics:
        // ======
        // Example 3.4: Additional Phaser Properties begins
        // ======
        // remote URL to game assets
        // Cross-origin resource sharing (CORS)
        this.load.setCORS = 'anonymous';
        this.load.setBaseURL('http://makingbrowergames.com/p3devcourse/_p3demos/img\\
ages/');
        console.log("Additional Phaser Properties set in preload!");
        //Example 3.4: ends
        // ======
    }

    //GM: Attributes, Objects, & States Game Mechanics:
    function create() {};

    //GM: Attributes, Objects, & States Game Mechanics:
    function update() {};

</script>

```

---

The sample above is composed entirely of “Game Mechanics” snippets and instead of using the typical game phase “launch.js”, we could **relegate all of this code**<sup>49</sup>

<sup>47</sup>[http://localhost/\\_GIS/GISUS-MakingBrowserGames/makingbrowergames.com/p3devcourse/standard/lesson15.html](http://localhost/_GIS/GISUS-MakingBrowserGames/makingbrowergames.com/p3devcourse/standard/lesson15.html)

<sup>48</sup><https://leanpub.com/p3gskc>

<sup>49</sup><https://www.vocabulary.com/dictionary/relegate>

into a separate file dedicated to just ***the “Game Mechanics” (GM) — perhaps named gMech.js?*** Doing so, would permit us to dynamically exchange the GM Rules, Space and composition of the game.

**Example 2:** [http://makingbrowsergames.com/book/p3gp-book/\\_p3demos/Ch5-game.js](http://makingbrowsergames.com/book/p3gp-book/_p3demos/Ch5-game.js)



**Exercise:** Download and search for “// GM ” in the “***Ch5-game.js***” file. Count how many time and where these game mechanics (*gm*) are embedded; ***you'll discover a pattern emerging.*** You'll further learn that Jesse Schell ***has gone over-the-top***<sup>50</sup> (*ott*)<sup>51</sup> with his game mechanics definitions. To make this feasible for our ***Game Design System™ method***, we could eliminate several of his recommendations based on a software implementation.



**Exercise:** Time to learn what the senior software engineers already know. Read ***The Principles of Good Programming***<sup>52</sup>



**Exercise:** Read the “Deeper Dive” on Separation of Concerns: ***“An architectural approach with separation of concerns to address extra-functional requirements in the development of embedded real-time software systems”***<sup>53</sup>

From these articles, you'll find that Game Prototypes and Components are the visual and abstract components of the User Interface (UI). Game Mechanics handle the game's behavior — modifying attributes and state variables, regulating avatar capabilities and interactions, providing random and balanced statistics, monitoring goal achievement — all within the space of the camerview. The Game's theme and artwork provides the “mood”.

## 6.5 Game Genres

**Game Genres**<sup>54</sup> are specific game categories related by ***similar gameplay characteristics.*** Remember that gameplay are the “rules of a game”. Single-player and multi-player are therefore not game genres. They are a “delivery” mechanism dictating the “mode” of gameplay.

<sup>50</sup><https://www.urbandictionary.com/define.php?term=over%20the%20top>

<sup>51</sup><https://dictionary.cambridge.org/us/dictionary/english/over-the-top>

<sup>52</sup><https://www.artima.com/weblogs/viewpost.jsp?thread=331531>

<sup>53</sup><https://www.sciencedirect.com/science/article/pii/S1383762114000824>

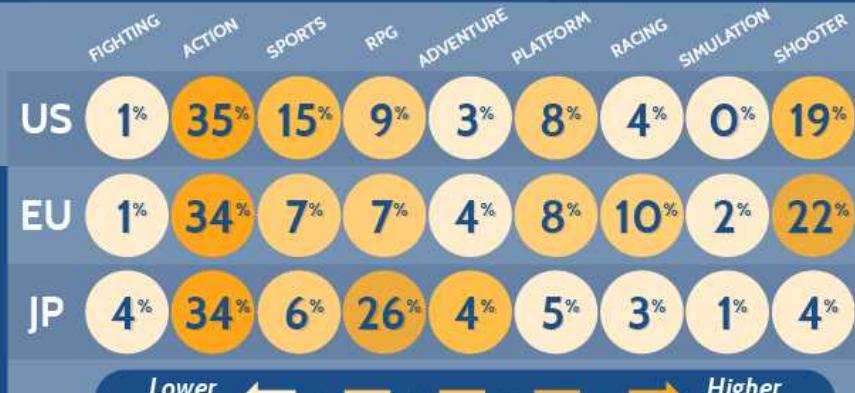
<sup>54</sup>[https://en.wikipedia.org/wiki/List\\_of\\_video\\_game\\_genres](https://en.wikipedia.org/wiki/List_of_video_game_genres)

# Game Genres Across The World

*Top Selling  
Video Game  
Genres by Region*

## ACTION GAMES

*are the highest  
selling genre  
in the world!*



*Lower Sales ← • — • — • — • — Higher Sales*

*Percentages reflect the number of games sold by genre of  
the top-selling 100 titles in each region for the year of 2013.*



*Find more statistics on video games at  
[bigfishgames.com/blog/stats/](http://bigfishgames.com/blog/stats/)*

Sources: VG Chartz Global 2013: <http://goo.gl/LSReHl> | VG Chartz Europe 2013: <http://goo.gl/STGkQb>  
VG Chartz Japan 2013: <http://goo.gl/a6upnm> | VG Chartz USA 2013: <http://goo.gl/HyXQX8>

### Game Genres Across the World



**Exercise:** Research the Big Fish's updated **Video Game Statistics Database**<sup>55</sup>



**Exercise:** Big Fish is the 7th largest game publisher world-wide. I am fortunate to have earned Silver Partner status. Big Fish has an extensive library on game development. Visit their **game development education site**.<sup>56</sup>

<sup>55</sup><https://www.bigfishgames.com/blog/stats/>

<sup>56</sup><https://www.bigfishgames.com/blog/category/game-development/>

## Deeper Dive: Game Genres

Game classification is diverse. "Just as game publishers have convoluted the terminology for "game mode", game genres are just as confusing. The inconsistency comes from trying to describe the game mechanics, game delivery and details associated with the game theme." **Phaser Game Starter Kit Collection**<sup>57</sup> or single chapters available from Amazon.com.

### GGInteractive – Game Design Course

Genres don't help market a game – instead, the selection of a genre has an effect on what (and what size of) audience is likely to be interested and willing to purchase a game. Genres do help an audience understand the basics of a game by promising them a certain amount of familiar elements they desire(or demand)in a particular genre. For example, fans of medieval fantasy will expect some common themes in a Fantasy RealTime Strategy game (FRTS): knights, castles, troop combat, and magic, to name a few. As a game designer, it is important to understand what kind of audience expectations exist across different genres. Stray too far from these expectations without designing a brilliant alternative and the game will lose its audience. Stick too close to what has come before and the game will be overlooked as offering nothing new. Even the bold designer that intends on rewriting all that we know about how a genre game is played, needs to understand what, to this point, has made the genre popular before deconstructing it and making it better.

Download Free Content: **Game Category Classifications Compared**<sup>58</sup>

**"Genres are not usually defined by the actual content of the game or its medium of play, but by its common challenge."** (pg 70, Fundamentals of Game Design) "The usual challenges are the rules which govern game play. Game genres are separate from their interfaces, management operating system platforms. ..." **Phaser Game Starter Kit Collection**<sup>59</sup>

## Deeper Dive: Game Modes

This is an often-misused term and is often confused with a **game's perspective**<sup>60</sup> or participants. Let's agree that game mode refers to game participation and the number

<sup>57</sup><https://leanpub.com/pgskc>

<sup>58</sup>[http://makingbrowsergames.com/starterkits/\\_GameCategoriesSubmissions.pdf](http://makingbrowsergames.com/starterkits/_GameCategoriesSubmissions.pdf)

<sup>59</sup><https://leanpub.com/pgskc>

<sup>60</sup>[https://en.wikipedia.org/wiki/Video\\_game\\_graphics#Top-down\\_perspective](https://en.wikipedia.org/wiki/Video_game_graphics#Top-down_perspective)

of players within the game session. Armed with this definition let's further define how gamers participate.

**Single Player** — A single-player game accepts input from only one participant, and refers to those games which can be played by only one active person. "Single-player mode" converts game-play into a single-player input. Furthermore, the "Single-player mode" **might have representation** as an avatar found in PacMan which shows the position and interaction of the player within the gaming world. Single-player is not restricted in representation as in arcade style games of Tetris. Many military strategy games allow **multiple avatars per single gamer** while battling the computer's artificial intelligence. Much of the online games, today, were designed for single-player mode. Here's an interesting and controversial article on ***the future of computer-monitored single-player games. Are single-player games doomed?***<sup>61</sup>

**Multi-Player** — This mode, as its name suggests, allows more than one game participant. Many game publisher mistakenly set Massive Multi-Player Online Games (MMoG) as a separate gaming genre. Yet, in my opinion, it is not. Massive Multi-Player Online Games (MMoG) is a **mode of play** couched within many different game genres just as single-player games have a varying genre. Gamers can play cooperatively in teams<sup>62</sup> or antagonistically as opponents<sup>63</sup> in this mode. Like single-player games, Multi-Player normally have avatar representation as "one player to one avatar" or "one participant with multiple avatars". The gaming party might have local or remote access to the single game session. Two gamers could play in "**hot-seat**" mode; in which, they would pass the input devices to the next player for their game-turn. This was popular in the early days of consoles gaming when local area networks were lacking or significantly diminished as compared to current modern-day network capabilities. Modern networks have provided the foundation for Massive Multi-Player Online Games. I have dedicated an entire section on various consideration in multi-player game development.

"Few gaming genres can claim to be as influential or lucrative as the **Multi-Player Online Battle Arena (MOBA)** can right now; even fewer started life as a mere custom map." Quote from: "**A brief introduction to gaming's biggest, most impenetrable genre.**"<sup>a</sup>

**"Multi-player online battle arena (MOBA)**, also known as **action real-time strategy (ARTS)**, is a genre of strategy video games that originated as a sub-genre of real-time strategy, in which a player controls a single character in one of two teams. The objective is to destroy the opposing team's main structure with the assistance of periodically spawned computer-controlled units that march forward along set paths.

<sup>61</sup><http://www.raphkoster.com/2006/02/10/are-single-player-games-doomed/>

<sup>62</sup>[https://en.wikipedia.org/wiki/Cooperative\\_game\\_theory](https://en.wikipedia.org/wiki/Cooperative_game_theory)

<sup>63</sup>[https://en.wikipedia.org/wiki/Non-cooperative\\_game\\_theory](https://en.wikipedia.org/wiki/Non-cooperative_game_theory)

Player characters typically have various abilities and advantages that improve over the course of a game and that contribute to a team's overall strategy. MOBA games are a fusion of action games, role-playing games, and real-time strategy games, in which players usually do not construct either buildings or units."(Wikipedia.org<sup>b</sup>)

<sup>a</sup><https://www.polygon.com/2013/9/2/4672920/moba-dota-arts-a-brief-introduction-to-gamings-biggest-most>

<sup>b</sup>[https://en.wikipedia.org/wiki/Multiplayer\\_online\\_battle\\_arena](https://en.wikipedia.org/wiki/Multiplayer_online_battle_arena)

## 6.6 Summary

You discovered that **Game Prototypes and Components** are the visual and abstract components of the User Interface (UI). **Game Mechanics** handle the game's behavior — modifying attributes and state variables, regulating avatar capabilities and interactions, providing random and balanced statistics, monitoring goal achievement — all within the space of the `camerview`. The Game's theme and artwork provides the "mood". The core process of the **Game Design System™** were these topics we covered:

- Comparing a game's mechanics to physical architecture frameworks.
- Contrasted Game Mechanics, Game Mechanism and Game Play.
- Study the formula to create "fun" and compared to 16 Human Motivations.
- Game Play = Human Computer Interaction (HCI) = rules, plot, objective, and interactions.
- Game Play can be "deconstructed".
- Game Genres are groupings of similar game mechanics rules.
- Read two Master Thesis on Game Mechanics.
- Learned the difference between games and other entertainment media.
- Compared Jesse Schell Art of Game Design to methods used to actually implement software solutions using Phaser Game Prototypes.
- Identified the 6 basic Game mechanics.
- Discovered that "Rules (aka "heuristic") are the most fundamental of all game mechanics."
- Learned that, "A game is not just defined by its rules, a game is its rules."
- Dissected various categories of rules.
- Studied how to "Perform a Heuristic Analysis" on games.
- Translated game theory into software code through "Deeper Dives" into rules.
- Found how to dynamically adjust game play mechanics to individual gamer's input.
- Cataloged various Goals for games genres.
- Discovered hints at using AI to adjust the game skills mechanics.

- Learned the source of Jesse Schell description for game space, and applied it to the Phaser camera.
- Read that the **Game Design System™** has been hinted about since 2006!
- Identified the three (3) methods to build and externalize game mechanics.
- Created JS modules to dynamically update and exchange game mechanics.
- Analyzed which game mechanics to export into its own JS module.
- Studied the “Principles of Good Programming”.
- Realized how to increase current game production through recapturing work formerly invested in released game products.
- Read another Master’s Thesis on “Separation of Concerns”.
- Discovered that “single-player” and “multi-player” are modes of delivery, and that they are not true game genres.
- Studied which game genres are popular in different target markets.
- Found the “Video Game Statistics Database” created by the 7th largest game publisher.



## 7. Dem's fightin' words

In this chapter we will develop several methods for conflicts resolution (also known as “combat”) between 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:

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch7-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/ch7-examples/)

### 7.1 Launching Web Sockets

We talk more about web sockets and how to use them properly in ***Phaser Multi-player Gaming Systems***<sup>1</sup>; but for now here’s how to include web sockets in your game products. You can also experiment with our ***MMOG server***.<sup>2</sup>

---

<sup>1</sup><https://leanpub.com/rrgamingsystem/>

<sup>2</sup><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         //OLOO style; note that false warning
41         //var ws = Object.create(WebSocket("ws://localhost:9998/echo"));
42
43         //=====
44         // 4 WebSocket Protocol Msg
45         //=====
46         ws.onopen = function onopen(event) {
47           // Web Socket is connected, send data using send()
48           ws.send("Test Message sent");
49           alert("Test Message away, away ... Captain!");
50         };
51         ws.onmessage = function onmessage(event) {
52           var received_msg = event.data;
53           alert("Incoming Messages ... brace for impact, Captain!");
54         };
55         ws.onclose = function onclose() {
56           // websocket is closed.
57           alert("Connection is closed...");
58         };
59         //=====
60         // End of WebSocket Protocol
61         //=====
62     } else {
63       // The browser doesn't support WebSocket
64       alert("WebSocket NOT supported by your Browser!");
65     }
66   }
67 </script>
```

---

**Note:** Refer to these resource files: <http://makingbrowsergames.com/p3gp-book/>

[\\_p3demos/ch7-examples/lesson01.html](http://p3demos/ch7-examples/lesson01.html)

## 7.2 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 scene.transition
617     // =====
618
619     // previously, Chapter 4 method moved to new page
620     // window.open("lesson11a.html", "_self");
621
622 /**
623 // Chapter 7.2 methods moves to 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       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.*<sup>3</sup>

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 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 v2.x.x, CE and III gives us more flexibility than Flash when creating Dynamic Menus. There are several design options we could use for games discussed in Part III; some of these options are:

1. OR, we could create “menu option sets” inside of **“containers”** (*Read more details both pros and cons here.*);<sup>4</sup> 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]);

```

<sup>3</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch7-examples/lesson02.html](http://makingbrowsergames.com/p3gp-book/_p3demos/ch7-examples/lesson02.html)

<sup>4</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container.html>

2. OR simply toggle these 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 camera
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.html
714
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 -

```

---

```

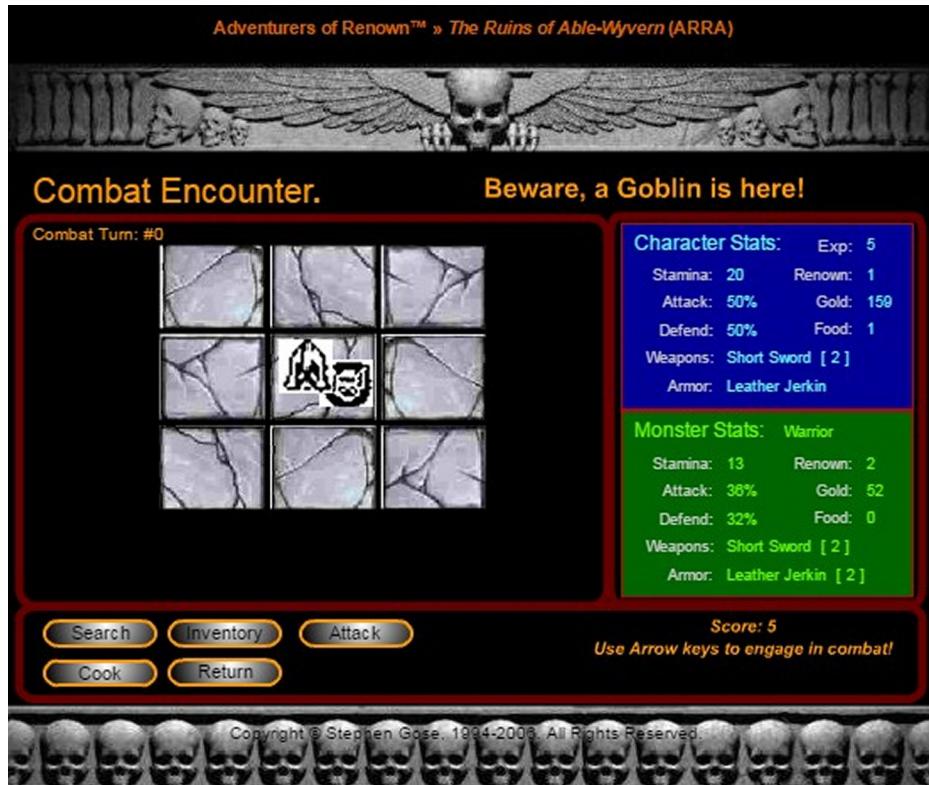
722 // i.e., retro dumb terminal or BBS style.
723 // =====
724 // Design Options #2
725 engageC.setX(0);
726 disengageC.setX(1000);
727 gameStatus = 1;

```

---

Review this source code at either:

- [http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch7-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/ch7-examples/)
- [http://makingbrowsergames.com/p3gp-book/\\_p3-bloodPitv1/](http://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/)



**Attack button only appears when engaged in melee Combat**

### 7.3 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***<sup>a</sup> or ***Monopoly***<sup>b</sup>, to say nothing of ***Checkers***<sup>c</sup> and ***Chess***<sup>d</sup>. 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.***<sup>e</sup>

<sup>a</sup><http://amzn.to/2l1flud>

<sup>b</sup><http://amzn.to/2mfZ5uZ>

<sup>c</sup><http://amzn.to/2mg3YUT>

<sup>d</sup><http://amzn.to/2l1qgJA>

<sup>e</sup><http://amzn.to/2m1WdWm>

***“Engage or Not to engage that is the question.”*** 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 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 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 ***toucning 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

In the event that 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; 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; 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”***<sup>5</sup> 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***<sup>6</sup>. Consult the Rexrainbow’s documentation about his ***Weapon Plugin here***<sup>7</sup> 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***<sup>8</sup> 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 save

---

<sup>5</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/>

<sup>6</sup><http://labs.phaser.io/edit.html?src=src/pools/bullets.js>

<sup>7</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/>

<sup>8</sup><http://labs.phaser.io/index.html?dir=pools/&q=>

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 which 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](#):**<sup>9</sup>

- **Average Focus**<sup>10</sup> — click while running the source code and observe how the avatar spins to follow the target.
- **Combat mechanics**<sup>11</sup> — 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**<sup>12</sup> — 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**<sup>13</sup> — 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 focus on the target instead of the avatar.

