Week 17_coinGame_info

Build a sample Game

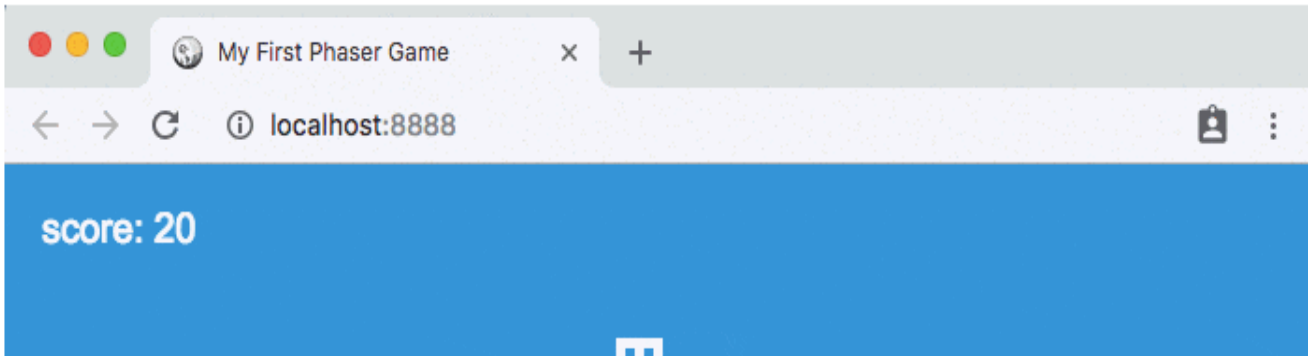
(see the whole demo on www.lesscake.com
Phaser Demo
[Learn to make HTML5 games with Phaser 3]

Learn to make HTML5 games with Phaser 3

Phaser is a popular Javascript framework to build HTML5 games that run in the browser. It is free, powerful, simple to use, and actively maintained. I believe it's the perfect framework to start making games.

In this tutorial we will build a game where the player is getting points every time he touches a yellow coin. That's super basic, but it's also a great example to learn how Phaser works.

Here are all the topics we will cover: scenes, sprites, texts, collisions, and tweens!



Setup

The setup of our project is going to be straightforward. We create a new directory called "first-game", and inside it we add:

- An empty file called "index.html", that will display our game.
- An empty file called "game.js", that will have our Javascript code.
- A directory called "assets" containing 2 sprites, that can be downloaded here.

Display the game

Let's start with the index.html file. Here is the code that will display our game in a webpage.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>My First Phaser Game</title>
<script src="https://cdn.jsdelivr.net/npm/phaser@
3.14.0/dist/phaser.min.js"></script>
<script src="game.js"></script>
```

```
</head>
<body>
<h1>My First Phaser Game</h1>
<div id="game"></div>
<p>Use the arrow keys to move around and collect the coins.</p>
</body>
</html>
```

This is pretty simple, but there are 2 important things to notice:

- We are loading 2 scripts: phaser.min.js (the Phaser framework version 3.14.0) and game.js (our file).
- We added a element, that's where the game will appear.

Now we will focus our attention on the game.js file.

Create a scene

Every Phaser project is made out of scenes. In our case we are going to only have one that will contain the whole game.

To create our scene we use a class with 3 methods. It's very important to understand what each of these methods is doing, since they are key to how

Phaser works.

```
// Create our only scene called mainScene, in the game.js file
class mainScene {
  // The 3 methods currenlty empty
 preload() {
    // This method is called once at the beginning
    // It will load all the assets, like sprites and sounds
  }
  create() {
    // This method is called once, just after preload()
    // It will initialize our scene, like the positions of the sprites
  }
  update() {
    // This method is called 60 times per second after create()
    // It will handle all the game's logic, like movements
  }
}
```

To summarize, these 3 methods will be called in this order: `preload()` → `create()` → `update()` → `update()` → `update()` → etc.

Start the game

Now that our scene is created, it's time to start the game. For that we call `Phaser.Game()` at the end of the Javascript file. There are a lot of optional

parameters available, but here are the main ones.

