Developing an

HTML5 BRICK-BREAKER

game with

PHASER

Jorge Palacios

Developing an HTML5 Brick-breaker Game With Phaser

Jorge Palacios

This book is for sale at http://leanpub.com/brick-breaker-phaser

This version was published on 2015-05-31

This book is dedicated to my parents, Betty and Elieser. They have given me the place and tools to develop my career as a computer scientist and game developer, leveraging some responsibilities so I could focus my efforts on the things that I love.

There's also a special dedication to my mom. She's always believed in me and my game-development goals despite the odds. I love you mom, thanks for the support and kind words during the storms.

Contents

Introduction

Creating a Brick-breaker Game With Phaser is the result of teaching HTML5 game development by example at a local institute, focused on web technologies with a philosophy of *teaching concepts, methodologies and standards* in a practical way.

This book is intended to be a hands-on introduction to HTML5 game development using the Phaser framework by developing a *complete* game with well-known mechanics. It is bundled with all the necessary assets, so we can focus on the code and not finding extra resources or developing "programmer artwork".

What you will learn

By the end of the book, you'll have learned to:

- Set up an organized file structure for developing games for the web
- Create a blank game and test everything runs OK
- Import images and sounds
- Show and move sprites on the screen
- Manage a group of sprites
- Play sound effects and background music
- Show text on the screen with custom web fonts
- Handle touch and keyboard input
- Detect collisions and use the Arcade physics system
- How to start using the particle system
- Manage game states
- Create a "loading" screen
- Optimize the game for mobile devices

You can play a version of this game on itch.io, where you can also find the Android version.

Target audience

It is aimed to programmers with some experience with JavaScript and want to learn by doing (*and finishing*). It is not a guide to JavaScript, but each step is explained and the source code is well-commented so you can get it right away and keep focusing on the big picture.

Development tools

The Phaser web site has a well-documented section on how to get started. However, I invite you to test and develop the code from this book using the Brackets editor. The code was developed and tested on it, and works well on Windows, Linux and Mac. Besides, testing the examples is far too easy:

1. Install Brackets
2. Make sure you have Chrome / Chromium browser installed
3. In the menu bar, go to ***File -> Open Folder...***
4. Locate a directory with an `index.html` file. For example: `brick-breaker-phaser/01core_-mechanics/01paddle/`
5. Select it
6. In the menu bar, go to ***File -> Live Preview***

There are other editors and tools, but the coding-testing pipeline on Brackets is very straightforward if you're new to web development.

Acknowledgements

The following people helped in the development of this book or the game associated with it:

- Richard Davey - creator of Phaser
- Christian Chomiak - editing and title page
- Kenney - artwork and some sound effects
- Joe Powell, courtesy of Freesound - background music (Electric Air)
- Sergio Marin, and the whole team at Escuela Web, for giving me a place to unite two of my passions; game development and teaching
- Luis Miguel Delgado - content review
- Ángel León - technical review

0 Project setup

Introduction

This chapter is focused on organizing our project to keep a tidy file structure. It will allow us to get a good start, and test the installation of Phaser. It is worth mentioning that we'll be using Phaser v2.3, available here for download.

0.1 File system

1. Create the following file structure: an empty `index.html` file and four additional directories; `css`, `img`, `js`, and `snd`, respectively. As we can see, it's a standard file structure for web development. As games grow, teams tend to specialize the file structure but right now it's not our case:

```
game
|   index.html
+---css
+---img
+---js
\---snd
```

2. Add the phaser `.js` file in the `js/` directory. Here we will store all of our scripting files:

```
+---js
|       phaser.min.js
```