#### **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');

```

<sup>9</sup><http://labs.phaser.io/index.html?dir=games/topdownShooter/&q=>

<sup>10</sup>[http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown\\_averageFocus.js](http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_averageFocus.js)

<sup>11</sup>[http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown\\_combatMechanics.js](http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_combatMechanics.js)

<sup>12</sup>[http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown\\_playerFocus.js](http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_playerFocus.js)

<sup>13</sup>[http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown\\_targetFocus.js](http://labs.phaser.io/edit.html?src=src/games/topdownShooter/topdown_targetFocus.js)

```

11     bullet.angle = this player.angle;
12     bullet.body.setBelocity(bulletPoints.tx*100,bulletPoints.ty*100)
13     this.bulletGroup.add(bullet);
14 }
15
16 function getDirFromAngle(angle){
17     // Description: converts degrees to radians
18     // ????? var rads = angle*Math.PI/180; ??????
19     // Use 3.142 and stop wasting battery power!
20     // var rads = angle * 3.142/180;    // OR better still!!
21     var rads = angle * 0.01745;        // and just pre-calculate it!
22     var tx = Math.cos(rads);          // see chapter footnotes
23     var ty = Math.sin(rads);          // see chapter footnotes
24     return {tx,ty}
25 }
26
27 this.physics.add.collider(this.bulletGroup,
28                         <targetGroup here!>,
29                         <what happens?>, null, this);

```

---



To preserve CPU processing and battery, pre-calculate math formula. Refer to [sine and cosine here](#)<sup>14</sup>. **One radian equals**<sup>15</sup>  $180^\circ / \pi = 57.30^\circ$ . Use this [online calculator](#)<sup>16</sup> to help reduce CPU work load at runtime and thereby save battery power. Refer to [this article](#)<sup>17</sup> for a gentle introduction (or reminder) on degrees, radians, and angles.

One of the common develop mistakes 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.

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

<sup>14</sup><http://www2.clarku.edu/faculty/djoyce/trig/cosines.html>

<sup>15</sup><https://ee.stanford.edu/~hellman/playground/hyperspheres/radians.html>

<sup>16</sup>[https://www.rapidtables.com/calc/math/Cos\\_Calculator.html](https://www.rapidtables.com/calc/math/Cos_Calculator.html)

<sup>17</sup><https://betterexplained.com/articles/intuitive-guide-to-angles-degrees-and-radians/>

mind that each bullet is a game object that requires CPU processing, memory, and rendering onto the display. To preserve on these, I chose to simply have a "FIRE" buttons to abstract missile combat.

Here is a Phaser III **Bullet plugin documentation here**<sup>18</sup> and **source code at GitHub**<sup>19</sup>

**And other sample tutorials with (FREE!) source code:**

1. **Zenna Game Academy: "How to make a Tower Defense Game"**<sup>20</sup>
2. **William Clarkson: "Phaser 3 Physics for beginners - Endless Bullets"**<sup>21</sup>
3. **Phaser3 Labs — Defenda!**<sup>22</sup>



**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"<sup>23</sup> 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**.<sup>a</sup> "Juice is typically auditory or visual, but it doesn't really need to be ... it's about maximum output for minimum input."

Here are some ways to enhance missile combat:

- Turn it side-ways and create a vertically scrolling shooter instead.
- Give the missile's acceleration; instead of 'Velocity' then watch them increase in speed over time.

<sup>18</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/bullet/>

<sup>19</sup><https://github.com/rexrainbow/phaser3-rex-notes/blob/master/examples/bullet/bullet.js>

<sup>20</sup><https://gamedevacademy.org/how-to-make-tower-defense-game-with-phaser-3/?a=47&campaign=Phaser3GamePrototyping>

<sup>21</sup><https://phasergames.com/phaser-3-physics-beginners/>

<sup>22</sup><http://labs.phaser.io/edit.html?src=src/games/defenda/test.js>

<sup>23</sup><http://www.gameanalytics.com/blog/squeezing-more-juice-out-of-your-game-design.html>

\* Give the missiles a “waypoint” in order to “home in” on targets.

<sup>a</sup><https://www.pocketgamer.biz/news/64630/gdc-europe-is-no-more/>

## 7.4 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”**;<sup>24</sup> 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 through a **story narrative of the combat**.

```
this.physics.add.collider(player, monster, bumpMonster, null, this);
```

<sup>24</sup>[makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/lesson11a.html](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/lesson11a.html)

**Example 7.5: Grid-less Combat Movement Lines 81 - 292**


---

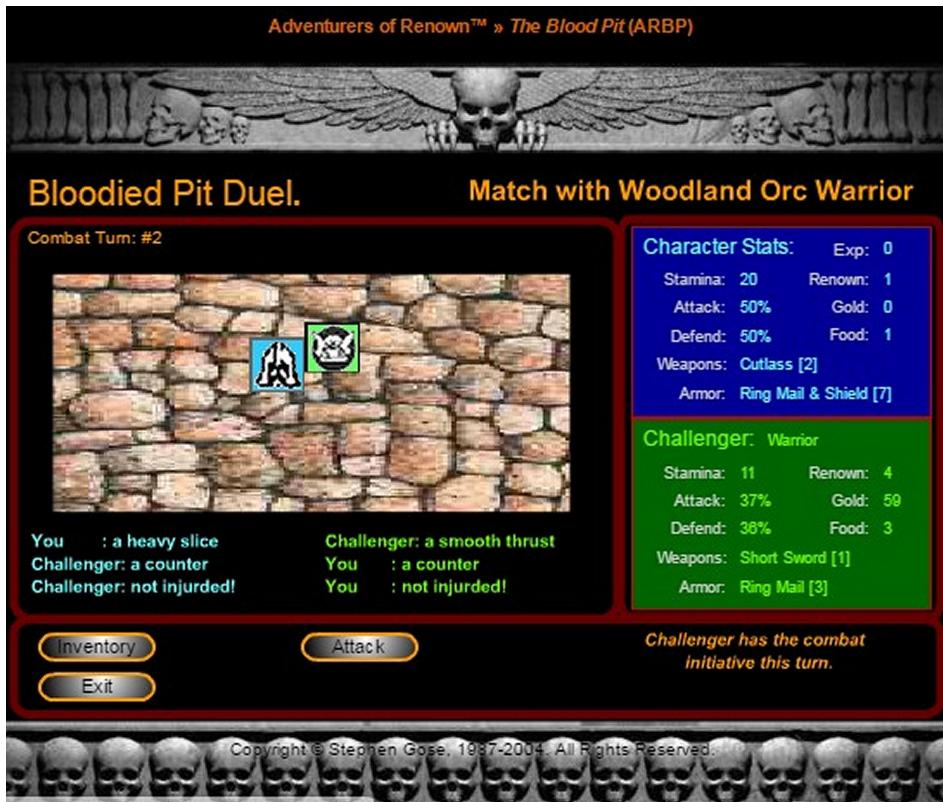
```

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 quotes
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:** [http://makingbrowsergames.com/p3gp-book/\\_p3-bloodPitv1](http://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 game-boards add a new dimension to our conflicts, that is namely: **maneuvers**. We will create this game-board using several methods such as "**tiled-maps**" OR the new "grid" features in Phaser III.

## 7.5 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 visual display

with its associated metadata in the game environment. I take the same approach with “Tactical Tiled-Maps” as I do with 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<sup>25</sup> 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 continue to use rectangles and hexagons.



**Warning:** Software such as **Texture Packer**,<sup>26</sup> **Adobe Animate** (formerly called Flash CC)<sup>27</sup> or **Shoebox**<sup>28</sup> all generate texture atlases, **not sprite sheets**. You must use `Loader.atlas` instead. You can download a **FREE** Sprite Sheet Packer here.<sup>29</sup>

## 7.6 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.
- 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?

---

<sup>25</sup><http://www.dictionary.com/browse/teetering>

<sup>26</sup><https://www.codeandweb.com/texturepacker>

<sup>27</sup><https://www.adobe.com/products/animate.html>

<sup>28</sup><http://renderhjs.net/shoebox/>

<sup>29</sup><https://www.codeandweb.com/free-sprite-sheet-packer>



Our Avatar wins in Combat & Salvages rewards

### 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*<sup>30</sup> 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 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 row and columns (i.e.: locations on the grid) where the avatar currently is (i.e., metadata).
2. **Or**, we could place “invisible” walls around the grid to prevent the avatar 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**<sup>31</sup>

---

<sup>30</sup><https://www.merriam-webster.com/dictionary/itty-bitty>

<sup>31</sup><http://makingbrowsergames.com/starterkits/jump2cap/Peg-Examples/trixAttacksMagix-Phaser/index-mobile-wctam.html#game>

### Sample: Grid-ed Movement

---

```

1 //for example, snapping a grid coordinate of 43 to the nearest
2 //multiple of of the grid size:
3
4 var dx = 43;           //distance of x from sprite or point.
5 var gridSize = 32;    //assuming every grid square is 32px
6 var columnX = Math.round((dx / gridSize) * gridSize);
7
8 //This provides a snapped sprite column as the first column grid.

```

---



**Exercise:** Download and study [this source code](#)<sup>32</sup>

### References from Mozilla Developers:

- **Square Grid Tile Maps samples:**<sup>a</sup> A collection of resources used by Mozilla developers for developers, technical evangelizing and similar such content.
- **JavaScript for game development**<sup>b</sup> A compilation of materials to learn JavaScript and make HTML5 games.
- **JS Game development examples for Tilemaps**<sup>c</sup> Examples of tile maps implementation with the Canvas API.
- **HTML5 games workshop**<sup>d</sup> 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 guiding for people to finish it at home if only a short session with a coach is possible.

---

<sup>a</sup><https://github.com/mozdevs/gamedev-js-tiles>

<sup>b</sup><https://github.com/mozdevs/js-for-gamedev>

<sup>c</sup><https://github.com/mozdevs/gamedev-js-tiles>

<sup>d</sup><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\*\*](#).

---

<sup>32</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/group/#create-game-objects>

- ***labs.phaser.io demo***<sup>33</sup>
- ***rexrainbow UI plugin***<sup>34</sup>

### ***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<sup>a</sup>

<sup>a</sup><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***,<sup>35</sup> and being such, you could add grids to any Scene, either inside ***Group(s)***<sup>36</sup> or ***Container(s)***.<sup>37</sup> 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***.<sup>38</sup> The “Grid” gives you 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)***<sup>39</sup>

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.

<sup>33</sup><http://labs.phaser.io/view.html?src=src/game%20objects/shapes/grid.js>

<sup>34</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/ui-gridsizer/>

<sup>35</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObject.html>

<sup>36</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Group.html>

<sup>37</sup><https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Container.html>

<sup>38</sup><http://labs.phaser.io/view.html?src=src/geom/rectangle/set%20empty.js>

<sup>39</sup>[https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObjectFactory.html#grid\\_anchor](https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.GameObjectFactory.html#grid_anchor)

```
//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**.<sup>40</sup>



**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**.<sup>41</sup>

## Hexagonal Grids

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 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<sup>42</sup> has collected some articles that may help you turn common square-grid algorithms into hex-grid algorithms.<sup>43</sup> 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,

<sup>40</sup><https://github.com/rexrainbow/phaser3-rex-notes/blob/master/examples/gridtable/gridtable.js>

<sup>41</sup><https://phasergames.com/scaling-games-in-phaser-3/>

<sup>42</sup><http://www-cs-students.stanford.edu/~amitp/gameprog.html#hex>

<sup>43</sup><http://www-cs-students.stanford.edu/~amitp/game-programming/grids/>

largely based on the guides by Charles Fu<sup>a</sup> and Clark Verbrugge.<sup>b</sup> 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 will update diagrams, code, and text to match.

<sup>a</sup><http://www-cs-students.stanford.edu/~amitp/Articles/Hexagon2.html>

<sup>b</sup><http://www-cs-students.stanford.edu/~amitp/Articles/HexLOS.html>



Creating a Hexagonal Mega-Squares in a map grid

### Example: Dynamically Created Hexagonal Grid Combat - *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 “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**.<sup>44</sup> in either North-South or East-West orientations.