```
new Phaser.Game({
  width: 700, // Width of the game in pixels
  height: 400, // Height of the game in pixels
  backgroundColor: '#3498db', // The background color (blue)
  scene: mainScene, // The name of the scene we created
  physics: { default: 'arcade' }, // The physics engine to use
  parent: 'game', // Create the game inside the <div
id="game">
});
```

Our job in the rest of this tutorial will be to fill the `preload()`, `create()`, and `update()` methods to build our little project.

Test the game

It's a good time to test what we did so far. However, directly opening the index.html file in a web browser is not enough, we need to use a local webserver for Phaser to work properly.

There are multiple ways to setup a local webserver. If you already know how to do that, then you can skip this part. Otherwise I will show you a very simple technique to start one.

Download and install Brackets, it's a free code

Download and install Brackets, it's a free code editor that runs on Windows, Mac and Linux. Then use it to open the directory "first-game" that we created earlier, and click on the small bolt icon on the top right corner of the app. That will automatically create a local webserver, and open a browser window with the game in it.

At this point you should only see the blue background. Don't worry, it will become more interesting soon!

Add the player

The first element to add to the game is the player (the white monster). That can be achieved in 2 short steps.

Step 1, load the sprite. Since we are loading an asset, we have to do it in the `preload()` method.

```
// Parameters: name of the sprite, path of the image
this.load.image('player', 'assets/player.png');
```

Step 2, show the sprite on the screen. This is part of the initialization of the scene, so we do that in the `create()` method.

```
// Parameters: x position, y position, name of the sprite
this.player = this.physics.add.sprite(100, 100, 'player');
```

There are 3 interesting things to mention here:

- The player is stored in the `this.player` variable, which is accessible in all methods of the class.
- We typed `this.physics` to create the player. It means that we are using the physics engine on the sprite, and that will make handling collisions super easy later on.
- If we set the player's position to `x = 0` and `y = 0`, then it would have been displayed in the top left corner of the screen.

Add the coin

Next, we will add a yellow coin that the player can catch. The process is going to be very similar

to what we did previously.

First we load the image of the coin in `preload()`.

```
this.load.image('coin', 'assets/coin.png');
```

Then we show the coin on the screen in `create()`:

```
this.coin = this.physics.add.sprite(300, 300, 'coin');
```

Add the score

The last element that is missing from our game is a score. Since there are no assets to load, we put this code directly in the `create()` method.

```
// Store the score in a variable, initialized at 0
this.score = 0;
// The style of the text
// A lot of options are available, these are the most important ones
let style = { font: '20px Arial', fill: '#fff' };

// Display the score in the top left corner
// Parameters: x position, y position, text, style
this.scoreText = this.add.text(20, 20, 'score: ' + this.score, style);
```

Now all the elements of the game are here, but the player is stuck on the screen and we don't handle collisions.

Move the player

In order to move the player around, we have to tell Phaser that we want the arrow keys as inputs. For that, we type this line in the `create()` method.

```
this.arrow = this.input.keyboard.createCursorKeys();
```

With this new variable we can check which arrow key is pressed, and change the player's position accordingly. To be able to move the player at any time, we put the following code in the `update()` method that is called 60 times per second.

```
// Handle horizontal movements
if (this.arrow.right.isDown) {
  // If the right arrow is pressed, move to the right
  this.player.x += 3;
} else if (this.arrow.left.isDown) {
  // If the left arrow is pressed, move to the left
  this.player.x -= 3;
}
// Do the same for vertical movements
if (this.arrow.down.isDown) {
  this.player.y += 3;
} else if (this.arrow.up.isDown) {
  this.player.y -= 3;
}
```

Handle collisions

Every time the player touches the coin, we would like to move the coin to a random position and increment the score by 10. So let's code all of that in a new method called `hit()`.

```
hit() {
  // Change the position x and y of the coin randomly
  this.coin.x = Phaser.Math.Between(100, 600);
  this.coin.y = Phaser.Math.Between(100, 300);
// Increment the score by 10
  this.score += 10;

// Display the updated score on the screen
  this.scoreText.setText('score: ' + this.score);
}
```

We need to call `hit()` when the player and the coin overlap. But how do we know when that happens? It's very simple with Phaser's built in physics engine, we just put this code at the beginning of `update()`.