3. Modify the `index.html` with the following code. This declares the type of document, its character encoding, includes the `phaser.js` file in the html, and creates a place for the game to be rendered:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>html5 Breaker game</title>
  <script type="text/javascript" src="js/phaser.min.js"></script>
</head>
<body>
  <div id="phaser_game"></div>
</body>
</html>
```

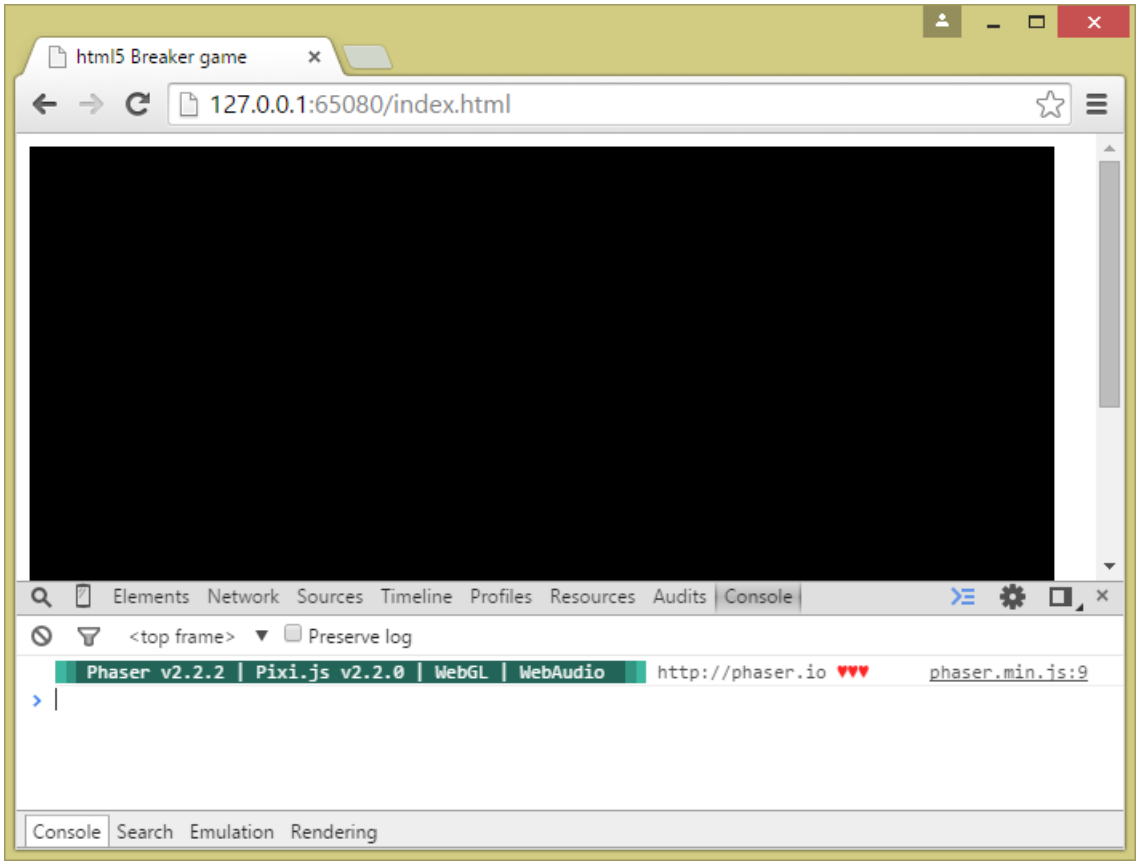
4. Create a file named `main.js` in the corresponding directory. This file will store the initialization code of our game:

```
// game's global variable
var game;
// after everything is loaded, then
window.onload = function () {
  // initialize the game object
  game = new Phaser.Game(640, 400, Phaser.AUTO, 'phaser_game');
};
```

5. Now include the previous file into the html file with the proper `<script>` tag. It must be included *below* the phaser include:

```
<script type="text/javascript" src="js/phaser.min.js"></script>
<script type="text/javascript" src="js/main.js"></script>
```

6. Finally, test the game by running it in the browser. If we open the browser's JavaScript console, we can read a message from the Phaser API stating that it is initialized and everything runs smoothly.



Phaser output message

0.2 Empty game

Now that we know how to set up an empty Phaser project, I can tell you that inside the downloadable content there is a directory called `00empty_game` with the previous code, *plus* all the images and sounds that we will use to build our game.

In case you're getting the book from Amazon, you can download the at: http://jorge.palacios.co/brick-breaker-book-resources/

In the following chapters we will build our game, starting from the core mechanics.

00_emptygame