```
var tileXYArray = scene.rexBoard.hexagonMap.hexagon(board, radius);
// var out = scene.rexBoard.hexagonMap.hexagon(board, radius, out);
```

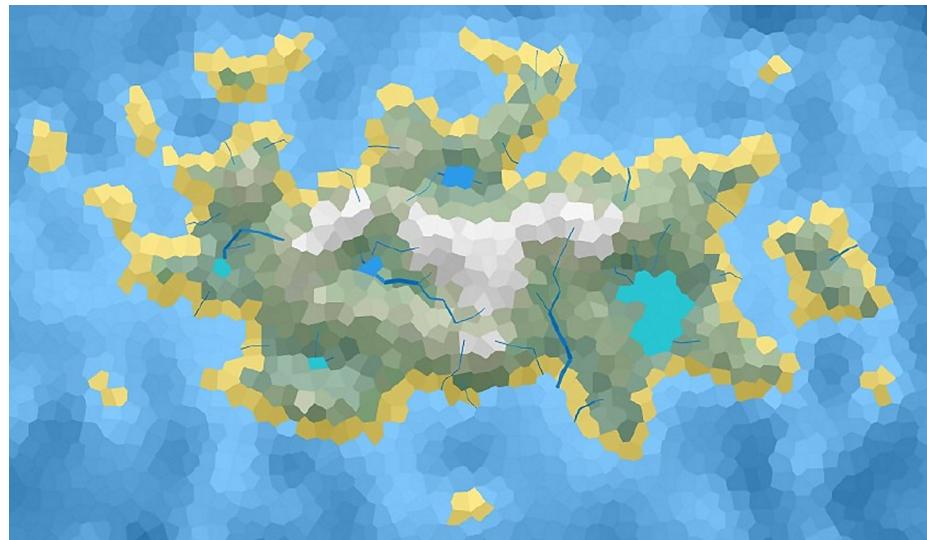
## Squishes

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”<sup>45</sup> at some point to come. If you would like to see a Phaser

<sup>44</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/board-hexagonmap/#hexagon>

<sup>45</sup><http://idioms.thefreedictionary.com/eat+words>

Plugin for squishes download from here.<sup>46</sup> Here's an illustration of squishes from the Phaser Squish Generator:



**Phaser Squish Plugin Map generator**

## 7.7 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 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" (シミュレーションRPG), abbreviated as SRPG).



**Note:** *Ruins of Able-Wyvern™ (ARRA)* has included both tactical combat and strategic roleplay since 1993.

**Learning the ropes**<sup>47</sup> 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

<sup>46</sup><http://luckylooke.github.io/phaser-islandjs-plugin/>

<sup>47</sup><http://idioms.thefreedictionary.com/learn+the+ropes>

understand why I selected these forms of combat resolution.



**Exercise:** 1. Research other combinations on **conducting combat here**.<sup>48</sup>

**Exercise:** 2. Research Controversy and Criticisms of preferred **conflict resolution across cultures here**.<sup>49</sup>

**Exercise:** 3. Explore **what has happened with RPG and what gamers currently expect**.<sup>50</sup>

## 7.8 “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**:<sup>51</sup>

- 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:**

- **Click fest!** Originally deployed in **Ruins of Able-Wyvern™ (ARRA)** and **Blood Pit™ (version 1)**.<sup>52</sup> 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 to **Chapter 5 GM Skills**. 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**,<sup>53</sup> acting out the attacks as in **“Street Fighter”**.<sup>54</sup> This combat system is heavily dependent on artwork animation.
- The **Society for Creative Anachronism**<sup>55</sup> **virtual trainer, game design by Steve Echols**
- **“En Guard!”**<sup>56</sup> a rival to D&D and still wildly popular today.

<sup>48</sup>[https://en.wikipedia.org/wiki/Role-playing\\_video\\_game#Combat](https://en.wikipedia.org/wiki/Role-playing_video_game#Combat)

<sup>49</sup>[https://en.wikipedia.org/wiki/Role-playing\\_video\\_game#Cultural\\_differences](https://en.wikipedia.org/wiki/Role-playing_video_game#Cultural_differences)

<sup>50</sup>[https://en.wikipedia.org/wiki/Role-playing\\_video\\_game](https://en.wikipedia.org/wiki/Role-playing_video_game)

<sup>51</sup>[http://www.roguebasin.com/index.php?title=Thoughts\\_on\\_Combat\\_Models](http://www.roguebasin.com/index.php?title=Thoughts_on_Combat_Models)

<sup>52</sup>[http://localhost/\\_GIS/GISUS-MakingBrowserGames/makingbrowsergames.com/p3gp-book/\\_p3-bloodPitv1/](http://localhost/_GIS/GISUS-MakingBrowserGames/makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/)

<sup>53</sup><https://www.battleon.com/aq-play.asp>

<sup>54</sup><http://gamequeryjs.com/>

<sup>55</sup><http://www.sca.org/>

<sup>56</sup>[https://en.wikipedia.org/wiki/En\\_Garde!](https://en.wikipedia.org/wiki/En_Garde!)

- **Yeap! Ya betcha' ur life!** — 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 lets the combatants 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).

### 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 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.***<sup>57</sup> 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 the unique narrative action that occurred during this single combat round. It generates the random events, compares combat skills with a helper function. Evaluates the

<sup>57</sup><https://medicalxpress.com/news/2017-01-attention-curtain-human-brain-important.html>

success or failure of each antagonists' attack and defense, and lastly calculates physical damage imposed upon 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.<sup>58</sup>

## Guitar hero - Time to get it Right!

- Red Fountain Swordsman (play the original flash game here)<sup>59</sup>: 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.**<sup>60</sup>



**Note:** There's a similar game to Guitar-Hero constructed in Phaser. I encourage you to download and review their GitHub **OOP** source code.<sup>61</sup>

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"<sup>62</sup> 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<sup>63</sup>, from which came my idea for this version of combat — the **Red Fountain Swordsman** game.

<sup>58</sup>[http://makingbrowergames.com/p3gp-book/\\_p3-bloodPitv1/js/state/combat.js](http://makingbrowergames.com/p3gp-book/_p3-bloodPitv1/js/state/combat.js)

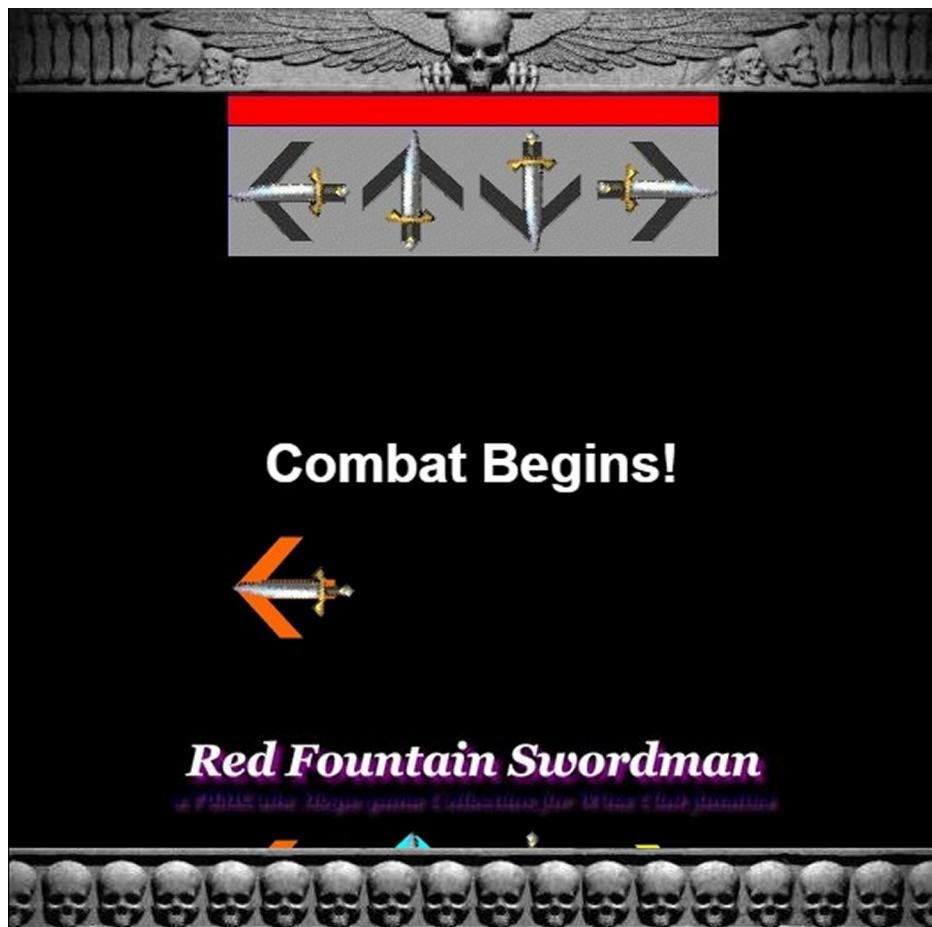
<sup>59</sup><http://www.renown-games.com/winx/red-fountain-swordman/index.html>

<sup>60</sup>[http://makingbrowergames.com/book/\\_rfs-Phaser/](http://makingbrowergames.com/book/_rfs-Phaser/)

<sup>61</sup><https://github.com/PBMCube/banjo-hero>

<sup>62</sup>[http://www.roguebasin.com/index.php?title=Time\\_Systems](http://www.roguebasin.com/index.php?title=Time_Systems)

<sup>63</sup><http://guitarflash.com/>



Phaser version of Red Fountain Swordsman



**Hint:** This complete source is available only in the Bonus Download content. Developer's Demo located at [http://makingbrowergames.com/book/\\_rfs-Phaser/](http://makingbrowergames.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"<sup>64</sup>. I had many visits and exchanges with them in their earlier years at the

<sup>64</sup><http://www.battleon.com/>

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”<sup>65</sup> of today’s RPG combat as seen in this online course below which mimics their combat style.



Sample of Single Page Combat from Zenva

- ***Single Page Combat from Zenva RPG Online Course***<sup>66</sup> FREE Source Code available in this excellent tutorial.
- ***Advanced Phaser 3 – Build an RPG***<sup>67</sup> 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.

<sup>65</sup><http://dictionary.cambridge.org/us/dictionary/english/bread-and-butter>

<sup>66</sup><https://academy.zenva.com/course/rpg-game-development-with-phaser/?a=47&campaign=Phaser3GamePrototyping>

<sup>67</sup><https://academy.zenva.com/product/advanced-phaser-3-build-an-rpg/?a=47&campaign=Phaser3GamePrototyping>

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.



**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 pseudo code and flowchart.**<sup>68</sup> I encourage you to combine this "SCA Virtual Trainer" with this interesting game called **Color ZAP (book)**<sup>69</sup> for a single-player combat system.

### 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 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+)**<sup>70</sup>

<sup>68</sup>[http://makingbrowergames.com/p3gp-book/p3-sca/code2flow\\_rps.pdf](http://makingbrowergames.com/p3gp-book/p3-sca/code2flow_rps.pdf)

<sup>69</sup><http://amzn.to/2njpDQn>

<sup>70</sup><http://makingbrowergames.com/p3gp-book/p3-sca/>

- **Demonstration multi-player game (v3.16+)**<sup>71</sup>

## 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 System (explained in detail here)**<sup>72</sup> — 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 moves 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.**<sup>73</sup>

<sup>71</sup>[https://leanpub.com/rrgamingistem](https://leanpub.com/rrgamingystem)

<sup>72</sup>[http://www.roguebasin.com/index.php?title=Time\\_Systems](http://www.roguebasin.com/index.php?title=Time_Systems)

<sup>73</sup><https://amzn.to/2Ncimw6>

**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 pull could spin up the attacks and defenses for both attacker and antagonist. 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.<sup>74</sup>

## 7.9 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.

---

<sup>74</sup><http://www.emanueleferonato.com/2010/04/13/17-jquery-powered-web-games-with-source-code/>

The screenshot shows a web page with a dark red header bar. In the top left corner, there is a navigation menu with options like 'Home', 'Step 1 of 3', and 'Your Quest's background story.' Below this, there are two main sections: 'Fantasy' and 'Sci-Fi'. The 'Fantasy' section is currently selected, indicated by an orange border around its title. It contains four style options: 'D&D Style' (selected), 'Generic Style', 'Simple Outline', and 'Warhammer Style'. The 'Sci-Fi' section also contains four style options: 'Generic Future' (selected), 'Simple Outline', and 'Warhammer 40K'. To the right of these sections, there is a large text area containing two paragraphs of generated text. The first paragraph is about a fantasy quest where the player meets a familiar face for a meal, and a noble waits for news. The second paragraph is about a sci-fi scenario involving major thefts from rich noblemen and their family tombs.

Here is Your Fantasy Quest ... –  
As you wander through the streets of a large city, you soon meet a familiar face, who treats you to a meal in exchange for willing ears. A noble of high esteem waits until most of the crowds have drawn away from you before telling you any news beyond small talk. Finally, the tale begins.

Recently, there has been rash of major thefts from the rich and powerful. This obviously has many locals concerned. Rumors say it is the work of the corrupt commander of the local army, but none know the truth. It's up to you to get to the bottom of the controversy and learn what it means for all the folk and lands around the family tombs of rich noblemen. In exchange for your aid, you'll gain the reward of a minor magical item as well as additional accolades as heroes of the family tombs of rich noblemen.

Copyright © 1978-2003 Stephen Gose. All Rights Reserved.  
Questions or comments? Please email support@pbmcube.com

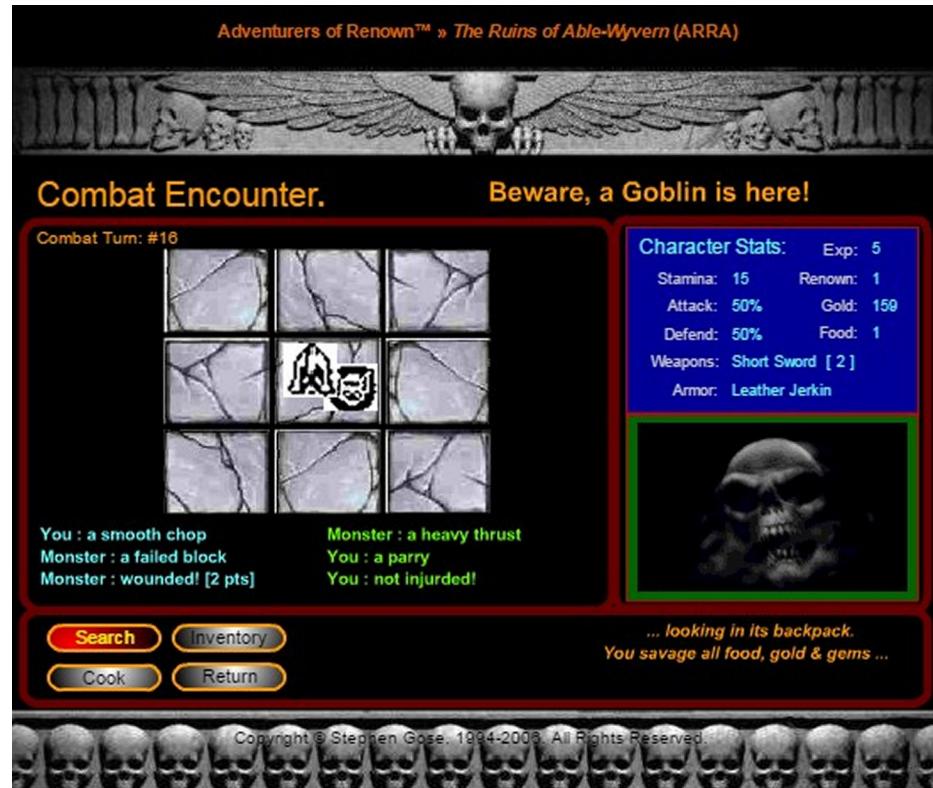
**Randomly Generate background stories**



**Hint:** This complete source is available in online *Source code Appendix*.<sup>75</sup>

<sup>75</sup><http://makingbrowsergames.com/book/index13.html#13.3>

## 7.10 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.

## 7.11 Chapter Source Code

<http://makingbrowsergames.com/book/index.html>

**Book Combat demo (online)**<sup>76</sup>



**Hint:** Use Developer's Console to study the internal game operations.

<sup>76</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3-bloodPitv1/](http://makingbrowsergames.com/p3gp-book/_p3-bloodPitv1/)

## 7.12 Complete Combat Prototypes

We have moved the Source Code Appendix onto a website and removed it from edition 1 book release. This allows us to update code changes dynamically for Phaser v3 as it nears completion.

- ***Combat Systems Game Prototype Library***<sup>77</sup>
- ***Chapter 7 online Examples***<sup>78</sup>

## 7.13 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 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 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 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.

<sup>77</sup><http://makingbrowsergames.com/p3gp-book/index-combat.html>

<sup>78</sup>[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch7-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/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.

## 7.14 Footnotes

- To preserving CPU processing and battery, pre-calculate math formula. Refer to **sine and cosine here**<sup>79</sup>. **One radian equals**<sup>80</sup>  $180^\circ / \pi = 57.30^\circ$ . Use this **online calculator**<sup>81</sup> to help reduce CPU work load.
- T, A. (2017, January 17). Finger-Friendly Design: Ideal Mobile Touchscreen Target Sizes - Smashing Magazine.<sup>82</sup> Retrieved May 19, 2017,
- EnGard game system. Product info<sup>83</sup>
- Color ZAP (book)<sup>84</sup>
- Phaser Plugin for squishes.<sup>85</sup>
- Hex grid resources<sup>86</sup>
- References from Mozilla Developers:<sup>87</sup>
- Square Grid Tile Maps samples:<sup>88</sup>.
- “The Four Virtues of a good tactical turn-based combat system”<sup>89</sup>
- JavaScript for game development<sup>90</sup>
- JS Game development examples for Tilemaps<sup>91</sup>.
- HTML5 games workshop<sup>92</sup>
- How to write a Rogue-like game in 15 Steps<sup>93</sup>

<sup>79</sup><http://www2.clarku.edu/faculty/djoyce/trig/cosines.html>

<sup>80</sup><https://ee.stanford.edu/~hellman/playground/hyperspheres/radians.html>

<sup>81</sup>[https://www.rapidtables.com/calc/math/Cos\\_Calculator.html](https://www.rapidtables.com/calc/math/Cos_Calculator.html)

<sup>82</sup><https://www.smashingmagazine.com/2012/02/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/>

<sup>83</sup><http://www.engarde.co.uk/useful.html#Top>

<sup>84</sup><http://amzn.to/2njpDQn>

<sup>85</sup><http://luckylooke.github.io/phaser-islandjs-plugin/>

<sup>86</sup><http://www-cs-students.stanford.edu/~amitp/gameprog.html#hex>

<sup>87</sup><https://github.com/mozdevs>

<sup>88</sup><https://github.com/mozdevs/gamedev-js-tiles>

<sup>89</sup><http://sinisterdesign.net/12-ways-to-improve-turn-based-rpg-combat-systems/>

<sup>90</sup><https://github.com/mozdevs/js-for-gamedev>

<sup>91</sup><https://github.com/mozdevs/gamedev-js-tiles>

<sup>92</sup><https://github.com/mozdevs/html5-games-workshop>

<sup>93</sup>[http://www.roguebasin.com/index.php?title=How\\_to\\_Write\\_a\\_Roguelike\\_in\\_15\\_Steps](http://www.roguebasin.com/index.php?title=How_to_Write_a_Roguelike_in_15_Steps)

- Rogue Basin articles on combat<sup>94</sup>

---

<sup>94</sup>[http://www.roguebasin.com/index.php?title=Articles#Combat\\_2](http://www.roguebasin.com/index.php?title=Articles#Combat_2)

## 8. Whazzz-sUP! .... HUD Development

### ***HUD Topics:***

- ***Review Phaser III 68 Plugins***<sup>1</sup>
- Character Inventory & Development Scene (ARRAv15)
- Forum discussions ***here***<sup>2</sup>
- 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.***<sup>3</sup>
- ***Create a Game UI with the HTML5 CANVAS***<sup>4</sup>

The “heads-up display” is a critical part of your game flow. 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 game and other entertainment services? The HUD should encourage, entice, taunt and ***titillize***<sup>5</sup>, in “Phaser Render phase”, for more of what’s to come from your game!

---

<sup>1</sup><https://rexrainbow.github.io/phaser3-rex-notes/docs/site/#list-of-my-plugins>

<sup>2</sup><http://www.html5gamedevs.com/topic/15822-building-in-game-ui/>

<sup>3</sup><https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47>

<sup>4</sup><https://gamedevacademy.org/create-a-game-ui-with-the-html5-canvas/?a=47>

<sup>5</sup><http://www.urbandictionary.com/define.php?term=titilizing>



Movement HUD as attached to Avatar

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<sup>a</sup>

<sup>a</sup><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 tile map inside the canvas.

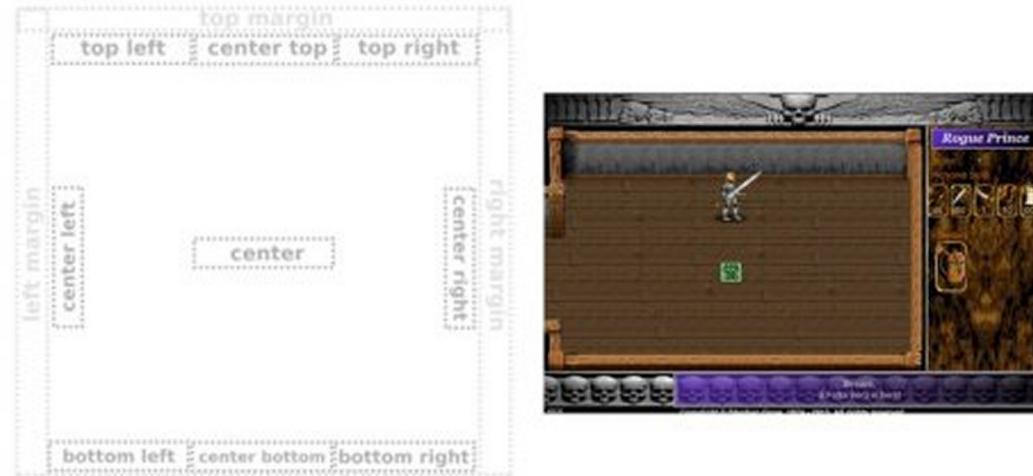
## 8.1 HUD Housing Development

You can have multiple Phaser III Scenes all running in parallel. This is similar in my mind as Flash MovieClip running on the 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 needed for an avatar actions, and current game state. Here are some suggestions from this excellent ***Game Dev Academy tutorial***<sup>6</sup> showing the border layout positioning style.

<sup>6</sup><https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47>



**Exercise:** Learn more about the border layout style<sup>7</sup> from Oracle and Java implementation.



**HUD information prior to combat**

The HUD is composed of various items such as text, images, sprites, animations. These should be contained within the HUD display group.

#### 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.
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);
```

<sup>7</sup>[http://www.java2s.com/Tutorial/Java/0240\\_Swing/WhatistheBorderLayout.htm](http://www.java2s.com/Tutorial/Java/0240_Swing/WhatistheBorderLayout.htm)

```

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);

```



HUD information The Rogue Prince Main Menu

## 8.2 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 other 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 as a sliding panel onto and from the game area using Phaser. Groups are also displayed objects; this means that groups could nest, as 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 “HUD Panel Group” with the buttons, graphics and text information as a single group package.



**Clicking Menu button reveals HUD Panel**

## 8.3 HUD Panels outside the Canvas?!?

Phaser helps display an `html5 canvas` element. Why should we limit ourselves only to HUDs only inside canvas elements? If you have studied mobile web design, as suggested by **Josh Morony**,<sup>8</sup> it becomes an innovation to have HUD Panels controlled by the Browser, jQuery, and CSS. It becomes a simple matter to have a in-sliding HUD information panel on one or both sides of the canvas game.



**Exercise:** Review examples at [w3Schools.com](http://w3Schools.com):<sup>9</sup>

- **Open Panel**<sup>10</sup>
- **Overlay, Reveal and Push Panels**<sup>11</sup>
- **Right-side**<sup>12</sup>
- **Create a Dialog Modal Plugin in Phaser 3**<sup>13</sup>
- <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**.<sup>14</sup>

<sup>8</sup><https://www.joshmorony.com/mobile-development-for-web-developers/getting-started-with-phaser.html>

<sup>9</sup>[https://www.w3schools.com/jquerymobile/jquerymobile\\_panels.asp](https://www.w3schools.com/jquerymobile/jquerymobile_panels.asp)

<sup>10</sup>[https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob\\_panels\\_basic](https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_basic)

<sup>11</sup>[https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob\\_panels\\_display](https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_display)

<sup>12</sup>[https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob\\_panels\\_rightpos](https://www.w3schools.com/jquerymobile/tryit.asp?filename=tryjqmob_panels_rightpos)

<sup>13</sup><https://gamedevacademy.org/create-a-dialog-modal-plugin-in-phaser-3-part-1/?a=47&campaign=Phaser3GamePrototyping>