```
// If the player is overlapping with the coin
if (this.physics.overlap(this.player, this.coin)) {
  // Call the new hit() method
  this.hit();
}
```

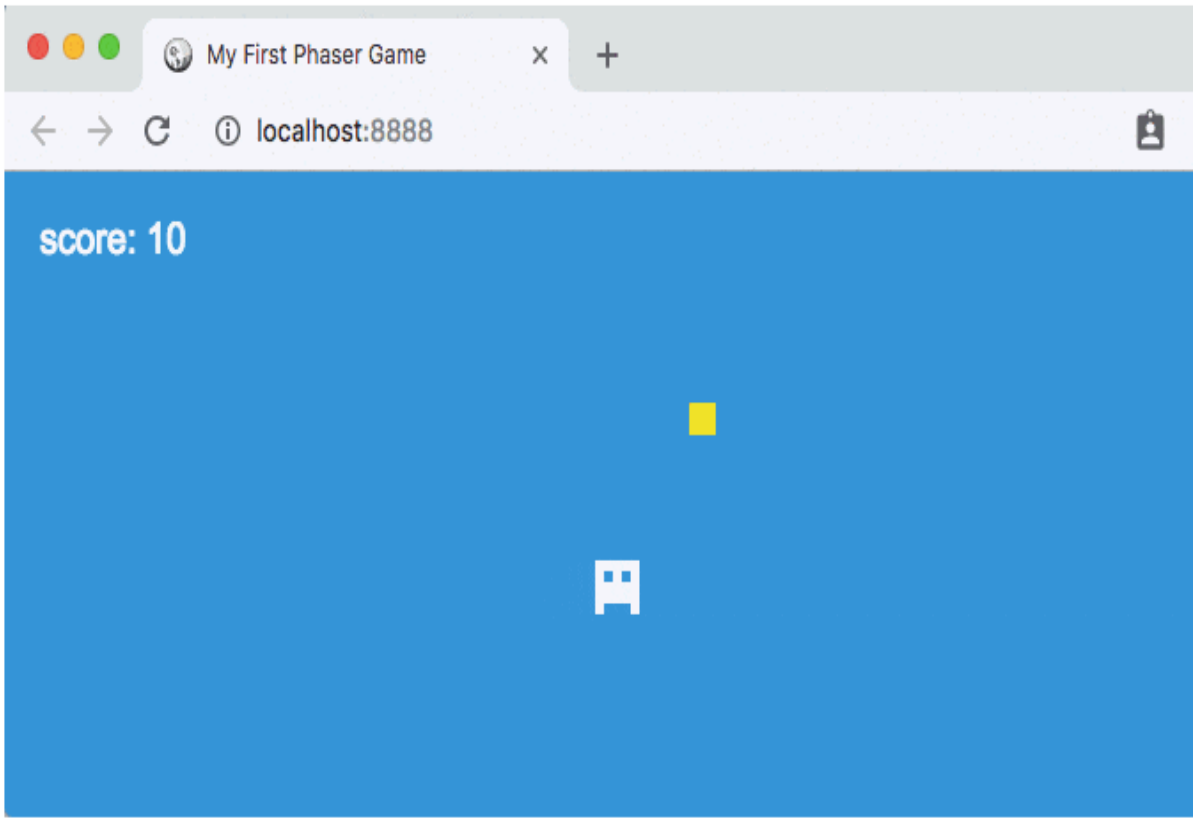
Improve collisions

Right now when grabbing a coin, the only thing that changes is the score. That's a little underwhelming, especially since it's the main goal of the game.

To improve this, let's make the player temporary grow when he takes a coin. This can be done with a tween in the `hit()` method.

```
// Create a new tween
this.tweens.add({
  targets: this.player, // on the player
  duration: 200, // for 200ms
  scaleX: 1.2, // that scale vertically by 20%
  scaleY: 1.2, // and scale horizontally by 20%
  yoyo: true, // at the end, go back to original scale
});
```

And we finished! Here's the final result.



Conclusion

In less than 60 lines of code we've built a game that is playable in a browser. Pretty cool! The full source code of the project is available on GitHub.

If you have any questions or feedback: thomas@lesscake.com :-)