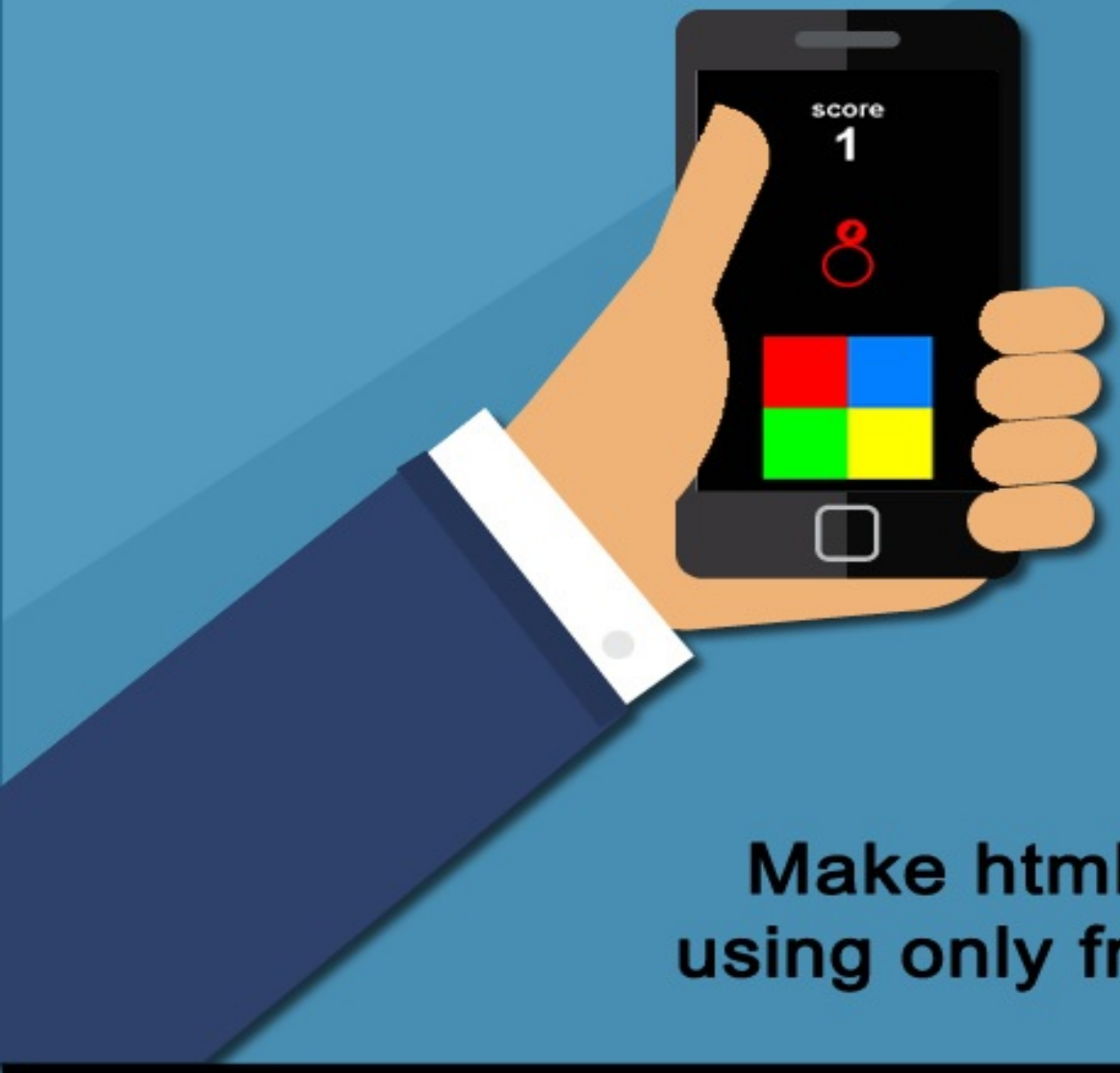
# Making Games With Phaser

## Color Zap




Make html5 games using only free tools!

by William Clarkson

# Contents

# CHAPTER ONE

## *Introduction*

### Introduction

When I was a kid about 13 or 14 years old, I had a series of computers. I started off with a ti-99/4a and moved on to a vic 20 and then to a Commodore 64. This was well before Internet, and these computers didn't even have hard drives. Most of the computing in those days were focused around word processing, spreadsheets, databases and games. Being a teenager I didn't have much interest in the first three, games, however I had a big interest in, and I had a subscription to a magazine called Compute! This would come in the mail every month and it would have printed code, that could be copied by typing from the magazine pages into your computer, this was well before copy and paste, and then the article would tell what the code meant and that's how I learn how to code. This was back in the 80's but a lot of the code I learned is still relative today.

Every month there would be several different games that I could type in and if something didn't work I had to go back over the code to see what I had done wrong and eventually I had a working game. At the time I could only save it on an audio cassette and often they were lost. Fortunately things are much more advanced now.