<sup>14</sup><https://www.merixstudio.com/skytte/>



## 8.4 HUD Demos



### Example 8.3: HUD Menu Grouping

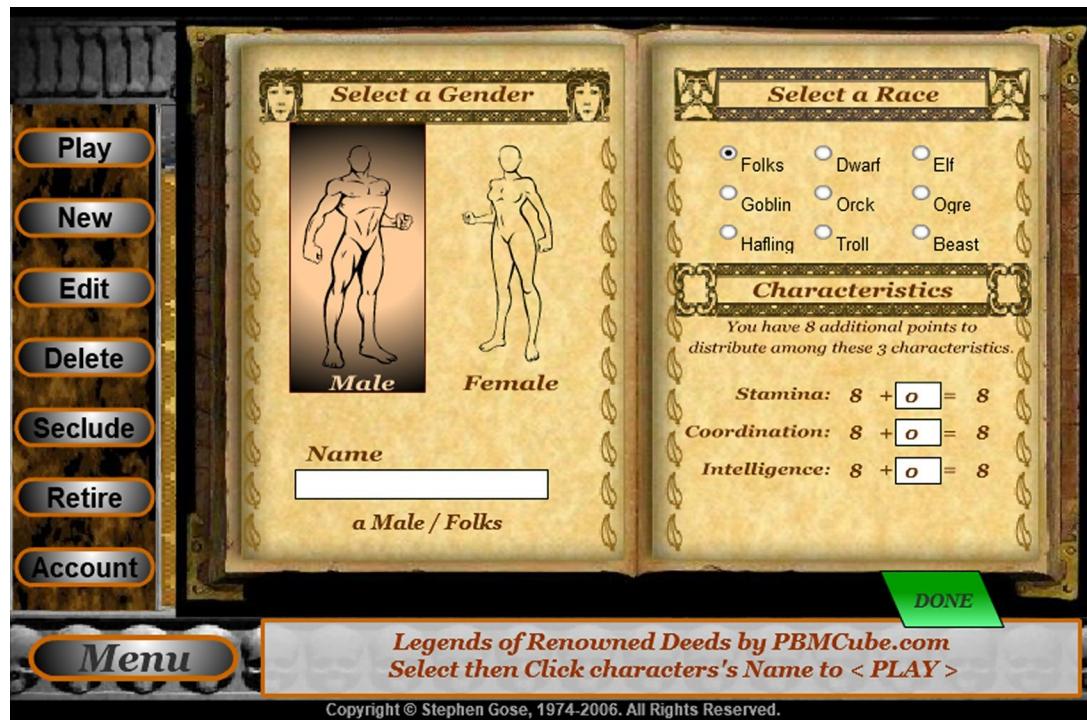
```

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 a way for user to communicate into the game and modify abstract data structures and properties.



Main Menu HUD for *Rogue Prince™ (ARRP)*



War Lord HUD details in War Lord Tourney™ (RRWT) MMoG



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>

## 8.5 Summary

### ***Step-by-step guide: Create a Game HUD Plugin in Phaser<sup>15</sup>***

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.

## 8.6 Footnotes

- Phaser.io Forum<sup>16</sup>
- HUD Manager via plugins.<sup>17</sup>
- Game Dev Academy tutorial<sup>18</sup>

---

<sup>15</sup><https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47&campaign=Phaser3GamePrototyping>

<sup>16</sup><http://www.html5gamedevs.com/topic/1924-hud-how-to-implement-it/>

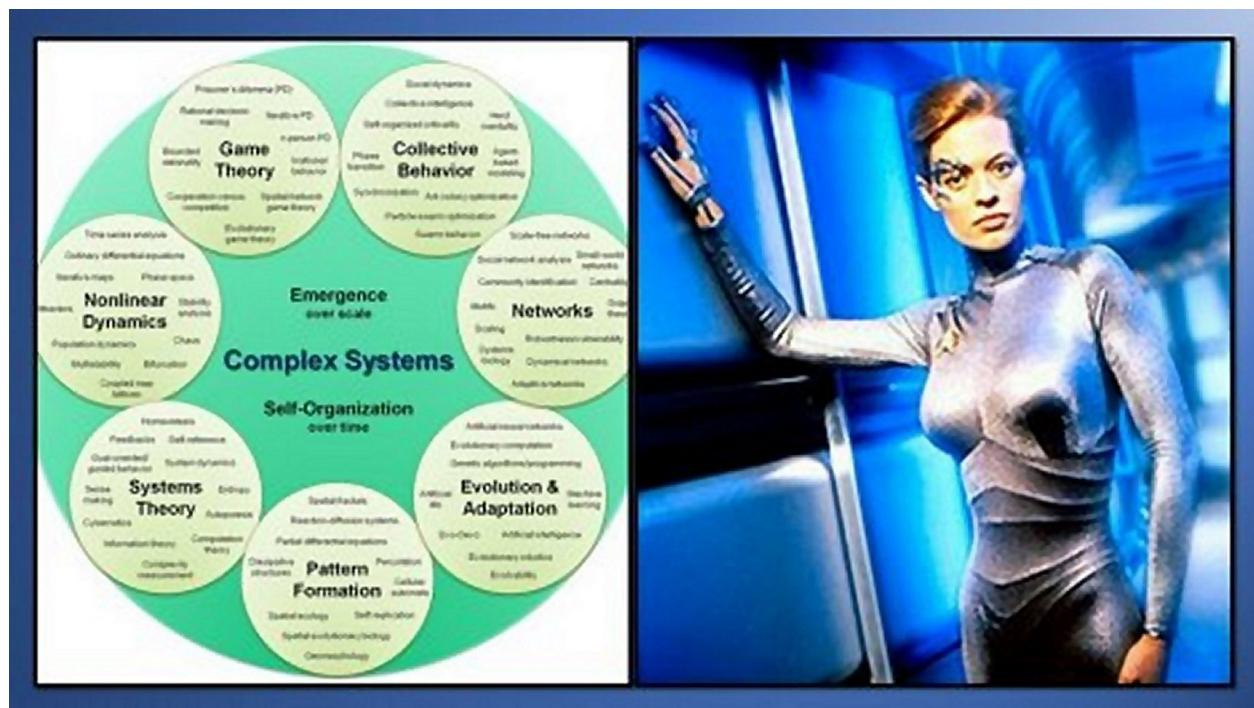
<sup>17</sup><http://phaser.io/docs/2.6.2/Phaser.PluginManager.html>

<sup>18</sup><https://gamedevacademy.org/how-to-create-a-game-hud-plugin-in-phaser/?a=47&campaign=Phaser3GamePrototyping>

## 9. Don't make me think or “Artificial Intelligence for Dummies”

So far in our game prototype, “monsters” have just stood there and took 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)<sup>1</sup>, a paraphrased quote (with generous liberties) from the movie ***“Network”***<sup>2</sup>.

### 9.1 The “6 of 9”



*Not to be confused with 7 of 9*

<sup>1</sup><https://www.youtube.com/watch?v=rGIY5Vj4YM>

<sup>2</sup><http://www.tcm.com/tmdb/title/342/Network/>

These are only 6 artificial intelligence (AI) routines<sup>3</sup> we'll review of the nine (9) in Gaming Theory.

- Chasing: the relentless hunting of a player's avatar.
- 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)***.<sup>4</sup>

## 9.2 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;}
```

---

## 9.3 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.

---

<sup>3</sup>[https://en.wikipedia.org/wiki/Artificial\\_intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence)

<sup>4</sup>[https://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM\\_Tutorial.pdf](https://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf)

### Example 9.2: Combat Pseudo Code

---

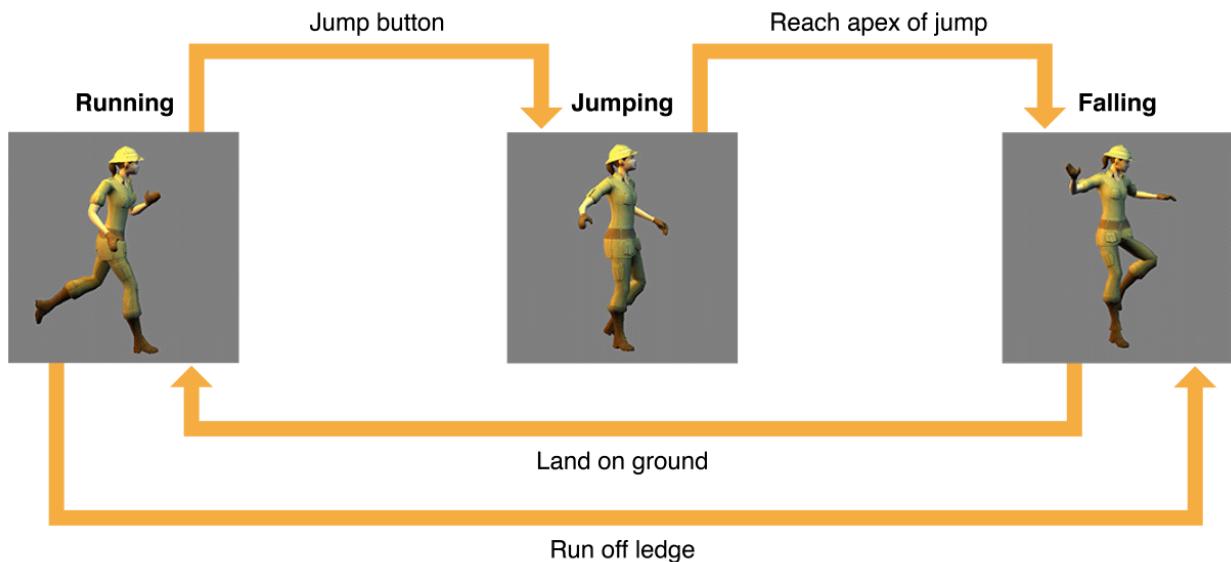
```
//Let ex, ey be the enemy x- and y- grid positions.
//Let px, py be the avatar's x- and y- grid positions.
//During the updates, enemy runs from avatar.

if (ex > px){ex += 1;}
if (ey > py){ey += 1;}
if (ex < px){ex -= 1;}
if (ey > py){ey -= 1;}
```

---

## 9.4 Patterns

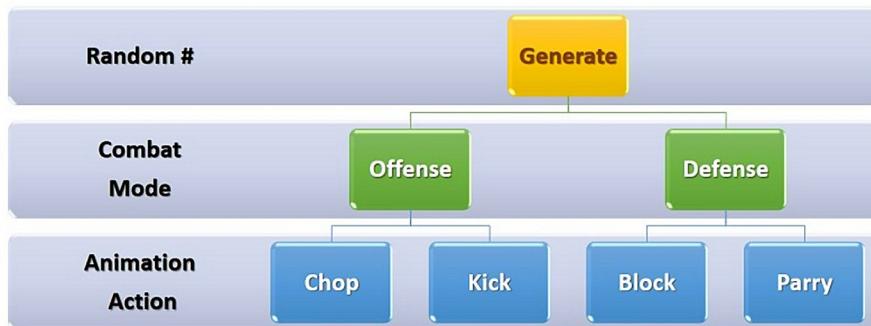
There are dozens of Phaser tutorials about artificial intelligence patterns. And if you squint really hard<sup>5</sup>, you might see that these AI patterns are similar to "Key Frame Animation" and their associated atlas.<sup>6</sup>



For a combat example, consider this pattern:

<sup>5</sup><https://en.wikipedia.org/wiki/Squint>

<sup>6</sup><https://www.joshmorony.com/how-to-create-animations-in-phaser-with-a-texture-atlas/>



**Random numbers to select a patterned response**

- “Patrolling” monsters<sup>7</sup> is a simple AI pattern discussed in this tutorial. You might want to compare another method to implement patrolling here.<sup>8</sup>
- “Flocking” is another AI pattern discussed here with Phaser examples.<sup>9</sup>
- “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<sup>10</sup> to your liking. Their Phaser Plugin<sup>11</sup> is open source.
- GameDevAcademy has my favorite tutorial on Path finding here.<sup>12</sup>
- My favorite author wrote an interesting article<sup>13</sup> on the use of “antenna feelers” for a top-down racing game.
- Here’s my low IQ routine<sup>14</sup> used in Ruins of Able-Wyvern™.

<sup>7</sup><https://phaser.io/news/2016/04/patrolling-enemy-ai>

<sup>8</sup><http://www.emanueleferonato.com/2015/03/05/create-an-html5-game-like-drop-wizard-with-phaser-patrolling-enemies/>

<sup>9</sup><https://processing.org/examples/flocking.html>

<sup>10</sup><https://phaser.io/news/2016/02/how-to-use-pathfinding-in-phaser>

<sup>11</sup>[https://github.com/appsbu-de/phaser\\_plugin\\_pathfinding](https://github.com/appsbu-de/phaser_plugin_pathfinding)

<sup>12</sup><https://gamedevacademy.org/how-to-use-pathfinding-in-phaser/?a=47>

<sup>13</sup><http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/>

<sup>14</sup><https://www.verywell.com/what-is-considered-a-low-iq-2795282>

### 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;
    this.enemy.body.velocity.x += speed;
}
//Applying fuzzy logic ...
```

---

### Play AI Combat demo<sup>15</sup>



**Exercise:** Find the “Street Fighter” game<sup>16</sup> and determine how your opponent selects its tactics in a single player game. Does it use a pattern of animation?

## 9.5 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

---

<sup>15</sup>[http://makingbrowergames.com/p3gp-book/\\_p3demos/ch4-examples/index.html](http://makingbrowergames.com/p3gp-book/_p3demos/ch4-examples/index.html)

<sup>16</sup>[https://en.wikipedia.org/wiki/Street\\_Fighter](https://en.wikipedia.org/wiki/Street_Fighter)

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.

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 stand 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 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.

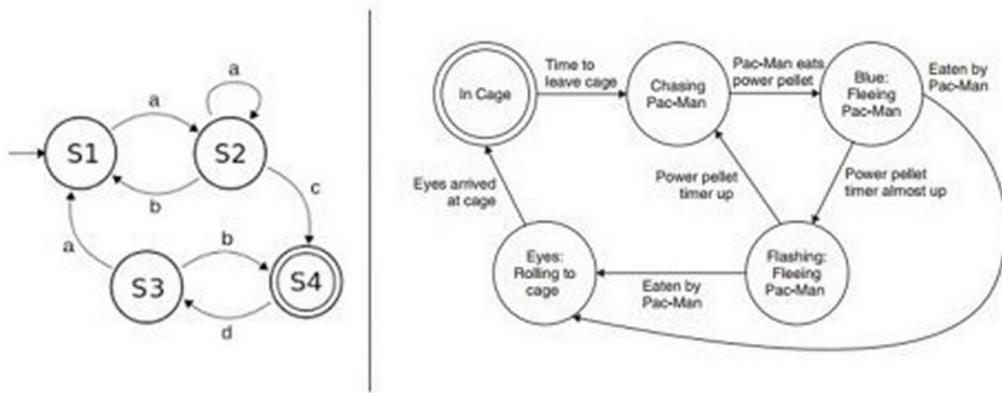
## 9.6 Finite State Machines (FSM)

The true power of AI is reached when **finite state machines (FSM)**<sup>17</sup> 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™.

A **finite state machine**<sup>18</sup> 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.

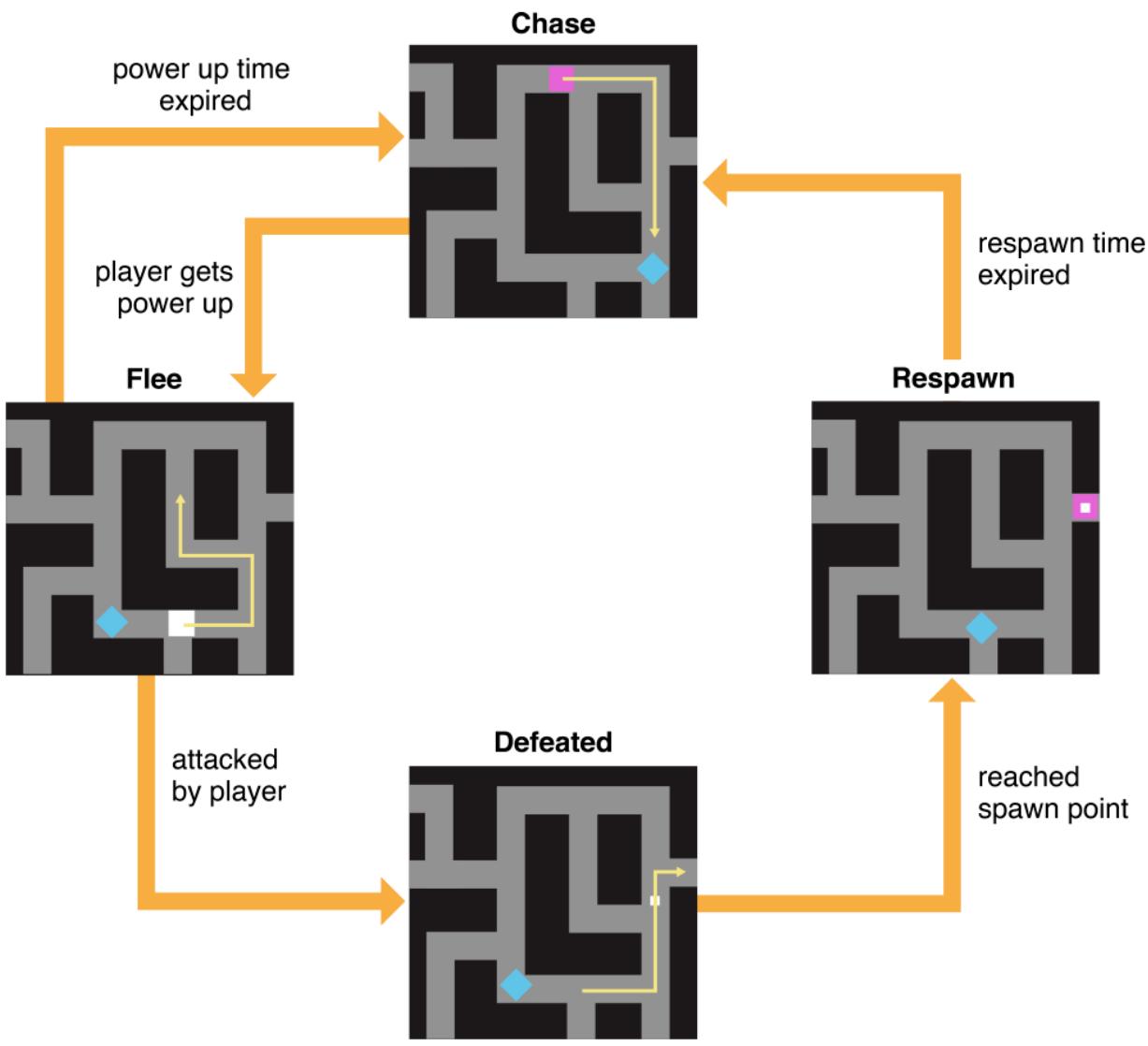
<sup>17</sup>[https://en.wikipedia.org/wiki/Finite-state\\_machine](https://en.wikipedia.org/wiki/Finite-state_machine)

<sup>18</sup>[https://en.wikibooks.org/wiki/A-level\\_Computing\\_2009/AQA/Problem\\_Solving,\\_Programming,\\_Data\\_Representation\\_and\\_Practical\\_Exercise/Problem\\_Solving/Finite\\_state\\_machines](https://en.wikibooks.org/wiki/A-level_Computing_2009/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Problem_Solving/Finite_state_machines)



**Sample FSM and PacMan FSM**

Below is an example of using a FSM for Opponent behavior.



Apple Gameplay Kit: Enemy Behavior State Machine



**Note:** Here is a tutorial on "How To Design A Finite State Machine"<sup>19</sup>, worked out from start to finish.

Machina.js<sup>20</sup> "... 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."

<sup>19</sup>[http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM\\_Tutorial.pdf](http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf)

<sup>20</sup><http://machina-js.org/>

## FSM Resolving Combat Outcomes

### 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:

$$\text{Hit Percentage} = (\text{Characteristic} * 2) + (\text{SpecificSkill} * 5)$$

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 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 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***"<sup>21</sup> of attacker and defender.

---

<sup>21</sup>[http://www.lukemartin.com/testing/phrases/cgi-bin/database.cgi?database2=phrases&action=view\\_product2&productID=288&category2=Latin](http://www.lukemartin.com/testing/phrases/cgi-bin/database.cgi?database2=phrases&action=view_product2&productID=288&category2=Latin)

Combat Matrix		Defender		
		0	1	10
Attacker	0	0	-1	-10
	1	1	0	-9
	10	10	9	0

FSM for Combat Outcome resolution

It is possible that your character's Hit Percentage may exceed 100%. This shows a high level of combat mastery, and the potential to engage multiple adversaries at once. If your character 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.