```
00empty_game
|   index.html
|
+---css
|       styles.css
|
+---img
|       ball.png
|       bg_black.png
|       bg_blue.png
|       brick_green.png
|       brick_purple.png
|       brick_red.png
|       brick_yellow.png
|       logo_game.png
|       logo_icon_square.png
|       paddle.png
|       progress_full.png
|       progress_void.png
|       star.png
|
+---js
|       main.js
|       phaser.map
|       phaser.min.js
|
\---snd
        bgm_electric_air.ogg
        fx_firework.ogg
        fx_hit_brick.wav
        fx_hit_paddle.wav
        fx_lose.ogg
        fx_lose_life.ogg
        fx_win.ogg
```

1 Developing mechanics

Introduction

This is the base chapter where most of the magic will happen, and that's because gameplay is the most important thing in our game (at least in the beginning). We will learn how to build our core mechanics progressively, starting with the paddle and user input, and finishing with collisions and high-level rules.

Also, we'll learn the basics of handling game states but only for the purpose of coding the gameplay screen. How to handle state transitions will be covered in the following chapter.

1.1 Paddle

Programming the paddle will teach us how to load images and get the user input, simple mouse input (like its position), and check whether a key is pressed.

1. Create the `state_main.js` file in the `/js` directory. This file will hold the code for the state that handles the core mechanics:

```
var StateMain = {
  // TODO
  // internal functions
};
```

2. Define the `preload` function. Here we will load all the assets to be used in the game. Right now it's just the paddle sprite, but that will change in the following sections:

```
preload: function () {
  game.load.image('imgPaddle', 'img/paddle.png');
},
```

3. Define the `create` function. This functions runs once and is used for setting up the sprite:

```
create: function () {

},
```

4. Define the member variables for handling paddle horizontal velocity and previous mouse position. We declare the movement to be 500px/sec and a member variable for storing the mouse position in the previous frame:

```
this.paddleVelX = 500 / 1000;
this.prevX = game.input.x;
```

5. Add the paddle sprite on the screen by using the game object's factory. Also, set its anchor point to the bottom-center and finally create a custom member variable for storing the sprite's half size for future reference:

```
this.paddle = game.add.sprite(0, 0, 'imgPaddle');
this.paddle.anchor.setTo(0.5, 1);
this.paddleHalf = this.paddle.width / 2;
```

6. Call the function for restarting the paddle into its original position (not coded yet):

```
this.resetPaddle();
```

7. Define the `update` function. It keeps running as long as the browser tab is focused, and here we'll code most of the loop-based and time-based game logic:

```
update: function () {

},
```

8. Declare and assign the variables for getting keyboard input from the arrow keys:

```
var isLeftDown = game.input.keyboard.isDown(Phaser.Keyboard.LEFT);
var isRightDown = game.input.keyboard.isDown(Phaser.Keyboard.RIGHT);
```

9. Assign the paddle position according to the mouse or keyboard input. If the mouse hasn't moved (this is why we declared `prevX`), then we check for keyboard input. It's important to mention that in this game the arrow keys are mutually exclusive:

```
if (this.prevX != game.input.x) {
  this.paddle.x = game.input.x;
} else if (isRightDown && !isLeftDown) {
  this.paddle.x += this.paddleVelX * game.time.physicsElapsedMS;
} else if (isLeftDown && !isRightDown) {
  this.paddle.x -= this.paddleVelX * game.time.physicsElapsedMS;
}
```

10. Assign current mouse position as the previous:

```
this.prevX = game.input.x;
```

11. Limit the movement of the paddle to the screen bounds:

```
if (this.paddle.x - this.paddleHalf < 0)
  this.paddle.x = 0 + this.paddleHalf;
if (this.paddle.x + this.paddleHalf > game.world.width)
  this.paddle.x = game.world.width - this.paddleHalf;
```

12. Define the `resetPaddle` function:

```
resetPaddle: function () {

}
```

13. Set the paddle's anchor point and place the paddle a little higher than the bottom of the screen:

```
this.paddle.x = game.world.centerX;
this.paddle.y = game.world.height - this.paddle.height;
```

It's time to link the `state_main.js` file to `index.html`:

```
<script type="text/javascript" src="js/phaser.min.js"></script>
```

This is how our `index.html` file should look like:

index.html

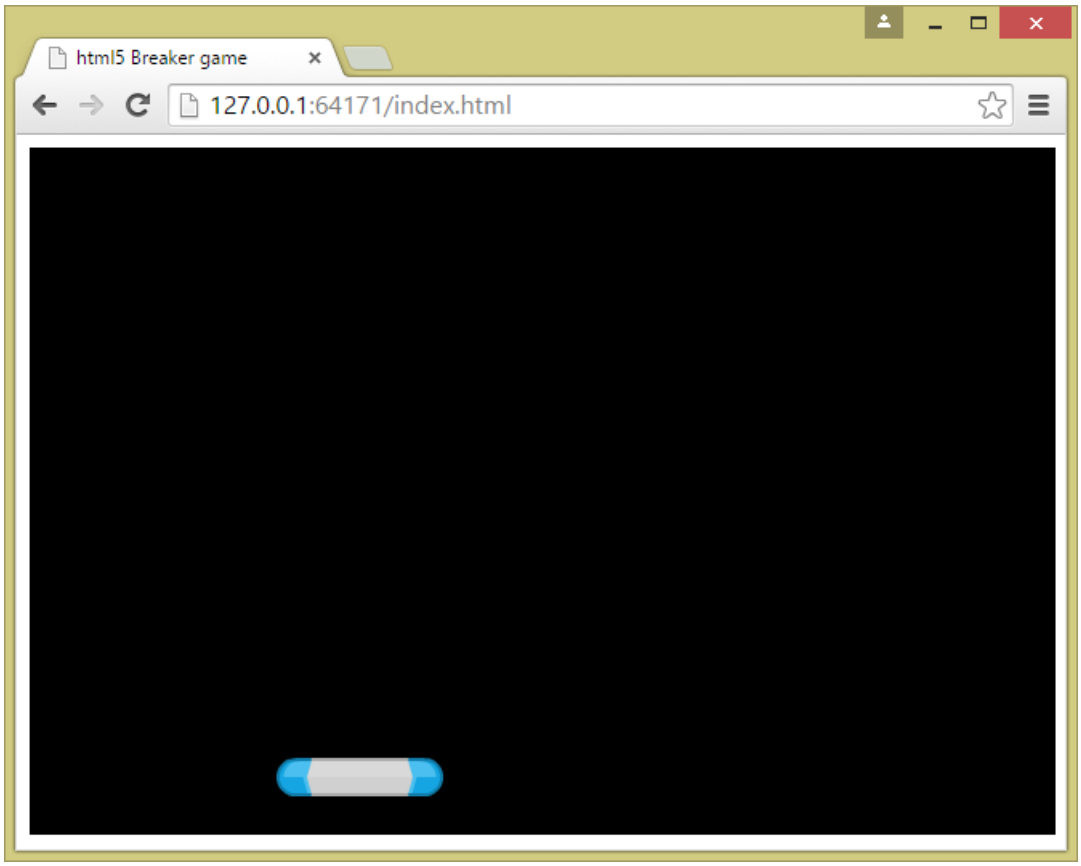
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>html5 Breaker game</title>
  <script type="text/javascript" src="js/phaser.min.js"></script>
  <script type="text/javascript" src="js/state_main.js"></script>
  <script type="text/javascript" src="js/main.js"></script>
</head>
<body>
  <div id="phaser_game"></div>
</body>
</html>
```

Finally, add `StateMain` to the game object and start the game with it:

main.js

```
var game;
window.onload = function () {
  game = new Phaser.Game(640, 400, Phaser.AUTO, "phaser_game");
  game.state.add("StateMain", StateMain);
  game.state.start("StateMain");
};
```

By the end of this section, we should have a nice paddle moving along the X axis with the keyboard's arrow keys or following the mouse position.



Paddle

Info

Remember to add a coma after each function's definition (but the last), and a semi-colon after each state's definition.

Not doing it will produce **errors** in the code. Please refer to the example below.

```
var You = {
  theFunction: function () {
    // your code here
  },
  other: function () {
    // your code here
  }
};
```