I went on to become a game programmer, and have written over 300 games in my career. I enjoy talking about game design and code, and sharing that with others.

I want to write a series of books including the one that you are reading now that focus on a single game that I will walk you through step-by-step and deconstruct the game as we are writing it and testing it each step of the way.

Some of the information such as setting up a local server, debugging tools, code editors and such will be included in every book, however each book will have a different game in it that will go over different techniques of game building.

Any of the books should give you enough building blocks to be able to create your own games. creating your own game for others to play and appreciate is one of the biggest Thrills that I have ever had in my life. I hope you will be able to have the same experience and I can't wait to see what games you will come up with on your own!

~William Clarkson

**Who this book is for**

This book is written for programmers who would like to learn how to make games.

To use this book, you will need to know a little javaScript.

Even basic javaScript will suffice. If you know variables, loops and if-then statements then you should be fine.

Every effort has been made to make sure the code functions as written in this book, however if you find a mistake, or have any questions please contact me through my website, http://www.williamclarkson.net

**About Phaser**

http://phaser.io/

Phaser is an html5 framework that is open source, developed by PhotoStorm. With it you can create games that will run on both desktop and mobile web browsers. It uses WebGL (web graphics library) if available or falls back to the HTML5 canvas if not available.

It has a lot of code that simplifies game development. I use it for most of my game development.

# CHAPTER TWO

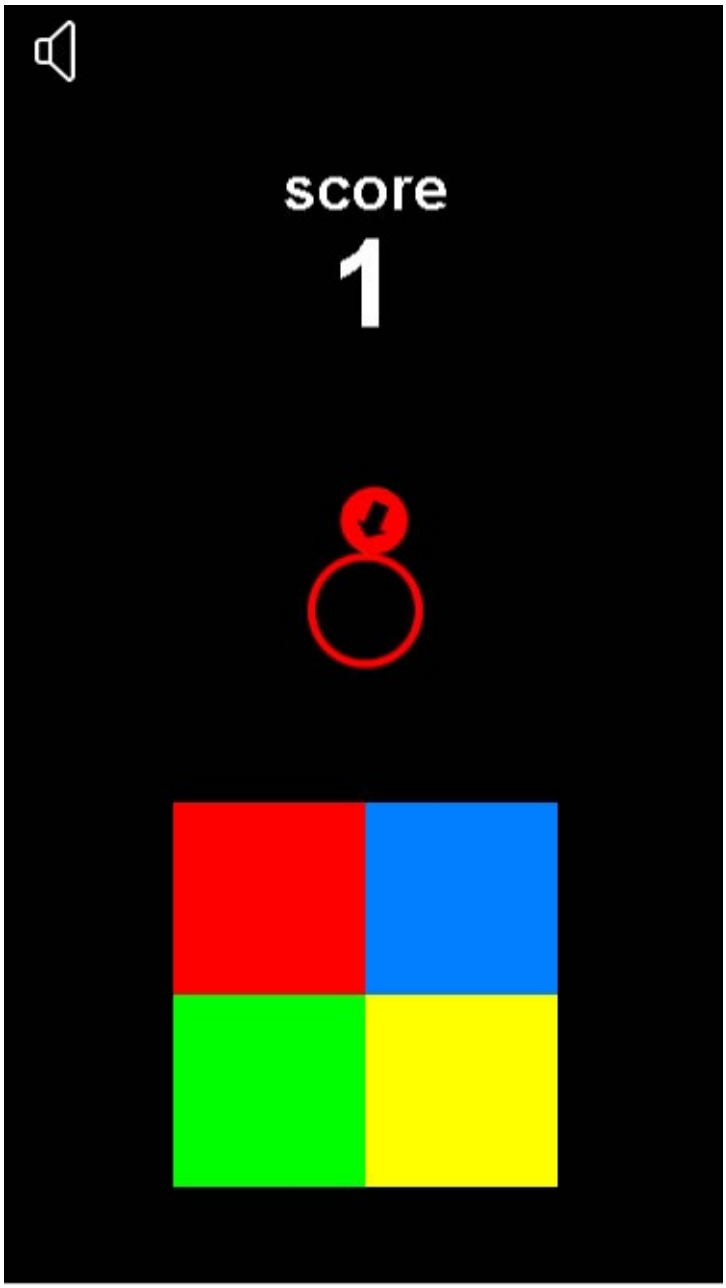
## *About the Game*

### Color Zap

A few months ago I saw a game that was inspiring a lot of clones. It was a very simple game that had dot that would change from black to white depending on whether you pressed the phone screen or not there was a ball that would always be rushing towards the dock and if the color matched you got a point and if it didn’t match the game was over I made this whole little bit more complicated by adding more colors , and that is how I came up with color zap.