```
Attitude FSM object for predispositions:  
var attitude = {hate:-1, neutral:0, like:2}
```

Pre-disposition	Columns Shifted	Actions / Attitudes				
		Hand to Hand (Grapple)	Fight	Idle	Flight	Talk
Hate	-1	X	Engage	*	Morale check	Morale check
Neutral	0	Morale check	Morale check	*	Morale check	Morale check
Like	2	Continue	Continue	X	Fire	Continue
		Disengage	Disengage	X	Fire	Morale check
		Fight or Weapons	Move to H2H	Fire	Fire	Fire

\* - Start

X - N/A

#### Prestigious 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.

- **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.
- **Third combat round begins.** The Orc's "morale checked" failed and attempts to withdraw from combat; it forfeits its attack in order 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]);
}
```

---

## 9.7 Recursive World Feedback

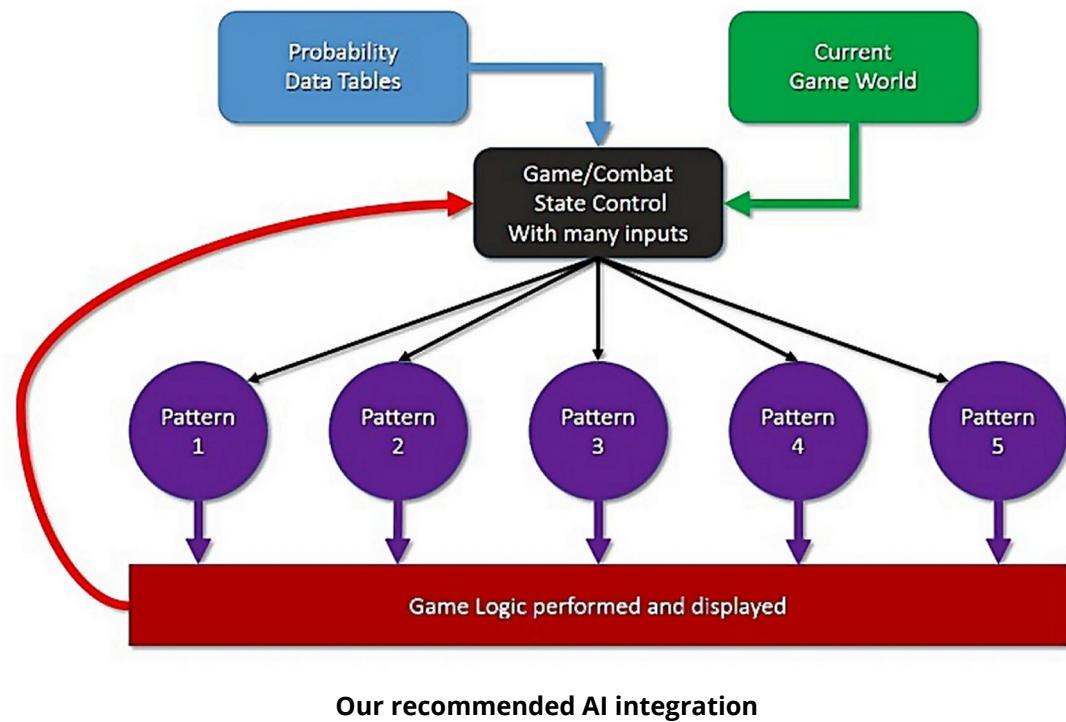
Let's get to the fun part of AI — ***the environment-driven Finite State Machine (FSM)!*** We could force an FSM to make state changes based on a set of parameters that

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.



## 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.

```
//Probability Table with 50% of selecting "1"
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 appended 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<sup>a</sup>

<sup>a</sup><https://phaser.io/docs/2.6.2/Phaser.Loader.html>

## 9.8 Complete AI Prototypes

We have moved the Source Code Appendix onto a website and removed it from edition 1 book release. This allows us to update code changes dynamically for Phaser v3 as it nears completion.

## 9.9 Chapter Source Code

<http://makingbrowsergames.com/book/index.html>

## 9.10 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).
- Play sample demonstration of AI.
- 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.

## 9.11 Footnotes

- Machina.js<sup>22</sup>
- "How To Design A Finite State Machine"<sup>23</sup>
- "Patrolling" monsters<sup>24</sup>
- Implement patrolling here.<sup>25</sup>
- "Flocking" is another AI pattern discussed here with Phaser examples.<sup>26</sup>
- "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<sup>27</sup> to your liking. Their Phaser Plugin<sup>28</sup> is open source.
  - GameDevAcademy has my favorite tutorial on Path finding here.<sup>29</sup>
- My favorite author wrote an interesting article<sup>30</sup> on the use of "antenna feelers" for a top-down racing game.
- My favorite author wrote an interesting article<sup>31</sup>

<sup>22</sup><http://machina-js.org/>

<sup>23</sup>[http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM\\_Tutorial.pdf](http://www.cs.princeton.edu/courses/archive/spr06/cos116/FSM_Tutorial.pdf)

<sup>24</sup><https://phaser.io/news/2016/04/patrolling-enemy-ai>

<sup>25</sup><http://www.emanueleferonato.com/2015/03/05/create-an-html5-game-like-drop-wizard-with-phaser-patrolling-enemies/>

<sup>26</sup><https://processing.org/examples/flocking.html>

<sup>27</sup><https://phaser.io/news/2016/02/how-to-use-pathfinding-in-phaser>

<sup>28</sup>[https://github.com/appsbu-de/phaser\\_plugin\\_pathfinding](https://github.com/appsbu-de/phaser_plugin_pathfinding)

<sup>29</sup><https://gamedevacademy.org/how-to-use-pathfinding-in-phaser/?a=47>

<sup>30</sup><http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/>

<sup>31</sup><http://www.emanueleferonato.com/2010/06/28/create-a-flash-racing-game-tutorial-artificial-intelligence/>

# 10. 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 “;)”!!

## 10.1 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 why the code is not working as expected.



I've found “**Log Rocket**”<sup>1</sup> 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 a period of 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.

---

<sup>1</sup><https://logrocket.com/>

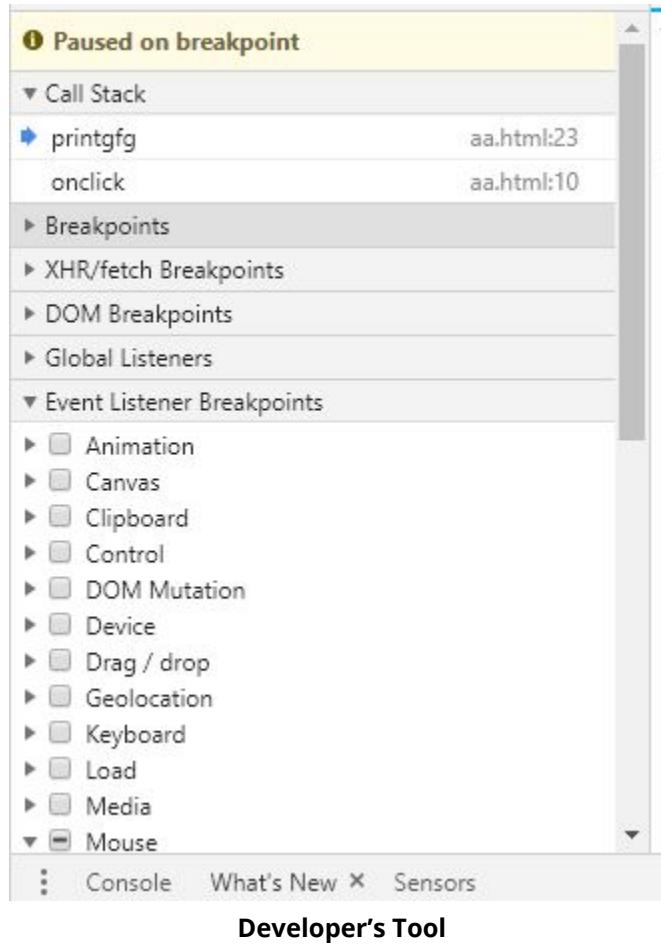
### JavaScript debugger

---

```
1 // html page
2 <p>The solution of 20 * 5 is: <span id="test"></span> </p>
3
4 <script>
5     var x = 20;
6     var y = 5;
7     var z = x * y;
8
9     debugger;
10
11    document.getElementById("test").innerHTML = z;
12 </script>
```

---

Enter Developers tool section by pressing **F12** key and go to “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 the the canvas — and how using workers can help in this situation.

## Deeper Dive: Console Commands



**Exercise:** Review **Console Commands here.**<sup>2</sup>



**Exercise:** Study the how to use the console from the **Chrome Console Tutorial here**<sup>3</sup>

<sup>2</sup><https://developer.mozilla.org/en-US/docs/Web/API/Console>

<sup>3</sup><https://developers.google.com/web/tools/chrome-devtools/console/>

- `Console.assert()` — Log a message and stack trace to console if the first argument is false.
- `Console.clear()` — Clear the 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();
```

<b>▼ doStuff()</b>	<b><u>VM178:10</u></b>
<b>▼ name</b>	<b><u>VM178:2</u></b>
first: Wile	<b><u>VM178:3</u></b>
middle: E	<b><u>VM178:4</u></b>
last: coyote	<b><u>VM178:5</u></b>

### 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

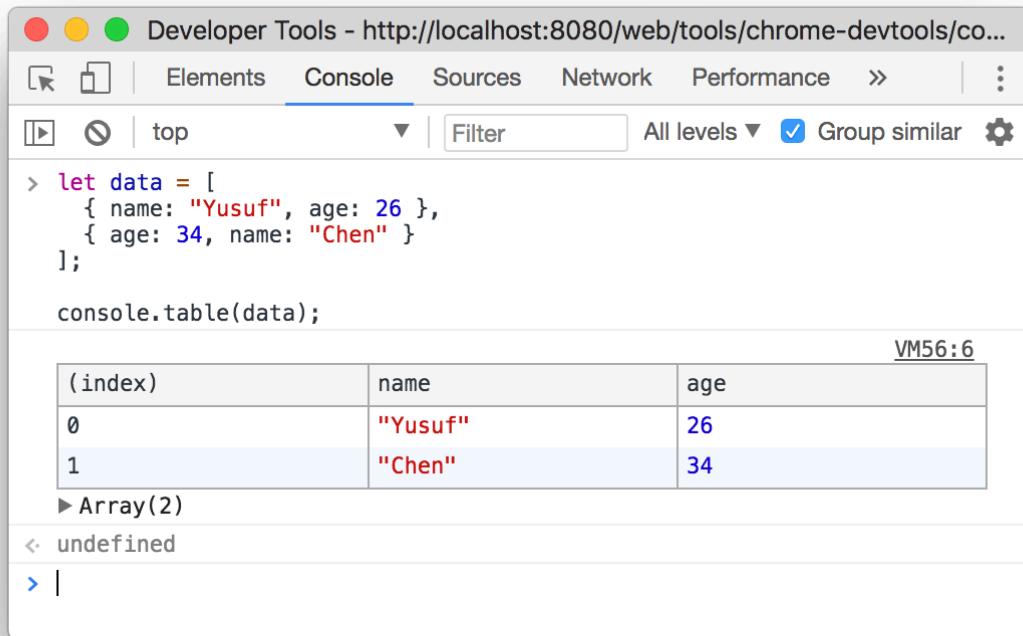
---

```

1  let data = [
2    { name: "Yusuf", age: 26 },
3    { age: 34, name: "Chen" }
4  ];
5
6  console.table(data);

```

---



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. A code snippet is being run in the console:

```
> let data = [
    { name: "Yusuf", age: 26 },
    { age: 34, name: "Chen" }
];
console.table(data);
```

The output is a table titled 'VM56:6' containing the following data:

(index)	name	age
0	"Yusuf"	26
1	"Chen"	34

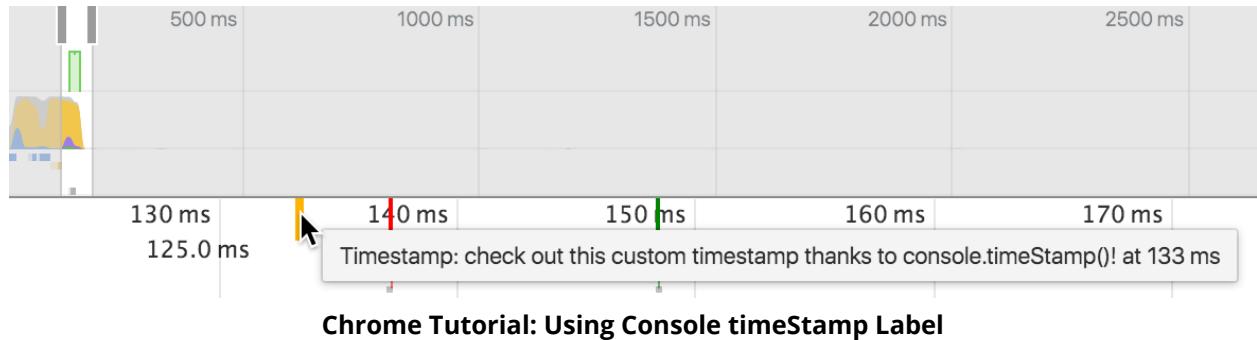
Below the table, the console shows:

```
▶ Array(2)
< undefined
> |
```

### 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 the ***Using the Timeline Tool guide***<sup>4</sup>

<sup>4</sup><https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool>



- `Console.trace()` — Outputs a stack trace.

```
> function add(num) {
  if (num > 0) {
    // you can pass labels and objects to trace, too
    console.trace('recursion is fun:', num);
    return num + add(num - 1);
  } else {
    return 0;
  }
}
<- undefined
> add(3);
▼ recursion is fun: 3 VM771:4
  add @ VM771:4
  (anonymous function) @ VM790:1
▼ recursion is fun: 2 VM771:4
  add @ VM771:4
  add @ VM771:5
  (anonymous function) @ VM790:1
▼ recursion is fun: 1 VM771:4
  add @ VM771:4
  add @ VM771:5
  add @ VM771:5
  (anonymous function) @ VM790:1
```

**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**<sup>5</sup>

## 10.2 Same “Name-spaces”

When `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 cannot have the same name of an animation.

It is recommended to prefix the audio file names with a common label, for example sound effects with the `sfx` file name prefix, in order to avoid conflicts.

## 10.3 Callbacks

Quote from "**How Do I Organise Files in A Phaser.js Project?**"<sup>6</sup>

When using `<script>` tags in a 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<sup>a</sup>](#)

<sup>a</sup><https://glcheetham.name/2016/03/18/organise-files-phaserjs-project/>

---

<sup>5</sup>[https://developer.mozilla.org/en-US/docs/Web/API/Console#Browser\\_compatibility](https://developer.mozilla.org/en-US/docs/Web/API/Console#Browser_compatibility)

