Making a HTML 5 Game Using Phaser

Workshop Made by Michael Rechenberg

So you want to make a HTML 5 game? Learn more about Javascript and the awesome Phaser library? Just bored and went to a random workshop? No matter the reason, by the end of this workshop you’ll understand the tools necessary for if you want to go on afterwards and make a more advanced game.

**A Bit of Background on Web Development**

Since Phaser creates games designed for use in web browsers for mobile and desktop, you’ll need to be familiar with a few of the core languages used in the web. [Talk about using console/inspect element to view a web page]

1. HTML

* HTML stands for Hypertext Markup Language and it designates the structure of webpages. For this workshop we won’t use a lot of HTML, but we need a little of it to display our game. If you would like to learn more about HTML, W3Schools.com has a good tutorial on basic HTML.

1. CSS

* CSS stands for Cascading Style Sheets and it designates the aesthetics/style of webpages. We won’t use CSS in this workshop, but if are designing a game to be hosted on your website and you want your website to look nice, CSS will do that.

1. Javascript

* Javascript is the language that we will spend the majority of the workshop using. It is a programming language whereas HTML and CSS are not. Javascript is not the only programming language for the web (PHP), but Javascript is extremely popular with great libraries and frameworks (jQuery, Node.js, and Phaser!)

**Setup**

Phaser needs a web server to work with, so we will be working with Cloud9, a nice cloud-based development environment that requires little work to setup. If you would like to work on a game after this workshop, you can use your own machine as a webserver (‘localhost’)…the specifics of setting up a localhost depends on your operating system so if you’d like to set one up Google how to set up a localhost for your specific operating system.

To get set up with Cloud9:

1. Go to <https://c9.io/>
2. Make an account (or sign in with GitHub)
3. Create a new workspace by clicking the big plus symbol in the middle of the page. Name the workspace whatever you’d like and give it a brief description. There will be an option called “Clone from Git or Mercurial URL”…copy-paste this link into that text field

<https://github.com/MichaelRechenberg/PhaserWorkshop.git>

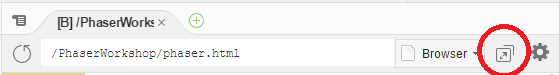
1. If everything goes well, you’ll see a file called README.md that is already open. If it says “This repo contains all of the example code for the CS Sail 2016 Workshop on Phaser” and you see a bunch of files ending in .js then everything worked. Raise your hand if you have any questions.

This is the structure of the repository:

* phaser.min.js: Contains the Phaser library in minimized form (uses the least amount of space possible i.e. all unnecessary whitespace is removed). You should not modify this file.
* phaser.html: This is the HTML that the webserver will serve to the client. It contains a div element that the Phaser game we make will be inserted into.
* Various Javascript files: These files contain snippets of code that we will go over in the workshop. We will make our own file in which we will type in the code.
* Assets folder: this will contain any assets we need (images, sound, etc.)

Viewing Your Work: Viewing the Game in a Web Page

Throughout the workshop we will work on a Javascript file and then view the result of the code in a web page. In order to do this we’ll do a “live-preview” of the HTML page that links to our Javascript code.

* Open the phaser.html page (or whatever HTML page you create if you wish to work on games after this workshop)
  + Look at the 2nd <script> tag in the HTML page. In its “src” attribute it should equal the name of a file we are working on (main.js, main01.js, main02.js, etc.). Make sure that whatever Javascript file you want to view/test is included in this src attribute…otherwise you’ll be viewing the result of different code.
* While the HTML source code is open in a tab, click the “Preview” button on the top-center of the page
  + Click the “Live Preview” Option
  + To open the preview in a new tab within your actual browser click this button:

Debugging (Optional)

Being able to debug your game is essential: lucky for you Chrome comes with its own console for print statements and debugger that you can use to step through your code.

* While your HTML page is open right-click anywhere on the page that is not within the 800x600 rectangle that the game resides. Click “Inspect”. Then on the top toolbar click “Console.” This will display anything printed out by console.log() statements.
* To debug your code line by line, click “Sources” on the top toolbar. Click your .js file and you can insert breakpoints, step into, and step over functions from here.
  + You should never have to “step into” phaser.min.js. It’s impossible for a ‘sane’ human to read, so don’t even bother.

**Creating the Phaser.Game Object (main01.js)**

Now with all that boring setup out of the way let’s get down to coding!

First, create a new file called “workshop.js”. You do this by first clicking on the PhaserWorkshop directory and clicking File > New File. Then to save the file click File > Save as and name the file “workshop.js”. You will type your code here (or if we are running short on time and you want to follow along, you can copy the corresponding main#.js file here).

Open up phaser.html and where it says src=”main.js” in the second <script> tag, replace “main.js” with “workshop.js”

Copy and paste the contents of main01.js into workshop.js

View the game in a webpage like how I described in the previous page. You should see a block box 800x600 pixels wide. Let me know if you’re having errors.

**Displaying Images (main02.js)**

Now we’re going to start displaying sprites to the screen.