### How To Play

To play this game you have to hold down one of the four blocks to change the ring color releasing the blocks will turn the ring back to the color white the ball will always be rushing towards the ring if the ball matches the color you got a point if it doesn’t match the game is over the ball gets faster as the game progresses.


score
1

# CHAPTER THREE

## *Setting Up*

### Source Files

All the resource files including code and images are available on my site

http://www.williamclarkson.net/colorZap

The game itself may be played on

http://www.happyplacegames.net

### Resources

This is an overview of the image and sound files used to make the game.

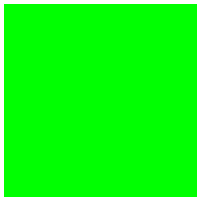
### Images

Folder:images/main/blocks

These are the blocks pressed to change the ring color

 yellow.png

 blue.png

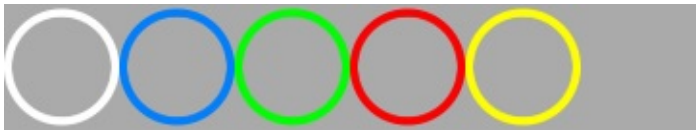
 green.png

 red.png

Folder:images/main

These are sprite sheets, which contain several images in a single file.
They have transparent backgrounds, but changed to gray here to be more visible.

 balls.png

 rings.png

Folder:images/title

gamelogo.png

Folder:images/ui


buttons.png

soundButtons.png

Folder:images


turn.png

**Sounds**

Folder:sounds

gameOver.mp3

points.mp3

**Tools required**

You only need a few things to get started, and all of them can be downloaded free!

You will need

1. A code editor
2. A web browser
3. A web or local server

**Choosing a code editor**

Basically any program that lets you write text and save that text as file can be a code editor.

For example Notepad on windows can be used to write this code, but there are many better tools for the job.

Some of the editors will complete code for you, and that can be a great help when you are learning a new language.

Auto-complete can also speed up production when writing code.

Here are a few that are favorites of programmers

NotePad++

Price:Free

I have not used this one much myself, but some of my friends swear by it.

The website boasts features such as syntax highlighting and syntax folding

Brackets

Price:Free

This is a good editor from Adobe. It has a lot of extensions and themes that you can install right from the editor itself.

It is the editor I use in all my YouTube videos and my Udemy course.