<sup>6</sup><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>

```

---

## 10.4 Missing Documentation

Lacking source code documentation and internal comments, in my opinion, is ***the worst sin<sup>7</sup> 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<sup>8</sup>***) I was thinking at that time. ***Can you relate senior software engineers?? (you know who I'm talking to!)***

It takes mere moments to insert comments into source code; and ***if you don't have time*** then use “***Dragon Speak<sup>9</sup>*** or the ***professional version<sup>10</sup>*** if your running a financially successful studio.

Creating documentation is a snap with open source tools such as ***JSDoc 3<sup>11</sup>*** JSDoc 3 is an API documentation generator for JavaScript only. JSDoc even has a ***toolkit<sup>12</sup>*** which can export to `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:

---

<sup>7</sup><https://www.merriam-webster.com/dictionary/sin>

<sup>8</sup><https://biblehub.com/commentaries/proverbs/6-12.htm>

<sup>9</sup><https://amzn.to/2q51UCN>

<sup>10</sup><https://amzn.to/2S4jVOA>

<sup>11</sup><https://github.com/jsdoc3/jsdoc>

<sup>12</sup><https://en.wikipedia.org/wiki/JSDoc>

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 datatype 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.
@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**<sup>13</sup> 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

---

<sup>13</sup>[https://en.wikipedia.org/wiki/Comparison\\_of\\_documentation\\_generators](https://en.wikipedia.org/wiki/Comparison_of_documentation_generators)



# Part III - Project Walk-through & Resources

Part III has “walk-through Tutorials” on how to create ***Game Recipes***™ either manually or with the help of our ***online automation tools***.<sup>14</sup>

---

<sup>14</sup><http://makingbrowsergames.com/gameDesigner/>



# 11. Phaser III Game Prototype Library

- **Book's website:** <http://makingbrowsergames.com/p3gp-book/>
- Phaser III Game Prototype Library: <http://makingbrowsergames.com/book/index10.html>
- Consolidated Book Examples:

<http://makingbrowsergames.com/p3gp-book/index3.html>

<http://makingbrowsergames.com/p3gp-book/index3a.html>

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch4-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/ch4-examples/)

[http://makingbrowsergames.com/p3gp-book/\\_p3demos/ch7-examples/](http://makingbrowsergames.com/p3gp-book/_p3demos/ch7-examples/)



# 12. Walk-through Tutorial Series

**Are you “new” to Phaser III JavaScript Gaming Framework?** Start with these first two easy tutorials. Then tackle the intermediate **Game Recipes™**.

## 12.1 Difficulty Rating: Easy

- **Making your first platform game**<sup>1</sup> — the Official Phaser v3.x.x Primer Tutorial
- **Phaser III Game Design Course**<sup>2</sup> (included with your book purchase; a \$19.95 value, FREE!) — joins the 100s who've already earned their **Phaser III Game Developer Certifications**. Free license included in this Bonus Content.

## 12.2 Difficulty Rating: Moderate or Intermediate

- **Blood Pit (walk-thru tutorial)**<sup>3</sup> (included with your book purchase; a \$19.95 value, FREE!) — hero vs hero combat; an in-game module expansion for the **“Legends of Renown Deeds”**.<sup>4</sup> A **\$60 value**<sup>5</sup> included in this book.
- **Kiko Escapes**<sup>6</sup> — a “Clue Mystery Game” (“Cludo” in the UK) Engine. Find and rescue the Fairies’ Queen.
- **Red Fountain Swordsman** — modeled after the popular game “Guitar Hero” for sword-play competition.

## 12.3 Difficulty Rating: Advanced (aka “The Full Monty!”)

- **Ruins of Able-Wyvern™: (play here)**<sup>7</sup> source code License included with this book. Use your book invoice number and **registration here**.<sup>8</sup> Rule book included FREE — sold separately on Amazon.com. Searching for **“Stephen Gose”**.<sup>9</sup>

---

<sup>1</sup><https://phaser.io/tutorials/making-your-first-phaser-3-game>

<sup>2</sup><https://leanpub.com/c/p3gdc/c/Tx4iHQ6m64c5>

<sup>3</sup><https://leanpub.com/c/bloodpit-wtts/c/9uLERavhWp7n>

<sup>4</sup><http://legends-of-renown-deeds.com/>

<sup>5</sup><http://shop.pbmcube.net/downloads/adventurers-of-renown-blood-pit/>

<sup>6</sup><http://makingbrowergames.com/p3gp-book/index-bonus-clue.html>

<sup>7</sup><http://www.adventurers-of-renown.com/quests/arra.php>

<sup>8</sup><http://stephen-gose.com/affiliates-program/signup/>

<sup>9</sup>[https://www.amazon.com/s/ref=nb\\_sb\\_noss\\_2?url=search-alias%3Daps&field-keywords=stephen+gose](https://www.amazon.com/s/ref=nb_sb_noss_2?url=search-alias%3Daps&field-keywords=stephen+gose)

**References:**

- ***Creating Adventure Games On Your Computer*** by Tim Hartnell<sup>10</sup>
- ***Modular Game Worlds in Phaser 3***<sup>11</sup>
- ***How to Make a Roguelike Game***<sup>12</sup>
- ***RogueLike Engines***<sup>13</sup>

**The Ruins of Able-Wyvern™**<sup>14</sup> (ARRA), rv\_8 1994-2008 release, is the first introductory game-module of man-to-Hero tactical combat with arcane weapons in the ***Adventurers of Renown™***<sup>15</sup> 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 and flash plugin versions here<sup>16</sup>. We are building a new HTML5 version using Phaser. The Town of Lake-Shore provides the best place to begin your first-few campaign. This game is statistically balanced, and allows your newest characters to grow in status, skill and renown before launching into the sister game — Legends of Renown Deeds™: The Hero's Quest (LoRD).<sup>17</sup> (... also continuously online since 1994!)

<sup>10</sup><https://amzn.to/2LAkjb5>

<sup>11</sup><https://itnext.io/modular-game-worlds-in-phaser-3-tilemaps-3-procedural-dungeon-3bc19b841cd>

<sup>12</sup>[http://www.gamasutra.com/blogs/JoshGe/20181029/329512/How\\_to\\_Make\\_a\\_Roguelike.php](http://www.gamasutra.com/blogs/JoshGe/20181029/329512/How_to_Make_a_Roguelike.php)

<sup>13</sup>[http://www.roguebasin.com/index.php?title=RogueLike\\_Engines](http://www.roguebasin.com/index.php?title=RogueLike_Engines)

<sup>14</sup><http://shop.pbmcube.net/adventurers-of-renown-ruins-of-able-wyvern/>

<sup>15</sup><http://renown-games.com>

<sup>16</sup><http://www.adventurers-of-renown.com/quests/arra.php/>

<sup>17</sup><http://www.legends-of-renown-deeds.com/>



The Ruins of Able-Wyvern™ (ARRA), © 1994 — 2007, original Flash plugin expansion module

**Download and Review** the source code *from the website*.<sup>18</sup>  
 (v2.x x) <http://makingbrowergames.com/book/index11.html>  
 (v3.16+) [http://makingbrowergames.com/p3gp-book/\\_p3demos-arrav15/](http://makingbrowergames.com/p3gp-book/_p3demos-arrav15/)



*Hint: Included FREE with your book purchase are two Lifetime license (each valued at \$60) and Bonus Content Source Code<sup>19</sup> for "The Ruins of Able-Wyvern™ (ARRA)" versions rv\_8!*

<sup>18</sup><http://makingbrowergames.com/p3gp-book/>

<sup>19</sup><http://makingbrowergames.com/book/index11.html>



**Adventurers of Renown: Ruins of Able-Wyvern™ (rv\_15 2010 release)**

We have moved the Source Code Appendix onto a website and removed it from edition 1 to 3 book edition. This allows us to update code changes dynamically.

## 12.4 Source Code is here (online)

***v8 Life-time license (a \$60 value included with this book's purchase!)<sup>20</sup>***

***<http://makingbrowsergames.com/book/index.html>***

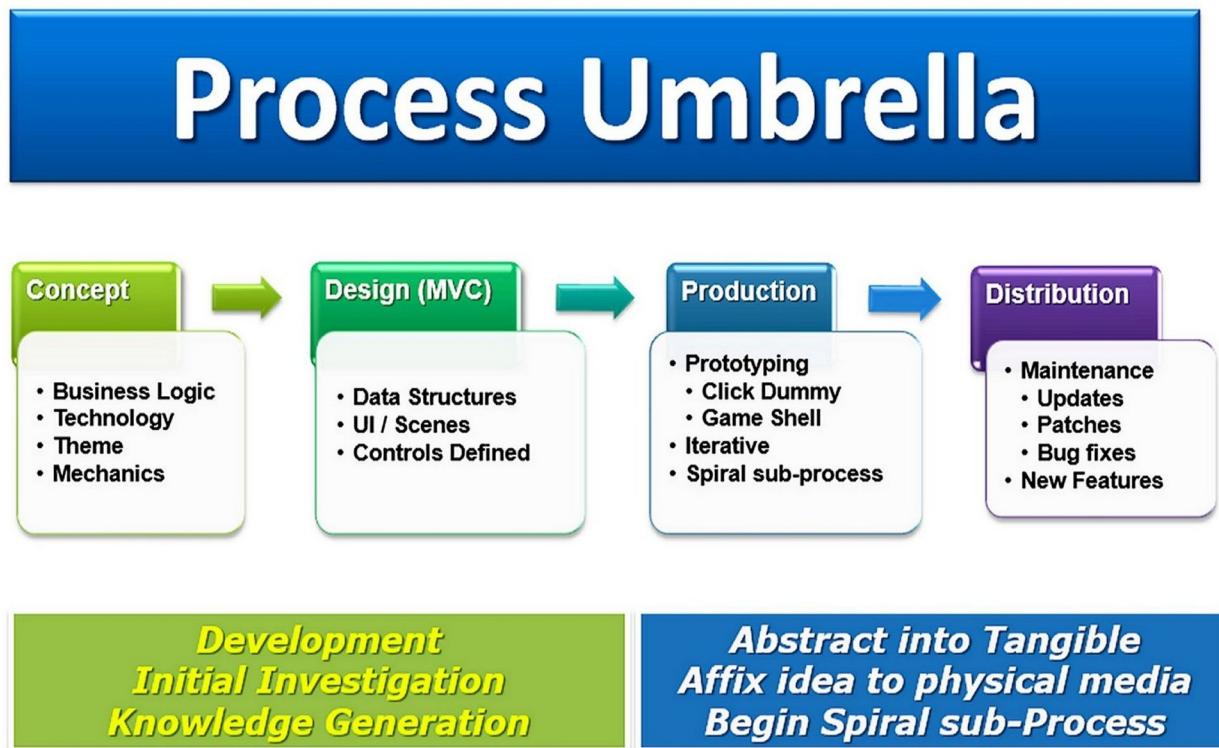
- JS Modules Listed alphabetically.
- ***Refer to your download Bonus Content Files***
- We use the Phaser Editor<sup>21</sup> to develop the room sub-states.

---

<sup>20</sup><http://shop.pbmcube.net/downloads/adventurers-of-renown-ruins-of-able-wyvern/>

<sup>21</sup><http://phasereditor.boniatillo.com/>

# Part IV - Next Steps ... Distribution!



### Part III: Next Steps ... Distribution Preparation

Part IV covers preparing your new game for “**Golden General Release**”.<sup>22</sup>

“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",

<sup>22</sup>[https://en.wikipedia.org/wiki/Software\\_release\\_life\\_cycle](https://en.wikipedia.org/wiki/Software_release_life_cycle)

which has only one scene, and exported it as a web game. OK, now I have a HTML and a UNITY3D 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 if someone could explain me how to publish my "game" on the web step by step. By the way, I've created a **WIX site,<sup>a</sup>** (*ed.: I forbid my students using WIX when attempting to demonstrate their Web Developer or Gaming Programming Skills in their portfolios.*) but I'm not sure if I can put a Unity3d game in there. *Read answers here<sup>b</sup>*

<sup>a</sup><https://www.wix.com/freesitebuilder/hiker-create>

<sup>b</sup><https://answers.unity.com/questions/59535/how-to-publish-a-game-on-the-web.html>

### **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<sup>23</sup>** gives a proper perspective.
2. Contact those publishers, and discover their submission policies. Read carefully about surrendering your rights. Learn what game genre peek their interests.
3. Create your game .... **(duh!) You can't sell "blue-sky" ideas.**
4. Create **a domain name and game website.<sup>24</sup>** (NOTE: Some ISPs **include a 1-year FREE domain name<sup>25</sup>** 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.<sup>26</sup>**
5. Refine your game mechanics. Get **strangers and other indie developers to play it.<sup>27</sup>** An excellent place to find participants is the new **Phaser Forum.<sup>28</sup>** 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.**
9. Start/Release your next/following game project(s). ... return to step #1

<sup>23</sup>[https://medium.com/@sm\\_app\\_intel/a-bunch-of-average-app-revenue-data-and-why-you-should-ignore-it-2bea283d37fc](https://medium.com/@sm_app_intel/a-bunch-of-average-app-revenue-data-and-why-you-should-ignore-it-2bea283d37fc)

<sup>24</sup><http://gose-internet-services.net/openvz-vps-2/>

<sup>25</sup><http://gose-internet-services.net/domain-names-hosting/>

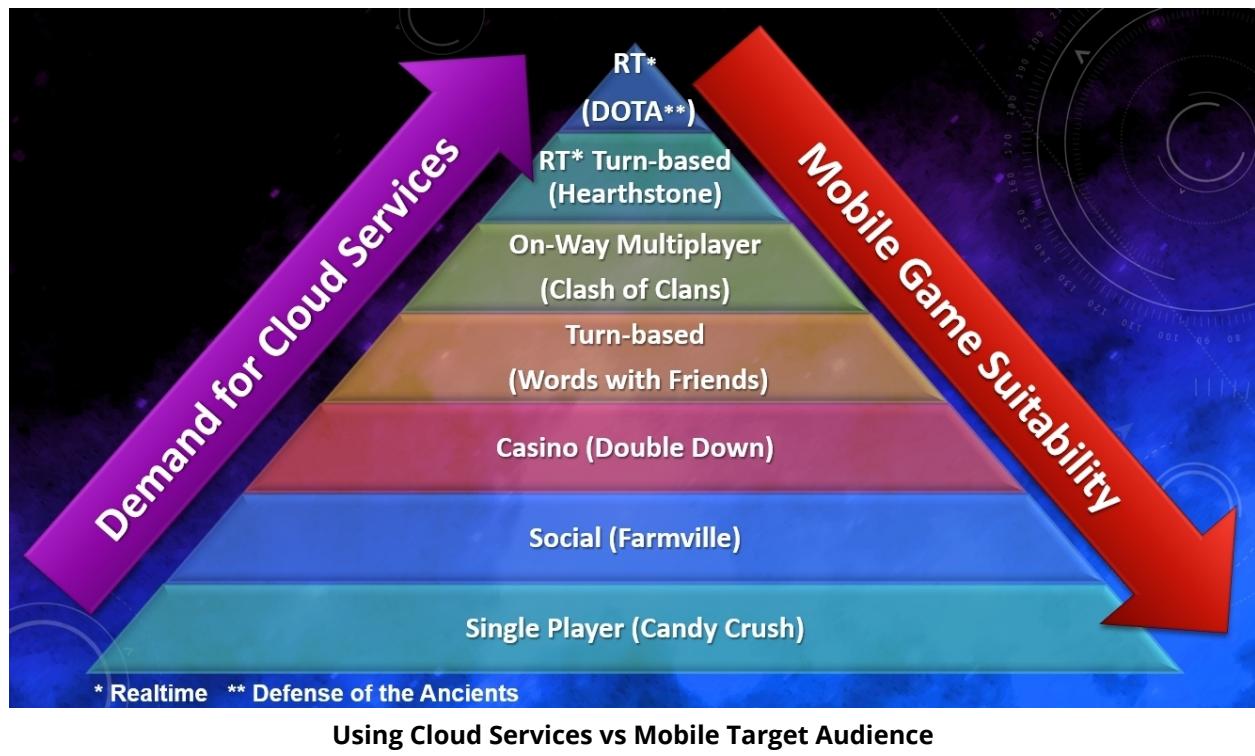
<sup>26</sup><http://makingbrowsergames.com/book/ch2/index.html>

<sup>27</sup><https://phaser.discourse.group/c/showcase>

<sup>28</sup><https://phaser.discourse.group/c/showcase>

<b>App Revenue from Action Games</b>	
From in-app purchases on 10/17/2016 on iOS & Android U.S. smartphones	
Top	\$2,188,000 (Clash of Clans)
Average	\$8,400
80th Percentile	\$3,100
50th Percentile (Median)	\$150
20th Percentile	\$0

Annual ROI for Mobile Game Markets



# 13. Distribution Preparation: Your Game Product

Now! Let's talk about those "***Front-end Building Tools***"!<sup>1</sup> If you haven't already, it would be beneficial to read:

- "***I finally made sense of front end build tools. You can, too.***"<sup>2</sup>
- "***Front-end Building Tools***".<sup>3</sup>
- ***Front-end Developer Tools***<sup>4</sup>

## 13.1 Development vs. Production

Parts I & II have given you a game development "pipeline" to mass produce game products across several game mechanics. You mixed the game mechanics and mechanism prototype components to your liking. You've found artwork and a theme and blended them into your game recipe concoction. Your doodles **have become a true skeletal framework/engine with "new muscles (game prototypes), organs (game mechanics) and flesh (artwork)**". It's time to massage your "Frankenstein" into a creation for the world to see. Now what?

The error that most new game studios make, is that they assume that other "game developers" are their target markets. So, they launch announcements on the Phaser forums, join the Phaser "discourse.groups" to market their games. Lots of developers play games; it was one of the mandatory steps I gave you. Don't confuse your competitors with your target audience and don't make the mistake that our game developers peer group is our market place.

---

<sup>1</sup><https://developers.google.com/web/tools/setup/setup-buildtools>

<sup>2</sup><https://medium.freecodecamp.org/making-sense-of-front-end-build-tools-3a1b3a87043b>

<sup>3</sup><https://developers.google.com/web/tools/setup/setup-buildtools>

<sup>4</sup><https://frontendmasters.com/books/front-end-handbook/2018/tools.html>

## 13.2 Create A Game Pipeline

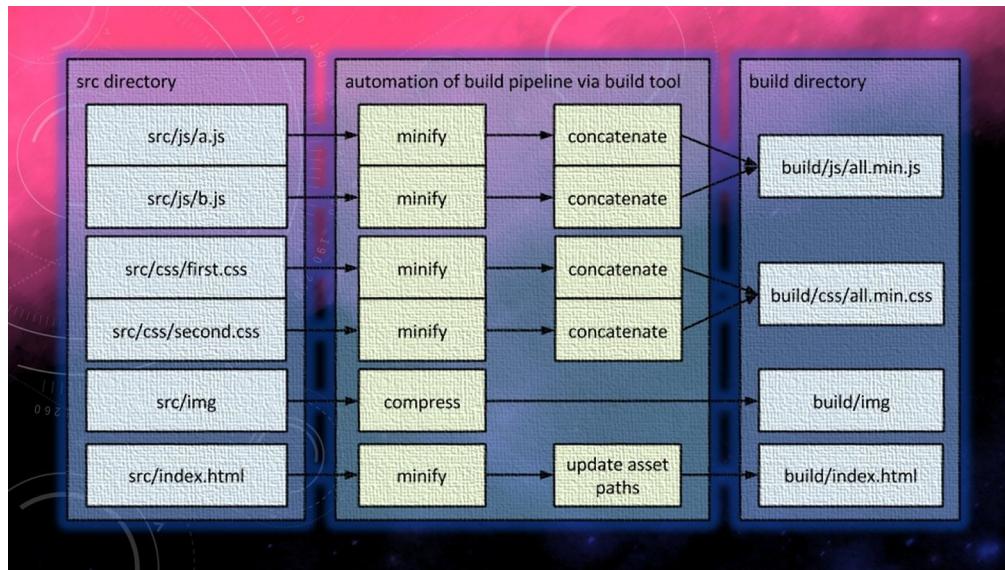
Gavin Davies wrote a wonderful tutorial entitled ***Using build tools (31 pages)***.<sup>5</sup> He illustrates “5 simple steps” to create a Front-end pipeline process. Read more details from his ***blog article or download his booklet***.<sup>6</sup>



From his illustration above, we have “designed” and “edited” our game. It’s now time to “build” our game for distribution. He shows that our next steps should be to “minify” and then combine (aka concatenate) all our game files. Let’s take a closer look into those “green colored” steps. He shows which game assets will undergo which process and that we can further “automate” our game production pipeline.

<sup>5</sup><http://radify.io/downloads/Using-Build-Tools.pdf>

<sup>6</sup><http://radify.io/blog/using-build-tools/>



**Creating an Automated Build pipeline**

This stage attempts to compact your game and resources as much as possible. All the initial code prototypes were inside one monolithic file. We need to return to a single JS file prior to distribution. You also may not want gamers digging into your JS source code; so, you may want to obfuscate (aka disguise) and preserve your technical innovations. A useful tool for just this thing is **Google Closure**<sup>7</sup>

### 13.3 Preparing for Mobile Deployment

The best place (and tailor specifically for Phaser v3.x.x) is the PhoneGap. Review what they have **available here**.<sup>8</sup> Cocoon was another popular JS helper to prepare your game for the mobile market. But by the time read this, **Cocoon is gone ... shutdown as of 20190204!**<sup>9</sup> They offer a helpful guide to migrate and **substitute Cocoon.io for Apache Cordova**.<sup>10</sup> The Phaser v3.x.x has a helpful **article here**.<sup>11</sup> Josh Morony offers more advice on using **Cocoon.io with Phaser v3.x.x**.<sup>12</sup> Emanuele Feronato provides even more guidance when developing games without the **Android Studio and CocoonJS**.<sup>13</sup>

<sup>7</sup><https://developers.google.com/closure/>

<sup>8</sup><https://cordova.apache.org/plugins/?q=phaser>

<sup>9</sup><https://blog.cocoon.io/>

<sup>10</sup><https://docs.cocoon.io/article/from-cocoon-io-to-apache-cordova/>

<sup>11</sup><http://www.html5gamedevs.com/topic/31828-using-cocoon-with-phaser/>

<sup>12</sup><https://www.joshmorony.com/how-to-build-html5-mobile-games-with-cocoonjs/>

<sup>13</sup><https://www.emanueleferonato.com/2017/11/01/step-by-step-guide-to-create-android-native-games-in-html5-with-cocoon-io-and-without-android-studio/>

It's only fair to warn you that there are problems preparing your mobile games with Cocoon. Visit the Cocoon forum and **read through those issues.**<sup>14</sup>



**Exercise:** Review ***the journal of another indie game developer***<sup>15</sup> using Phaser v3.x.x. and Cordova.

## Does mobile gaming still need publishers?

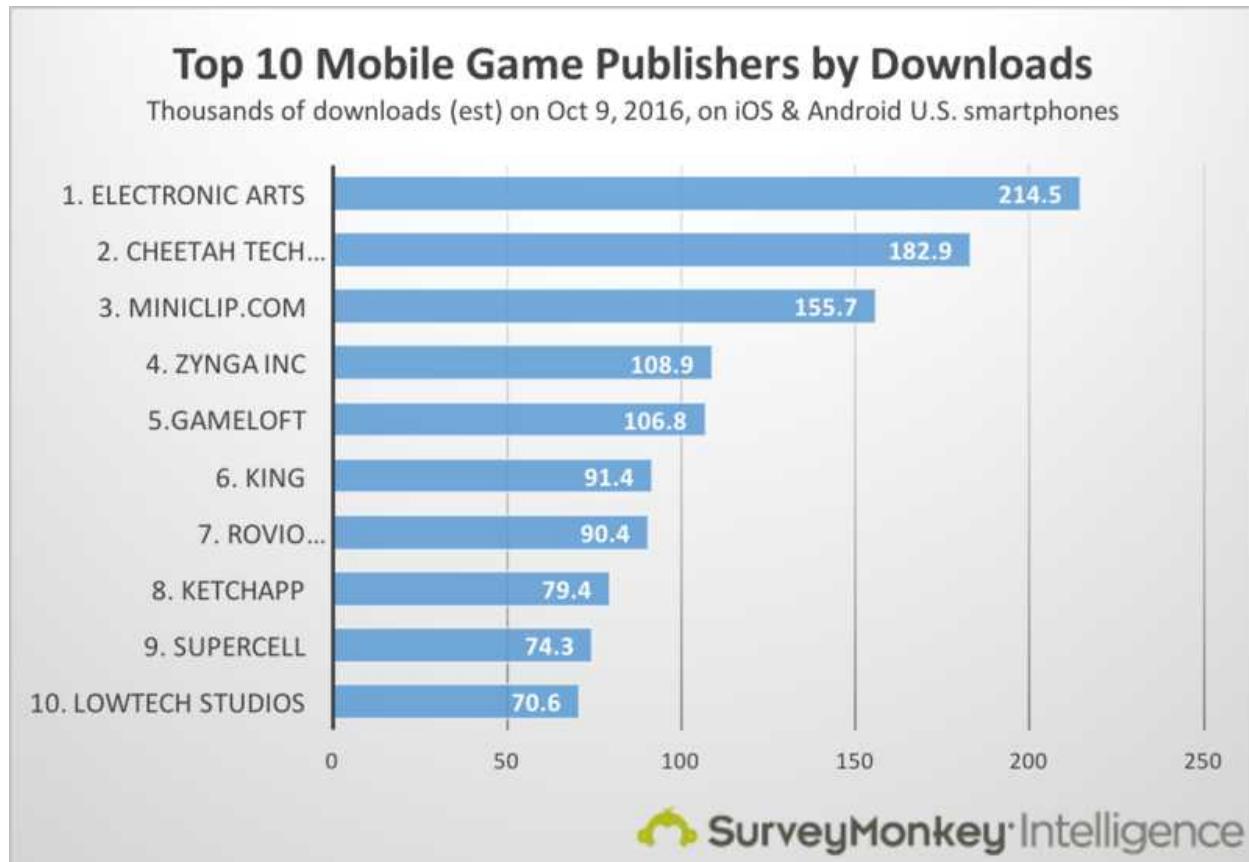