First we have to load our assets (which for now is just an image). We do this on line 41 with a call to game.load.image(). The first argument is the key (‘sail’ for this example) by which we will refer to the image in later function calls and the second parameter is the name of the file (if the file is not in the same directory as the HTML, you must provide the relative path to the file…assets/images/CS\_Sail.png).

But just loading the asset does not display the image in the game. To actually display the image we will create it in the create() function with a call to game.add.sprite(). This function takes 3 arguments: the first argument is the x position of where you want to place the sprite, the second argument is the y position, and the third argument is the name of the key of the image we want (which is ‘sail’ for this example). This function also returns a reference by which you can manipulate the image later…we will store this reference in a variable named ‘sail’. To modify the image, simply invoke a Phaser.Sprite method on a reference to a sprite object, as we do on line 52 by calling the .anchor.setTo() function [explain what anchor does] on the variable sail.

If you preview the HTML page (did you remember to change the src attribute in the second <script> tag to “workshop.js”?) you’ll see a picture of CS Sail in the middle of the screen. Let me know if you have issues.

**Physics (main03.js)**

[load background and dude]

For line 44 we use game.load.spritesheet() instead of game.load.image()…I will go over why we do that later on in the workshop.

[declare player variable, add background]

To use physics we’ll first have to start the system. We do so by calling game.physics.startSystem(), passing in which physics system (Ninja, Arcade) we want to use.

[add player]

Now we want physics to work on the spritesheet that we’ve called the player, so we call game.physics.arcade.enable(), passing in the reference to the player spritesheet (variable player)

[bounce, gravity, collidWorldBounds]

These lines give the player some physics characteristics.

If you preview the HTML you’ll see the player fall downwards and bounce when he hits the bottom of the game. Let me know if you have any issues.

**Movement and Animation/SpriteSheet (main04.js)**

[Add movementControls and ground variables]

[Add ground, enable physics on ground, scale, enableBody, body.immovable]

Now we are going to go more into the concept of SpriteSheets. Go to the assets folder and open up dude.png. You’ll see that it is one image that contains different frames of the dude. When we loaded in the spritesheet earlier, the parameters we sent in split up the image into each of its frames. The spritesheet contains 9 frames, where 0-3 contain animations for running to the left, 5-8 contain images for running to the right, while frame 4 is for when the player is standing still.

So on lines 82 and 83 we create two animations, one for walking to the left and one for walking to the right using animations.add(). The first argument is key that we use to refer to a specific animation, the second parameter is an array containing the frames we want to use in our animation, the third argument is the frame rate (a larger number means that the animation will play faster), and the last argument is a Boolean value indicating if we want the animation to loop.

We want to have the animations play when the player moves left or right, but before we can do that we have to create our input. To do this we call game.input.keyboard.createCursorKeys(), which returns a reference to which we can access the up, down, left, and right arrows later.

Now we start to do work in the update() function, which is called every frame. First, we want to calculate if the player has collided with the ground, so line 92 does that by calling game.physics.arcade.collide(). The arguments given are the two sprites to test collisions against. This function is extremely powerful and we use it in an even better manner later.

[Just type out the giant if/else statement and jumping if statement]

These statements basically say if the right or left arrow is pressed, move the character the corresponding direction by adding velocity in that direction and playing the corresponding animation. If neither left or right is pressed, the dude will stop moving in the x-direction, all animations will stop, and the dude will face the screen.

The last if statement says if the up button is pressed AND the player is touching down, then jump in the air (without the second condition, the dude could ‘fly’ indefinitely like he had a jetpack)

**Groups and Shooting (main05.js)**

[add spaceBar and playerProjectiles variables]

[add projectiles image in preload]

Now we are going to work with Groups, another Phaser goodie. Groups allow you to refer to a bunch of objects with only one variable. It’s like if you are running a store and you sell apples but you want to increase the price of apples. You could have variable apple1, apple2, …apple1337 and so on and update their price, OR you could use a group and make the change in one line.

The group we are going to make contains the projectiles that the dude will shoot. First, we tell the Game that we want to make a Group by calling game.add.group() and storing its return value in playerProjectiles. Next, we modify some of the attributes of the playerProjectiles group. enableBody = true means that physics will apply to any sprite made from this group. The 2 calls to setAll sets 2 properties for all sprites made from the playerProjectiles group: checkWorldBounds means that the sprite will update its internal Boolean value indicating if it within the “World” i.e. what the player can see (this is not the full definition of a “World”, but it is sufficient for this workshop); and outOfBoundsKill, which means that if the sprite is out of bounds of the world the kill() method, which makes it so the sprite is not redrawn on when update() executes (saving time because drawing is a relatively expensive operation.

Next, we work on adding the spaceBar [addKey, addKeyCapture, onDown.add()]

[if timeElapsed statement]

Write shoot() function.

**Enemies, Looping, Overlapping Sprites (main06.js)**

**Text Objects(main.js)**

02: Adding images…go over coordinate system x->, y goes down; by default (0,0) in an image is the top-left corner, this can be changed with anchor.setTo([0-1],[0-1])

03: Adding dude (no animation yet), enabling physics