Sublime

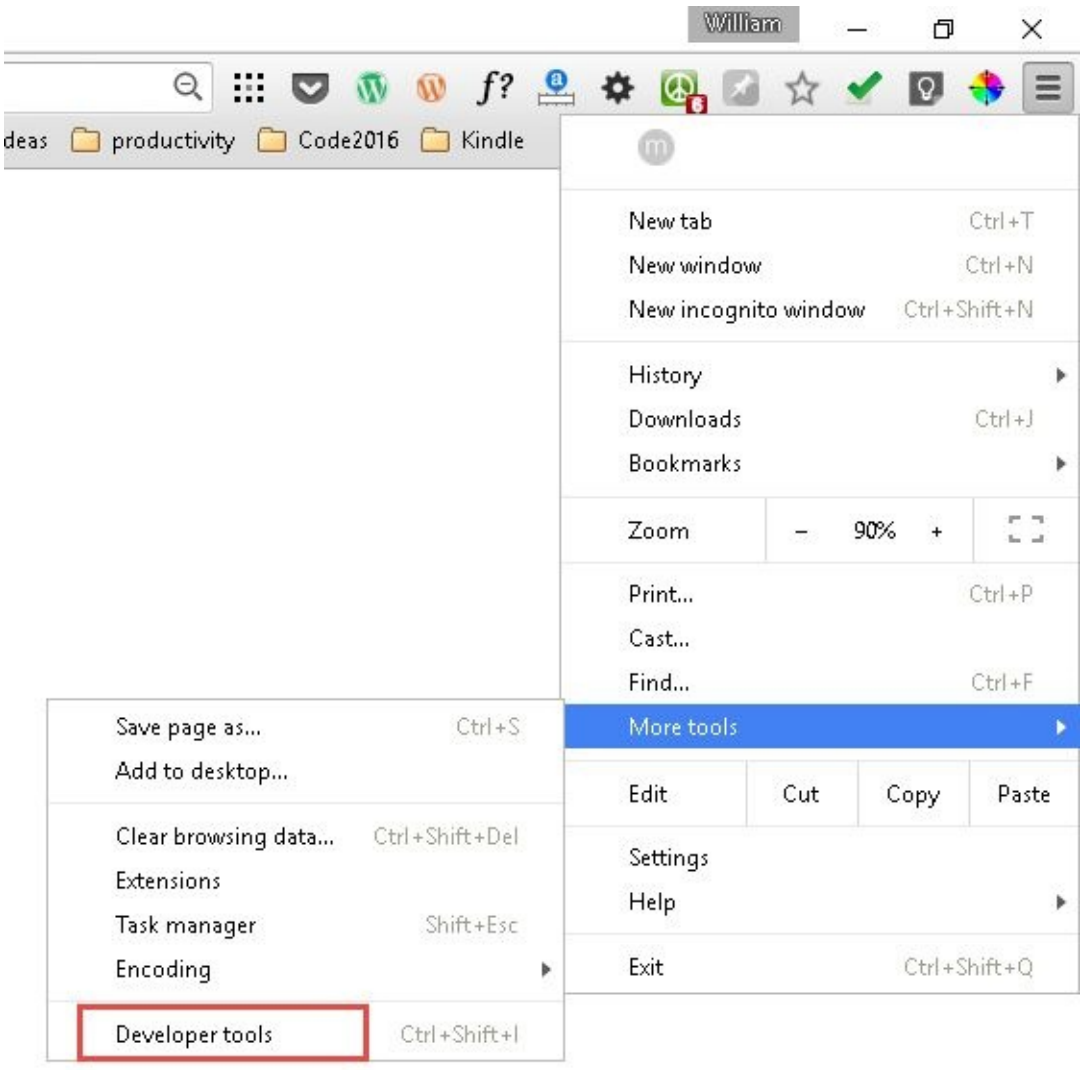
Price:Free to Try

I've been using this on trail for about 2 months now. There doesn't seem a time limit on the trail, but I'm so impressed with it I will probably purchase it soon. It does very well on learning what you type, so it can suggest words in the future. It also has wonderful search features so you can jump to any file almost instantly. It is what I use for most of my work now.

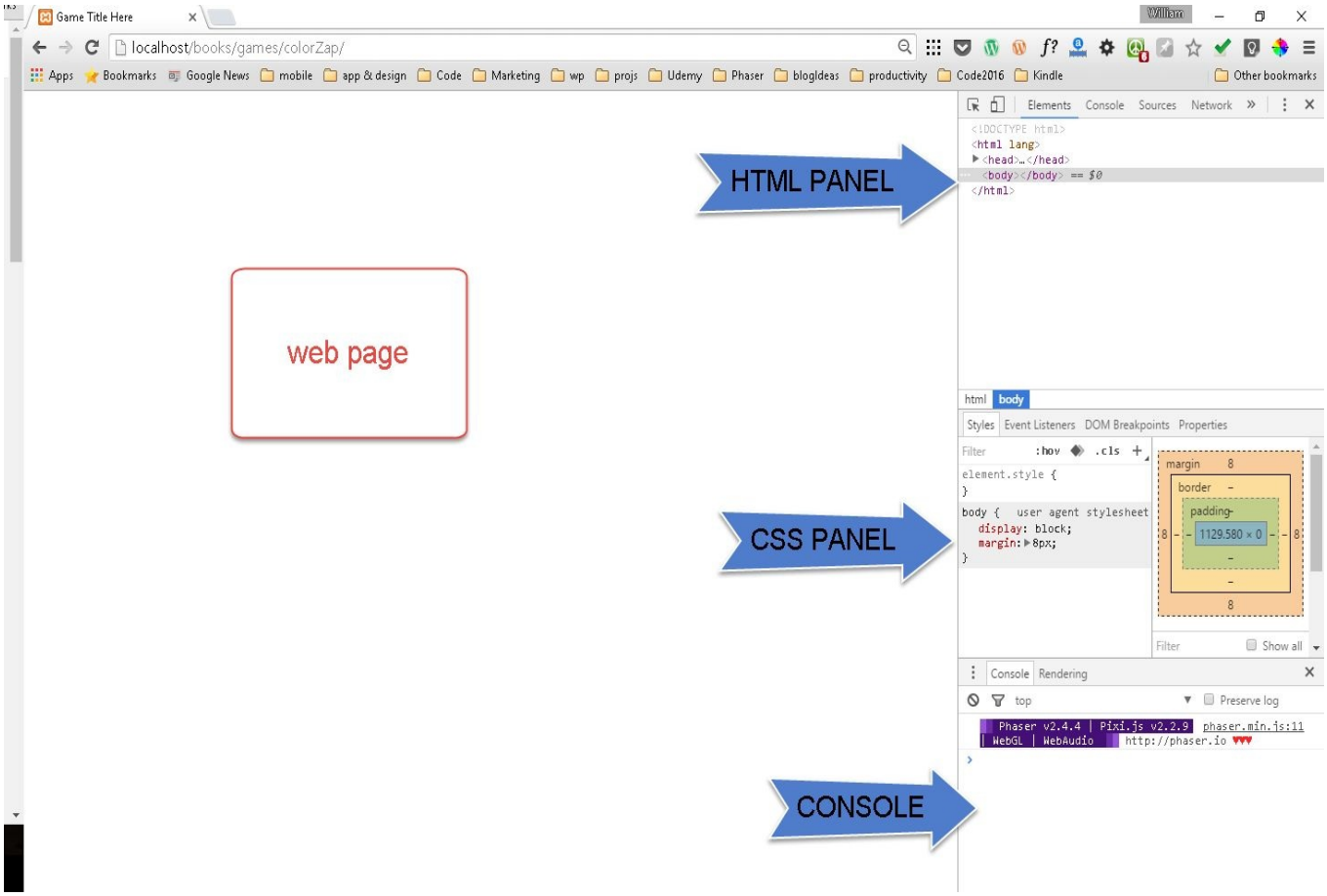
**Testing and Debugging**

For playing and testing the games I recommend using Goggle Chrome because of the nice built in debugging features. That is what I'll be using for the examples in this book. FireFox with firebug installed comes a close second.