"Today, mobile has furthered transformed the distribution paradigm for game developers as it's now exponentially harder for indie devs to compete. As the app market became more open, the capital and skills required to compete for top rankings increased. This served to restrict the top end of the market to those with deep pockets, which begs the question: While any mobile developer can publish their own game, should they?"

. . . To put that in perspective, the **top 10 grossing iPhone games** on the App Store at the time of writing, which are **all free to install, average 29,544 downloads per day**. So, in order to compete, ***you can plan on spending \$1,497,585 daily*** just to get those users to the point where they make their first purchase. Read complete ***article here.***<sup>a</sup>

<sup>a</sup><https://venturebeat.com/2018/02/03/does-mobile-gaming-still-need-publishers/>

<sup>14</sup><https://forums.cocoon.io/t/cocoon-developer-app-phaser-js-canvas-issue/7204>

<sup>15</sup><https://thomashunter.name/posts/2018-03-27-developing-cobalt-dungeon-using-phaser-and-cordova>



There are many prolific mobile game publishers in the app stores. Of the publishers who have released the most mobile games, **five of them don't show up in any of the previous top 10 lists mentioned in this post**. The following publishers fail to appear in the top 10 for downloads, revenue, users, and engagement.

- Glu Games (at least 44 games),
- Storm8 (39 games),
- Disney (35),
- Square Enix (22), and
- Warner Bros. (19)



**Hint:** See more information on best mobile publishers *in this article, "The top mobile game publishers by downloads, revenue, and users" By Mike Sonders*.<sup>16</sup>

<sup>16</sup>[https://medium.com/@sm\\_app\\_intel/the-top-mobile-game-publishers-by-downloads-revenue-and-users-a13be2693718](https://medium.com/@sm_app_intel/the-top-mobile-game-publishers-by-downloads-revenue-and-users-a13be2693718)

Publisher	# of Games	Avg Downloads per Game	Avg Revenue per Game	Avg Monthly Users per Game
Supercell	4	18,600	\$675,000	2,875,000
King	15	6,100	\$93,700	1,420,000
Zynga	33	3,300	\$17,100	558,000

**The best of the best mobile game publishers**

On average, ***Supercell drives the best performance*** on a per-game basis among the three top game publishers, based on performance numbers



**Exercise:** Prepare yourself on what to expect when submit a mobile game to a publisher. Read "***How to Choose a Mobile Game Publisher***"<sup>17</sup>

## 13.4 Chapter References:

- ***Rebuilding An HTML5 Game In Unity***<sup>18</sup>
- ***Principles Of HTML5 Game Design***<sup>19</sup>
- ***How To Design A Mobile Game With HTML5***<sup>20</sup>
- ***What Web Designers Can Learn From Video Games***<sup>21</sup>
- ***Finger-Friendly Design: Ideal Mobile Touchscreen Target Sizes***<sup>22</sup>

<sup>17</sup><https://partners.gamehouse.com/choose-mobile-game-publisher/>

<sup>18</sup><https://www.smashingmagazine.com/2014/04/rebuilding-an-html5-game-in-unity/>

<sup>19</sup><https://www.smashingmagazine.com/2015/09/principles-of-html5-game-design/>

<sup>20</sup><https://www.smashingmagazine.com/2012/10/design-your-own-mobile-game/>

<sup>21</sup><https://www.smashingmagazine.com/2011/07/what-web-designers-can-learn-from-video-games/>

<sup>22</sup><https://www.smashingmagazine.com/2012/02/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/>

# 14. Marketing Channels Deployment

It's time to distribute our game, we need to think about a few things before we launch our game into the "wild". First, we need to ensure future players, from all over the world, can understand our game. It's true that many people can read and write English, but it's better if our game dynamically adapts itself to a player's cultural region. For example, by translating ***all the text used in our game*** into the player's native language. This is known as ***localization***. If you want to do this properly, we needed to separate all text from the actual game code as much as possible. I've shown you how to do this in previous chapters using the game prototypes. Hopefully, you followed how to separate any text shown on buttons, tool-tips and helpful instructions shown when the player hovers over any user interface (UI) element. Localization can be costly if you want your game translated into every language of the world; but you can reduce those costs by ensuring the game code doesn't contain any "hard-coded" text elements and only deploys variables that referencing text defined in a single JSON data structure.

## 14.1 Channel Selection

Deciding which publisher and which platform to use for your game's distribution can be the next few challenges. Maybe you've published game before, maybe you haven't — but the bottom-line of this section is to offer a some helpful guidance what options are available.

Here's 18 recommendations — ***sorted alphabetically with no preference implied*** — that you may want to explore for publishing your game:

- Amazon Appstore <https://developer.amazon.com/why-amazon>
- Apple Store <https://developer.apple.com/programs/>
- Cloud Games <http://cloudgames.com/developers/>
- Big Fish <https://www.bigfishgames.com/daily/about/submit-games/>
- Envato <https://author.envato.com/>
- Game House <https://partners.gamehouse.com/about-us/>
- Game Jolt <https://gamejolt.com/marketplace>
- Gamefroot <http://gamefroot.com/>
- GamePix <https://www.gamepix.com/game-developers/>

- Gamers Gate <https://www.gamersgate.com/info/about-us>
- Gog <https://www.gog.com/indie>
- Google Play <https://developer.android.com/distribute/>
- Humble Bundle <https://www.humblebundle.com/publishing>
- **Independent Game Festival** <http://www.igf.com/submit-your-game>
- Itch.io <https://itch.io/developers>
- Kongregate <http://developers.kongregate.com/>
- Playsterr <http://playsterr.com/develop-games/>
- Spil Games <http://www.spilgames.com/developers/>
- Steam <https://partner.steamgames.com/steamdirect>
- Stephen Gose Game Studio <http://stephen-gose.com/affiliates-program/signup/>

## 14.2 What do I need?

Here's a skeleton list of the basic marketing support artifacts.

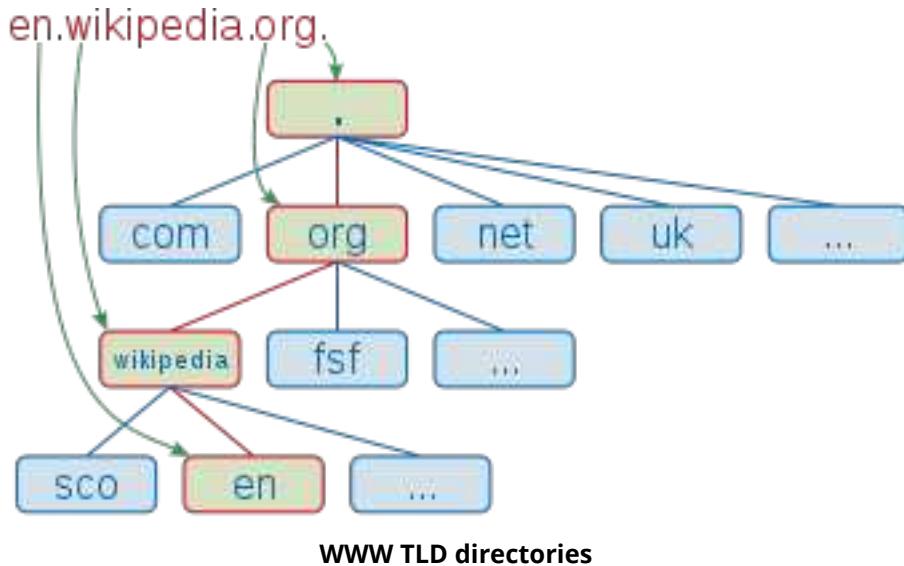
1. a “**Game Headquarters (HQ)**” such as a website **and domain name**.<sup>1</sup> Shop around and do some price comparisons before you commit. I can always spot the novice entrepreneur. Here’s what they typically do. They setup a Twitter and/or Facebook account, and they tell people and friends on Facebook to “follow me on Twitter” and then on Twitter to do the same on Facebook. Before too long, they start pointing more sites they’ve purchased to their “web ring”. If a gamer is interested in their game, they end up in a cyclone of whirling internet links. Naturally, they escape the spiderweb of confusion. **Your “Game Headquarters”** only goal is to sell your game product(s); it should be the central hub. Everything else should point into **Your “Game Headquarters”**. You also need **to OWN Your “Game Headquarters (HQ) ”**. If you think your Facebook page is your HQ and you spend all your time and money sending people there and then one day you are banned from Facebook or they change their policy (yea, that never happens??? We’re talking Facebook here!!). **What then??** Your website’s domain name should be short and easy to remember, all in lower case, and avoid (if possible) hyphens and underscores. It must be between 3 and 63 characters long. Domain names cannot begin or end with a hyphen symbol, and may not contain two consecutive hyphens. The entire domain name may not contain more than 253 characters — seriously?!. See further details here — **RFC 1035**<sup>2</sup>. The **Top Level Domains (TLD)**<sup>3</sup> provides your “purpose or **raison d’être**” on the World Wide Web (WWW). You will also want to use DNSSEC for commerce from your website; **not all TLD support DNSSEC**.<sup>4</sup> Placing your game products correctly on the WWW is the most important decision as you move forward.

<sup>1</sup><http://gose-internet-services.net/domain-names-registration/>

<sup>2</sup><https://tools.ietf.org/html/rfc1035>

<sup>3</sup>[https://en.wikipedia.org/wiki/Domain\\_name](https://en.wikipedia.org/wiki/Domain_name)

<sup>4</sup>[https://en.wikipedia.org/wiki/List\\_of\\_Internet\\_top-level\\_domains#Country\\_code\\_top-level\\_domains](https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains#Country_code_top-level_domains)



- **.com**<sup>5</sup> represents commercial endeavors and the best choice if you plan to sell your business in the future;
- **.net**<sup>6</sup> are for network cloud services and is the best choice if you plan to offer your unique gaming features to other game developers;
- **.org**<sup>7</sup> are for **non-profit** organizations and should be the primary consideration if you plan to offer your game products as "**Open Source**";
- **.edu**<sup>8</sup> are for legally registered educational institutions and is typically unavailable.
- **.info**<sup>9</sup> are for public publications for science research.
- various countries have a TLD designation. You should research whether your game qualifies for a country designation.
- **.io**<sup>10</sup> Before you even consider using this "... **trendy and fast-growing domain in the start-up world** ..." as your "top level domain" (TLD), please read **how it can about**,<sup>11</sup> and then decide **if you endorse**<sup>12</sup> what the **UK has done**.<sup>13</sup>

2. **A Trailer** (teaser movie). 60% of folks are "visually oriented". These days you **NEED** a video teaser to show off the game, art, and game play excitement. Marketing

<sup>5</sup><http://gose-internet-services.net/domain/com-domain-name/>

<sup>6</sup><http://gose-internet-services.net/domain/net-domain-name/>

<sup>7</sup><http://gose-internet-services.net/domain/org-domain-name/>

<sup>8</sup>

<sup>9</sup><http://gose-internet-services.net/domain/info-domain-name/>

<sup>10</sup>

<sup>11</sup><https://gigaom.com/2014/06/30/the-dark-side-of-io-how-the-u-k-is-making-web-domain-profits-from-a-shady-cold-war-land-deal/>

<sup>12</sup><https://gigaom.com/2014/07/11/uk-government-denies-receiving-io-domain-profits/>

<sup>13</sup>[https://en.wikipedia.org/wiki/Expulsion\\_of\\_the\\_Chagossians](https://en.wikipedia.org/wiki/Expulsion_of_the_Chagossians)

statistics has proven that a that “Trailers” increase conversion at 10% to 30%. Such tools as **Open Broadcast Software (OBS)**<sup>14</sup> help facilitate trailer construction. Learn from the professional and keep the time duration of your Trailer to the length of a “TV commercial”.

3. **Screen Shots** — are required by many distribution channels. These screen shots target those folks who don’t like watching videos. So using animated .gif may not be a good choice.

4. You’ve already created your **Elevator Speech**, right? Take it once more and tweak it to answer these questions.

- What’s your game is or about.
- Is it free or will it “cost” me.
- What will you get from the game.

5. **A Press Kit** — is just a web page (remember your Game HQ above?) that has a bunch of resources your affiliates can use to write about your game product and about you as a developer. For example, it should have screen shots from the live game, a video, an icon, quotes from players, awards won, game features, biography information about you and your game’s development, and logos. In short, it’s a package you can distribute which has the information they need to write articles on your game without contacting you directly. Download this “**press kit template**”.<sup>15</sup>

6. **Distribution Channels** — are the avenues you use to connect to players. Obvious choices are Steam, Google Play, any sites mentioned earlier or the App Stores. Refer to the Appendix — Channels for a comprehensive mobile and publishers.

### 14.3 Targeting Markets with the “Tower of Babel”

#### Internationalization (from Phaser Game Design Workbook 3rd Edition)

Does your game display text? in what language? Are you aware that English is only 52% of languages used on the Internet? Who are the other 48%? The number of non-English pages is rapidly expanding. The use of English online increased by around 281 percent from 2001 to 2011, a lower rate of growth than that of **Spanish (743 percent), Chinese (1,277 percent), Russian (1,826 percent) or Arabic (2,501 percent)** over the same period<sup>a</sup>. Ongoing monitoring by W3Techs<sup>b</sup> revealed in March 2015,

<sup>14</sup><https://obsproject.com/>

<sup>15</sup><http://dopresskit.com/>

just over 55% of the most visited websites had English-language homepages. Other top languages that are used (at least in 2% of the one million most visited websites according to W3Techs<sup>c</sup> ) are: Russian, German, Japanese, Spanish, French, Chinese, and Portuguese. Estimates of the quantity of Internet users by language as of November 30, 2015 are listed here.<sup>d</sup>. Of the 3.37 billion Internet users and native speakers' nationality, English represents only 25.3%;

- Chinese is 19.8%;
- Spanish is 8%;
- Arabic is 4.8%;
- Portuguese is 4.1%
- Japanese is 3%
- Russian is 2.8%
- Malay is 4.1%
- French is 2.8%
- German is 2.2%

\* All others is 23.1%

<sup>a</sup><https://web.archive.org/web/20130407032518/http://www.scottmclay.co.uk/foreign-language-internet-good-business>

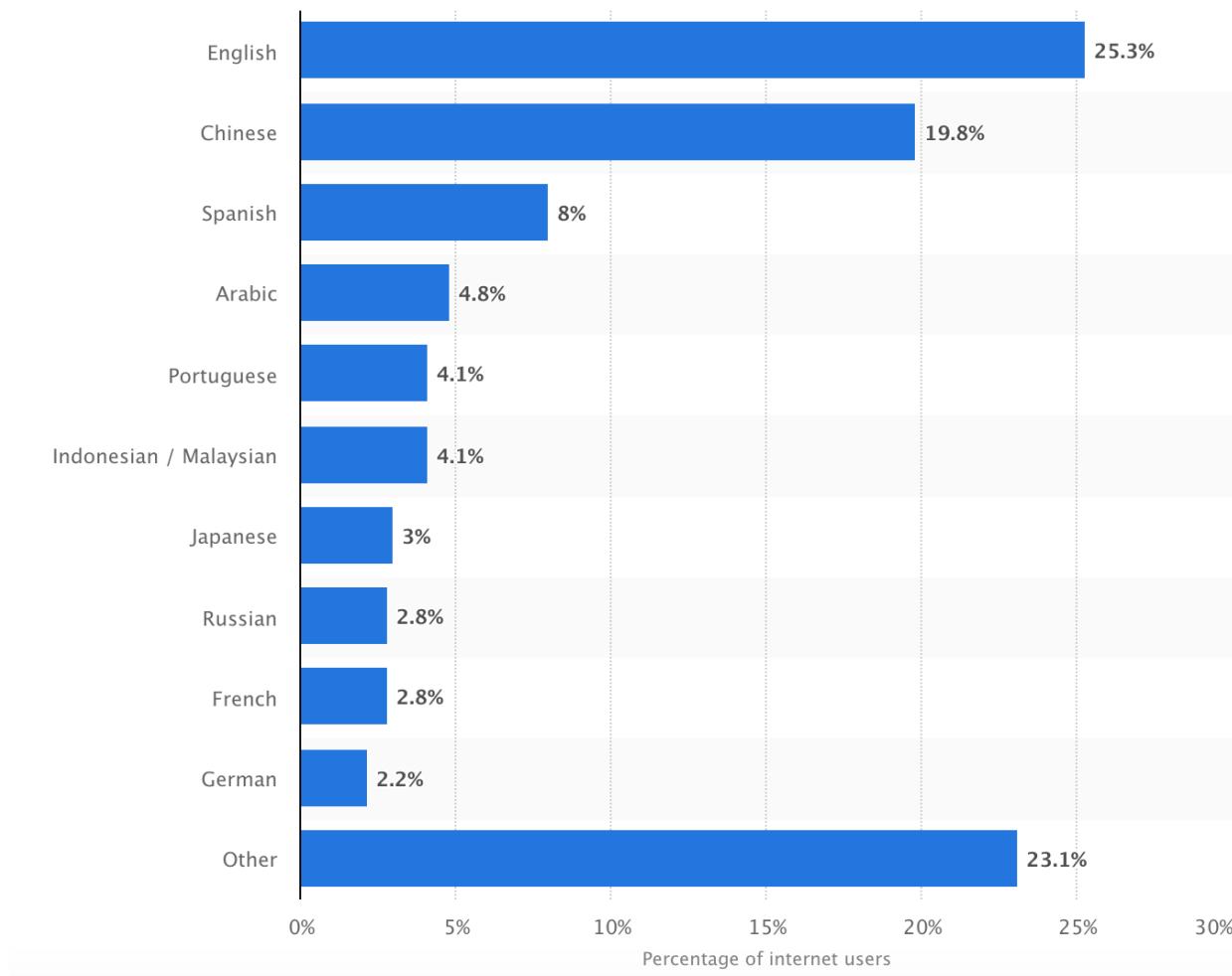
<sup>b</sup>[https://w3techs.com/technologies/overview/content\\_language/all](https://w3techs.com/technologies/overview/content_language/all)

<sup>c</sup>[https://w3techs.com/technologies/overview/content\\_language/all](https://w3techs.com/technologies/overview/content_language/all)

<sup>d</sup><http://www.internetworldstats.com/stats7.htm>



**Hint:** By 2040, India's population will exceed that of China and it will become the largest population on Earth.



**Exercise** How many more gamers would play your product(s) if they could read them? **Read more here**<sup>16</sup>

## 14.4 Channel Preparations

Startling evidence from Localytics shows that "...23% of Users Abandon an App After One Use<sup>17</sup>...". If this is true, then what are current game developers doing incorrectly? Humans haven't changed; is it something wrong in current game development? In Localytics' five-year study, they discovered:

<sup>16</sup><https://speakt.com/top-10-languages-used-internet/>

<sup>17</sup><http://info.localytics.com/blog/23-of-users-abandon-an-app-after-one-use>

The data proves it. The percentage of users who abandon an app after one use is now 23%, a slight improvement from the 25% we saw in 2015. But clearly, with about one in four users still only using an app once, not enough has been done to match what consumers want and restore apps to the success of just a few years ago.

With that in mind, today we are releasing an annual update to our app user retention study, which measures loyalty and abandonment across our user base of 37,000 apps. Five years in, we have a solid understanding of what user retention should look like as well as the factors that can cause it to fluctuate. Our study focuses on two key metrics:

- User Retention: The percentage of users who return to an app eleven or more times.
- User Abandonment: The percentage of users who abandon an app after one session.