To open up the developer console press the F12 key or find the developer tools under the menu under More Tools.




Here is the overview of the developer tools




**Html Panel**

The html panel shows you the html on the current page, you can edit, add or delete the html code, and it will change instantly on the web page.

**Css Panel**

Like the html panel you can edit, add or delete the css for any element here. It also gives you a visual view of the margin, border, and padding.

**Console**

This is the location where you can see messages and errors.

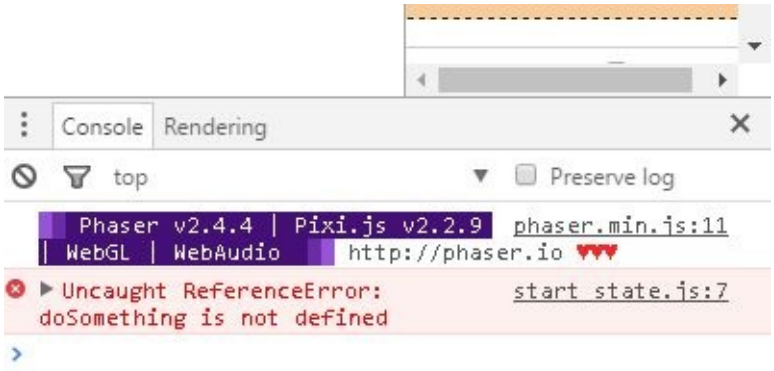
We can send information to the console by calling the following code

console.log(“my message”);

Or

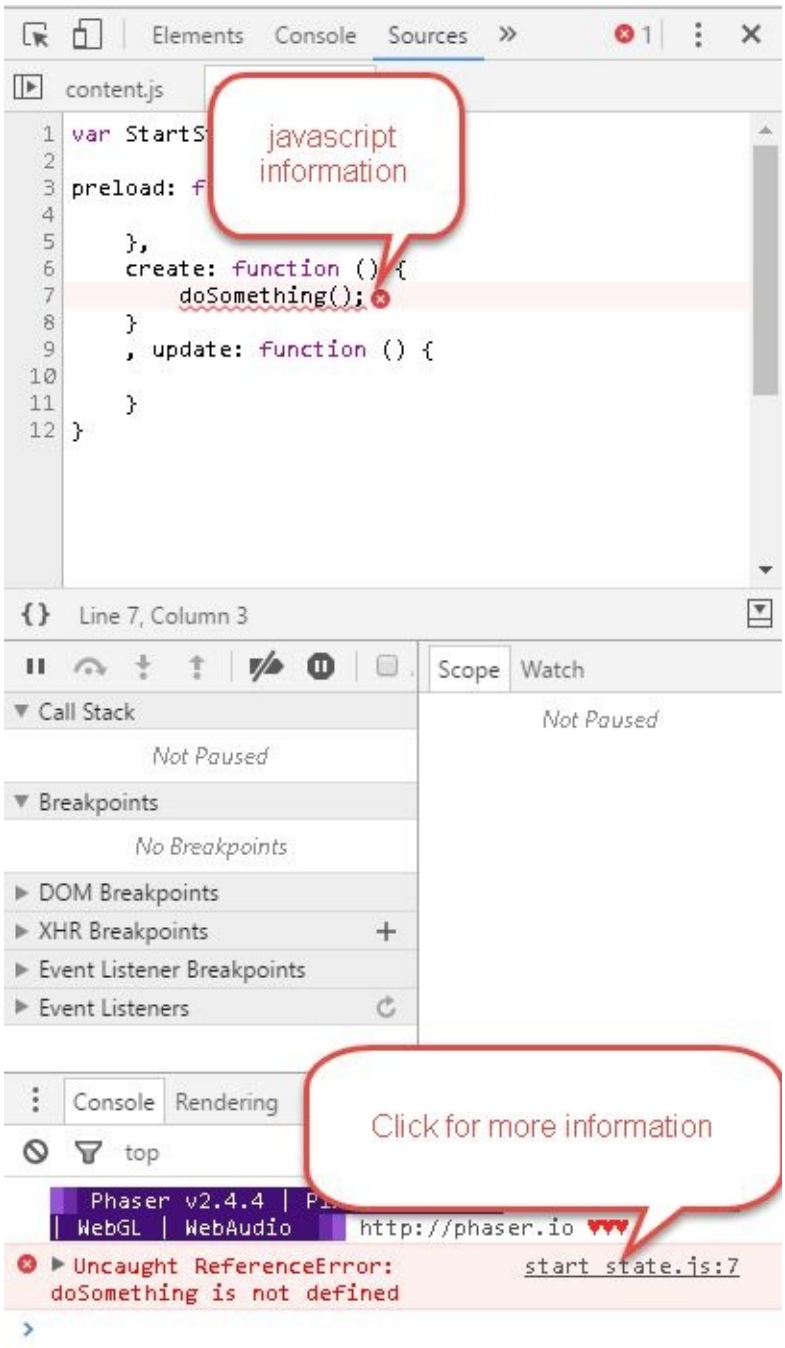
console.log(“score=”+score);

The console will also show any errors in the code in red with a line number




Click on the line number will display the html or javaScript information in the html panel.

Chrome also underlines the line with the error.


Elements
Console
Sources
content.js
javascript information
1 var StartS
2
3 preload: f
4
5 },
6 create: function () {
7 doSomething();
8 }
9 , update: function () {
10
11 }
12 }
Line 7, Column 3
Scope
Watch
Not Paused
Call Stack
Not Paused
Breakpoints
No Breakpoints
DOM Breakpoints
XHR Breakpoints
Event Listener Breakpoints
Event Listeners
Console
Rendering
top
Click for more information
Phaser v2.4.4
WebGL | WebAudio
http://phaser.io
Uncaught ReferenceError: doSomething is not defined
start state.js:7

# CHAPTER FOUR

## *Using a local Server*

**Installing a local server**

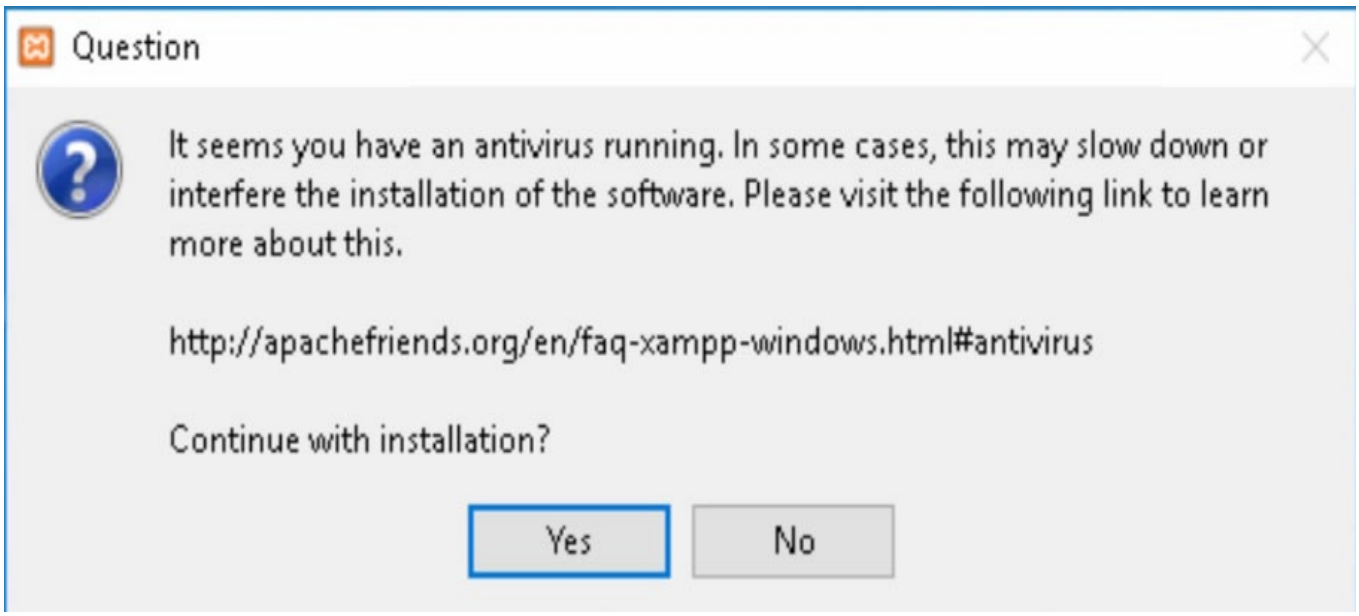
Phaser will not run from a local file like simple javaScript does. We either need to upload the files to a website, or install a local web server on our computer. This is a lot more simple than it sounds.

There are several local servers available but the one I use is Xampp.

There are versions for Widows, Mac and Linux and can be downloaded here.

After downloading run the program.

Usually the first thing I get is a warning about my virus protection.