In the study, we also found that user retention recovered from 34% in 2015, a dip of 5 percentage points from the previous year, to reach 38% in 2016. While this number is also an improvement, there is still work to be done in order to avoid churn and ultimately convert more users into loyal customers. Because even though 38% will return to an app 11 or more times, that means ***a whopping 62% will use an app less than 11 times. This is not a sustainable business model.***

The current statistics for 2016 show some improvement per their study, and they offer insider-secrets how to improve gamers' retention to 46% (i.e., encouraging players to use a mobile game more than 11+ times!). They recommend:

Apps utilizing in-app messages see 46% of their users come back 11 or more times, while apps who do not incorporate an in-app strategy see that number drop to 36%. In-app messages also have an effect early on in a user's lifecycle, as only 17% of users will only use an app once if they see an in-app message, while apps not utilizing in-app messages see 26% abandoning the app after one session.

Since users do not have to opt-in to in-app messages, they can be a great way to onboard users to the app and then keep them engaged with the most useful features of the experience. If used effectively, and personalized to the user, in-app messages are a strong tool in an app's arsenal to keep users around.

## Moving Forward

With more insights-driven mobile engagement like triggered in-app messages<sup>a</sup>, businesses will tip the scales towards success and rise out of the mobile engagement crisis. We'll be keeping a close eye on app user abandonment and retention over the next year with our quarterly benchmarks<sup>b</sup>. In the meantime, we encourage you to learn more about the steps towards mobile engagement success in our latest

infographic<sup>c</sup>.

<sup>a</sup><http://info.localytics.com/blog/in-app-messages-drive-higher-app-usage-and-engagement-benchmarks>

<sup>b</sup><https://www.localytics.com/resources/types/cheat-sheet/>

<sup>c</sup><http://info.localytics.com/blog/the-mobile-engagement-crisis-the-infographic>

Skeptical? Here's some background information on Localytics, their annual report and population sampling. Localytics is the leading mobile engagement platform across more than 2.7 billion devices and 37,000 mobile and web apps. Localytics processes 120 billion data points monthly. For this analysis, Localytics measured the percentage of users who only used an app once as well as the users who returned multiple times. For the in-app messaging analysis, we compared the percentage of users who did or did not return to apps who are using in-app messages compared to those who are not. The time-frame for the study was March 1st, 2015 - April 30th, 2016.

## 14.5 Generating a Profit

This question comes up in class lectures all the time, "What's the best way to monetize a game? Should I ask for payments, subscriptions or go **"Free-mium"**<sup>18</sup> This typically leads the class on a detour on these **7 examples of Free-mium done right**<sup>19</sup>

### When and Where to use Pricing

Gamer's Cost	Small Market Segment	Large Market Segment
High	Fails	Perhaps
Low	Limited Potential	Perfect



**Exercise:** Read the following articles.

- **The Ultimate Guide to Freemium Mobile Apps**<sup>20</sup>
- **5 Steps to Ignite your Free-to-Play Revenue (without Advertising)**<sup>21</sup>
- **Free-to-Play Monetization:**<sup>22</sup> 4 Popular Ways to Monetize Mobile Apps

<sup>18</sup><https://hbr.org/2014/05/making-freemium-work>

<sup>19</sup><https://www.forbes.com/sites/sujanpatel/2015/04/29/7-examples-of-freemium-products-done-right/#28304ef26f15>

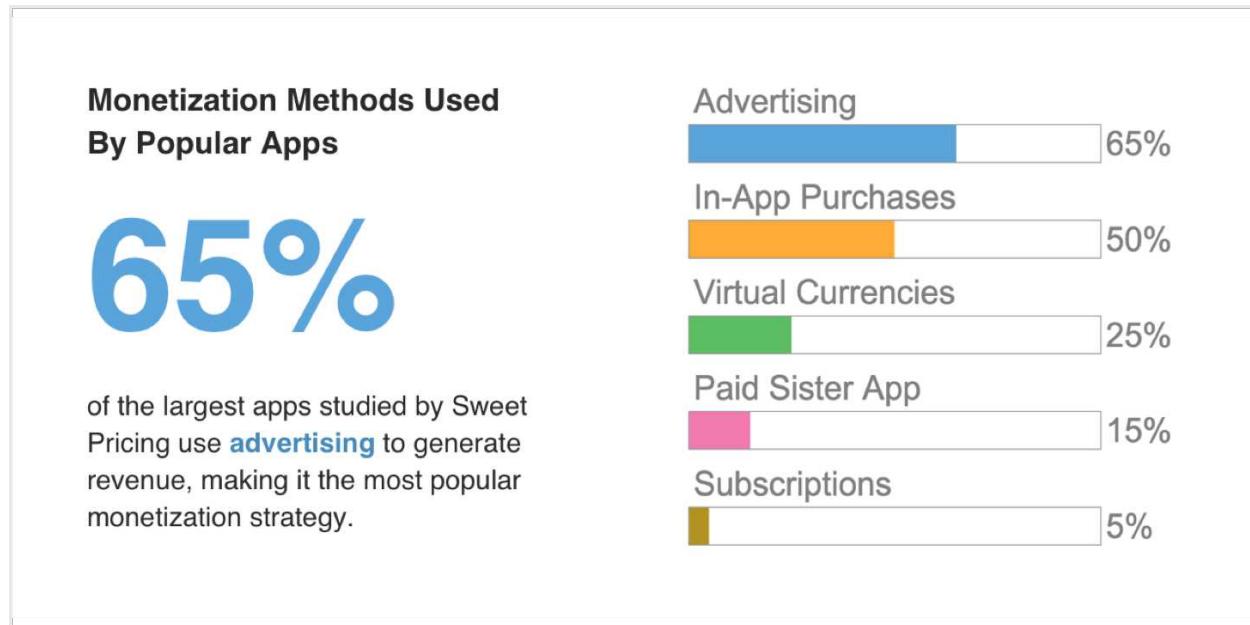
<sup>20</sup><https://sweetpricing.com/blog/2017/01/freemium-mobile-apps/>

<sup>21</sup><https://getbraincloud.com/apidocs/5-steps-to-ignite-your-free-to-play-revenue-without-advertising/>

<sup>22</sup><https://sweetpricing.com/blog/2016/06/free-to-play-monetization/>

- **App Monetization:**<sup>23</sup> 7 Stats From a Study of Popular Mobile Apps
- Promotional Pricing in Freemium Mobile Apps: **Time-Limited Offers**<sup>24</sup>

## In-Game Purchases



### Who uses what?

The second best cash generation method is **in-game purchases (IGP) (aka in-app purchases (IAP))<sup>25</sup>**. You give free access into your game, and then you offer enhancements, entitlements and incentives for a price of either real-cash, "in-game currency earned" or "time consumed" while playing the game. The players' purchases affect their game experience in some way. Using a combination of these is a good approach.

- In-game currency — the gamer exchanges cash for internal game currency. Some games allow the option to earn in-game currency and withdraw as cash; I cringe whenever I see this because game security should be paramount. A perfect example of this is action was "**Whirled**"<sup>26</sup> by **Three Rings Design**.<sup>27</sup>
- Consumables — a cash-machine! Instead of a one-time purchase and the player owns that item forever. Have it deplete after x number of usage — to a battery or a can of Pepsi.

<sup>23</sup><https://sweetpricing.com/blog/2016/07/7-app-monetization-stats/>

<sup>24</sup><https://sweetpricing.com/blog/2017/01/promotional-pricing/>

<sup>25</sup><https://www.joshmorony.com/how-to-create-ios-in-app-purchases-with-phonegap-build/>

<sup>26</sup><https://en.wikipedia.org/wiki/Whirled>

<sup>27</sup>[https://en.wikipedia.org/wiki/Three\\_Rings\\_Design](https://en.wikipedia.org/wiki/Three_Rings_Design)

- Seasonal Items — limit the items availability to holidays and seasons.
- Variable rewards based on scores, time duration played or achievements.
- Unlock-able Avatar — allowing gamers to swap-out different avatars or different “dress-up” features.
- Skills upgrades or enhancements — gamer pay for reduced time for upgrades or skill technology trees.
- Extra lives or turns — players pay for these addition beyond the “free-mium” model.
- Double Experience based on cash spent previously or as a single transaction upgrade.
- Limited items in stock — similar to Season items yet these have an inventory and stock level. Once “sold-out” these items are gone forever.
- Offer Walls — Allowing gamers to sell their collected “goods”.



**Exercise:** Read how Josh Morony builds his *IGP Phaser Stores*.<sup>28</sup>

## Advertising

Ads have been on the WWW since its beginning. It's a great method to generate cash without burdening your clients. Mobile ads are usually the first thing that comes to mind, but there are other ways:

- Banner ads have annoyed games for decades; up to the point that ad blockers are quite common. In fact, IAB claims that 1/4 of your game traffic have **active “ad blockers”**<sup>29</sup> and up to **40% of all laptops**<sup>30</sup> use ad blockers.
- Full screen and Interstitial ads — these fade into the background after consistent use and usually have a higher payout ration. The strategy is to place the ads just before submit a highscore or saving the games progress. Another recommendation is on the games terminations and return to the main menu.

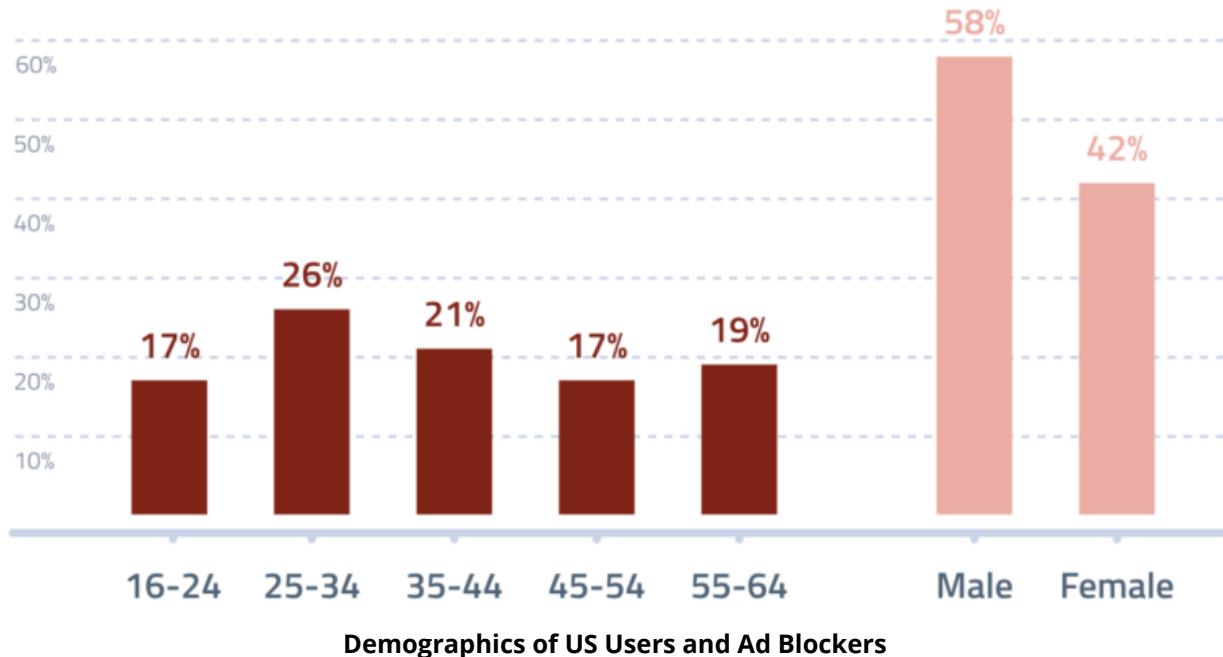
<sup>28</sup><https://www.joshmorony.com/creating-a-shop-with-purchasable-items-in-a-phaser-game/>

<sup>29</sup><https://www.adweek.com/digital/iab-study-says-26-desktop-users-turn-ad-blockers-172665/>

<sup>30</sup><https://marketingland.com/survey-shows-us-ad-blocking-usage-40-percent-laptops-15-percent-mobile-216324>

## DEMOGRAPHICS OF AD-BLOCKER USERS

% of US Ad-Blocker Users who are...



Review the ***current trends of ad blocker usage.***<sup>31</sup>

## Partnerships & Sponsors

An overlooked and underutilized method of monetizing a game, especially by indie, are partnerships and sponsorship. In some cases, these will pay before the game is finished or even before the game production starts. It is “cheaper” to contract an indie developer than maintain a full-time employee. Partnerships take many forms:

- Advertisement games. You could make an entertainment that are advertisements. I have several examples from Purina Dog Chow on my arcade network.
- Sponsored items or characters — for example, you’ve create a horse riding game and contact Levi Jeans to sell space and characters using their products.
- Licensing — you can license your games to web portals. This was the concept behind Mochi Media. You might like to see how I ***license and create affiliates.***<sup>32</sup>

<sup>31</sup><https://www.statista.com/statistics/804008/ad-blocking-reach-usage-us/>

<sup>32</sup><http://stephen-gose.com/affiliates-program/>

- Free with Purchase — Amazon allows author to offer free ebooks with the purchase of the hardback copy. You might do the same with your games.
- Bundling — link up with your fellow indie developers and offer a package set of games.
- White Labeling — similar to licensing but white labeling allows you to make profits up front from partners by inserting their logos into your games as non-exclusive.

## Retail

The old fashion store model in which gamers are paying for your game on CD or from Steam. Here's some other considerations:

- Game portals — uploading your game to various gaming sites.
- Pay to remove ads. The “free” version has ads which are removed if the gamer compensates for a low fee paid one or on a subscription basis. This is a good option for popular or high “replay” games.
- Premium Versions. Gamers are given access to special incentives or game features.
- Additional paid & download content (DLC) — is very common in the PC game industry. This is typically used after the game is launched.
- Episodic paid content — to DLC, except you are selling your game in “chunks. For example, the gamer pays \$5 for 6 additional story lines or final endings. This is the best choice for adventure and romance games.
- Sell your source code. Many indies make unique passion projects that tend to be very niche oriented. Source code for those segments don't usually sell well. However, if you made an “Action” or “Casual Mobile” game, you could consider selling the source code with or without royalties. The **“Envato Market”**<sup>33</sup> has been doing so for years.
- Subscriptions. The MMoG use this method. If folks love your game, and they play it consistently, then subscriptions are the way to make profits. You could also mix subscriptions with “free-mium” in which the game is free to play and subscription provide enhancements and incentives.
- Crowd funding — is the little brother to capital investment backings, penny stocks and **“pink sheets”**.<sup>34</sup> Read more about **Pink Sheets: Do's and Don'ts And My Key Profitable Secrets Revealed**<sup>35</sup>

<sup>33</sup><https://codecanyon.net/search/html%20mobile%20action%20game?ref=PBMCube&referrer=search&sort=sales&utf8=%E2%9C%93&view=grid>

<sup>34</sup><https://www.investopedia.com/articles/fundamental-analysis/08/pink-sheets-ottcb.asp>

<sup>35</sup><https://www.timothysykes.com/blog/pink-sheets-work/>

## Billing

How you collect payments should never come after a game launch. It should be one of the first considerations as you develop a piece of entertainment. Just as your games have phases, so does the payment process from “first impression” to “satisfied customer”. Your plans should address:

Multiple payment methods for local, regional and world-wide. Collection might include Vouchers, Gift certificates, Bit-coin, Paypal, Apple Pay, Google Wallet just to mention a few. The easier you make the payment process, the more likely someone will pay for your offering. For example “Carrier Billing” is even easier than logging into accounts or adding the gamers information. It is used in Brazil where folks don’t have credit cards.

## Data

This method is often completely ignored by most indie developers — Selling, Renting or using game statistical data collected from gaming sessions. Always tell your customers your terms of service and policies.

## Player Interactions

The ways you make money off of player interactions are different in that you are providing a means to players to gift each other. You may let players trade and sell avatar artwork or skills. Allow players to place wagers or bets; always check with local regulations. Germany is quite strict on games of skill versus game of chance. You could allow players to generate content for sell such as a comic book. This has been one of favorites for years.

## Paraphernalia Merchandising

**Paraphernalia**<sup>36</sup> are things to sell as supporting merchandise — for example “stickers”, and stamps, coffee/tea cups, T-shirts, music theme CDs, wall paper artwork, key chains, charm bracelets, etc.

---

<sup>36</sup><https://www.merriam-webster.com/dictionary/paraphernalia>

## 14.6 Chapter Reference

1. MDN [https://developer.mozilla.org/en-US/docs/Games/Publishing\\_games](https://developer.mozilla.org/en-US/docs/Games/Publishing_games)
2. Apple Game Center <https://developer.apple.com/game-center/>
3. Google Play Console <https://developer.android.com/distribute/console/>
4. Google Play Developers <https://developer.android.com/distribute/>
5. Amazon Lumberyard <https://aws.amazon.com/lumberyard/>
6. Amazon Game Tech <https://aws.amazon.com/gametech/>
7. Amazon Game Studios <https://www.amazongames.com/>
8. ***"Building JavaScript games: For phones, tablets, and desktop."***<sup>37</sup> by Egges, A. (2014).

---

<sup>37</sup><https://amzn.to/2ANMljK>

# Appendix

Excellent! You completed this workbook and constructed your game. Still hungry for more? Take some time and review the following resources.

Available Bonus Content as 67-pages of external downloads.

- **Appendix: Building HTML5 Web Page (3-pages)<sup>38</sup>**
- **Appendix: Distribution Channels (2-pages)<sup>39</sup>**
- **Appendix: Game Design Considerations (1-page)<sup>40</sup>**
- **Appendix: Game Design Overview (3-pages)<sup>41</sup>**
- **Appendix: Game Resources and References (17-pages)<sup>42</sup>**
- **Appendix: JS Coding Styles (7-pages)<sup>43</sup>**
- **Appendix: Networking (9-pages)<sup>44</sup>**
- **Appendix: Phaser 3 Resources (2-pages)<sup>45</sup>**
- **Appendix: Phaser III Plugins (1-pages)<sup>46</sup>**
- **Appendix: Project Management Analysis (13-pages)<sup>47</sup>**
- **Appendix: Security (5-pages)<sup>48</sup>**
- **Appendix: US Business Start-ups (4-pages)<sup>49</sup>**

---

<sup>38</sup><http://makingbrowergames.com/book/Appendix-buildHTML5webPage.pdf>

<sup>39</sup><http://makingbrowergames.com/book/Appendix-Channels.pdf>

<sup>40</sup><http://makingbrowergames.com/book/Appendix-GD.pdf>

<sup>41</sup><http://makingbrowergames.com/book/Appendix-GameDesignOverview.pdf>

<sup>42</sup><http://makingbrowergames.com/book/Appendix-Game.pdf>

<sup>43</sup><http://makingbrowergames.com/book/Appendix-CodingStyles.pdf>

<sup>44</sup><http://makingbrowergames.com/book/Appendix-Networking.pdf>

<sup>45</sup><http://makingbrowergames.com/book/Appendix-Phaser3.pdf>

<sup>46</sup><http://makingbrowergames.com/book/Appendix-PhaserIIIPlugins.pdf>

<sup>47</sup><http://makingbrowergames.com/book/Appendix-PM.pdf>

<sup>48</sup><http://makingbrowergames.com/book/Appendix-Security.pdf>

<sup>49</sup><http://makingbrowergames.com/book/Appendix-USBusiness.pdf>