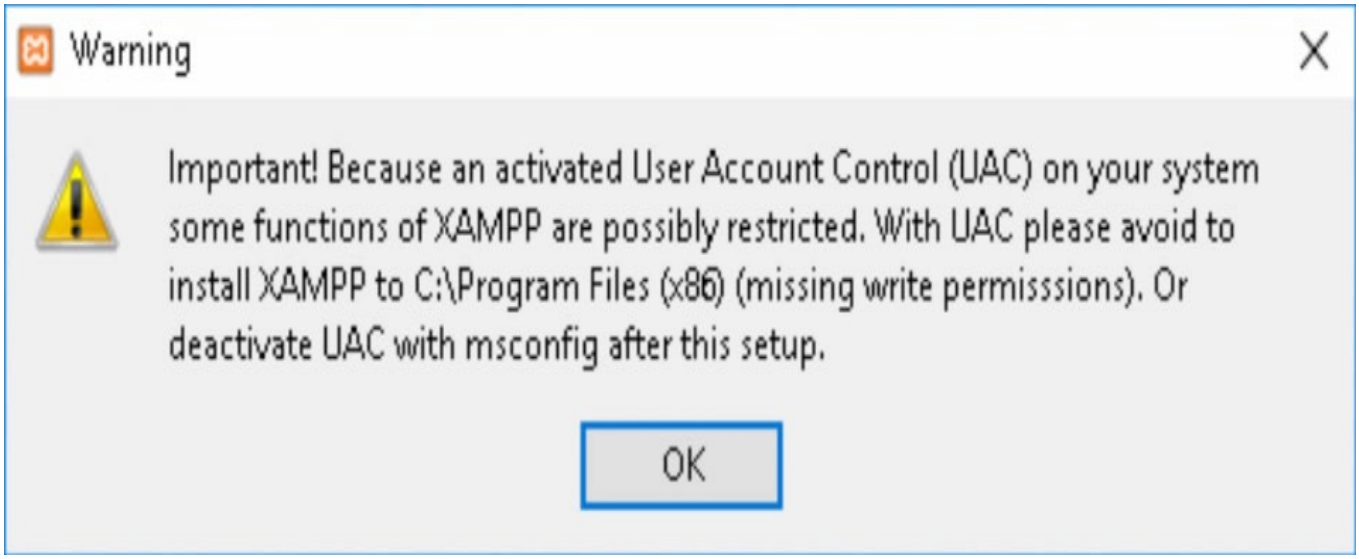


I’ve installed this many times, and have always gotten the warning, but it has never caused me any issues.

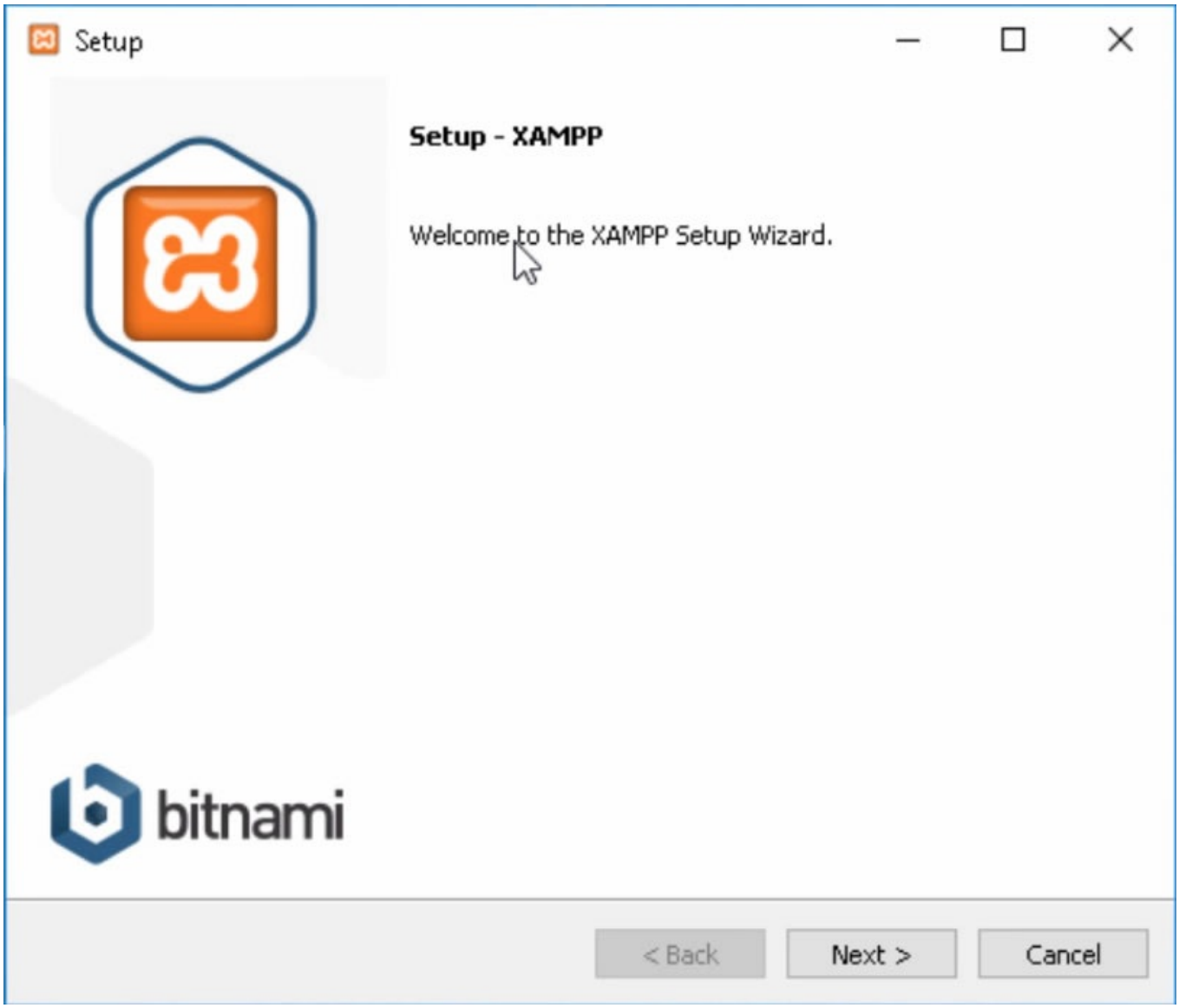
If you do have any issues, visit the link on the warning box to help resolve it.

Click YES to continue.

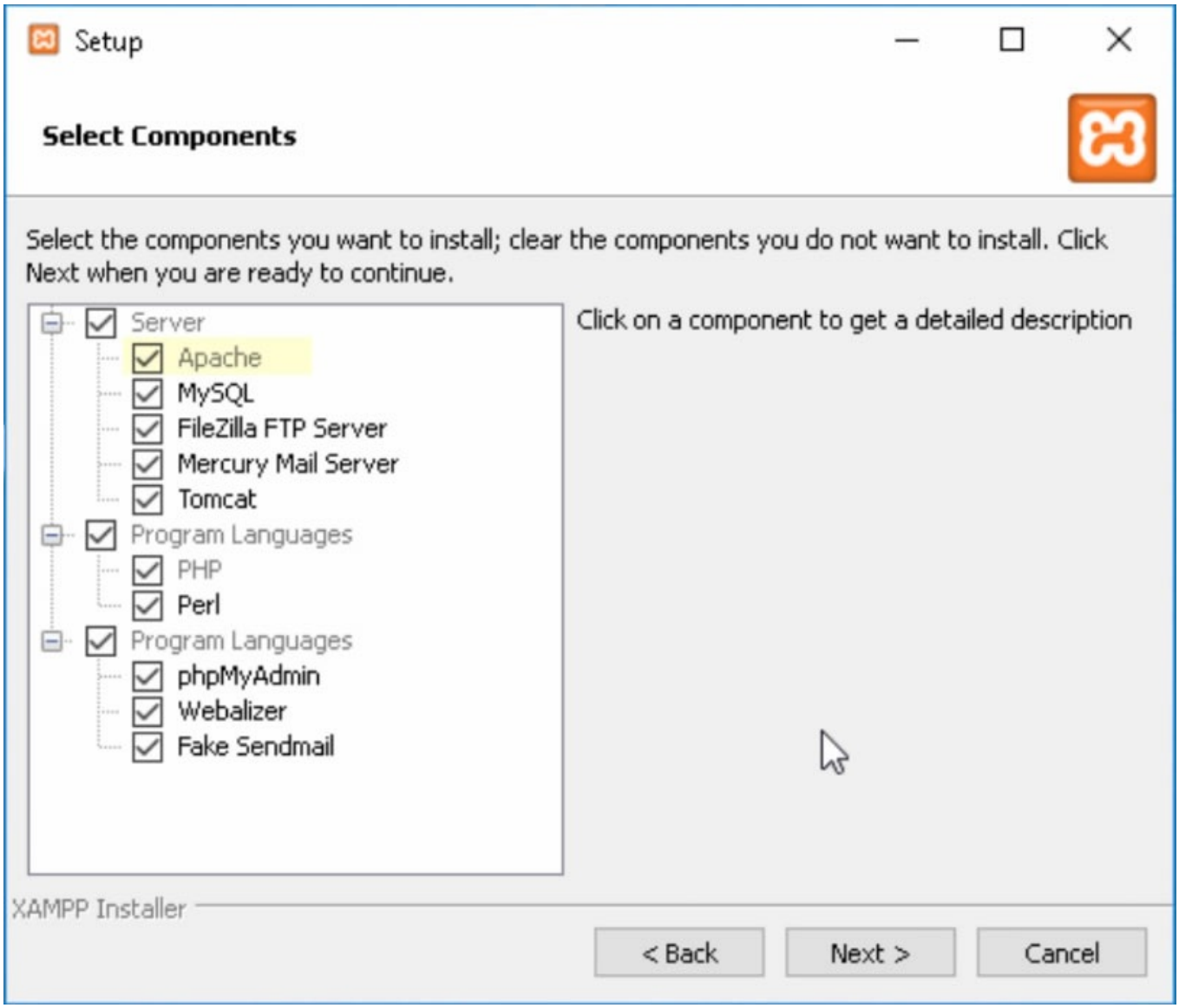
The second warning I get is about user accounts, but again I’ve never had any issue with it.




Click OK to continue.

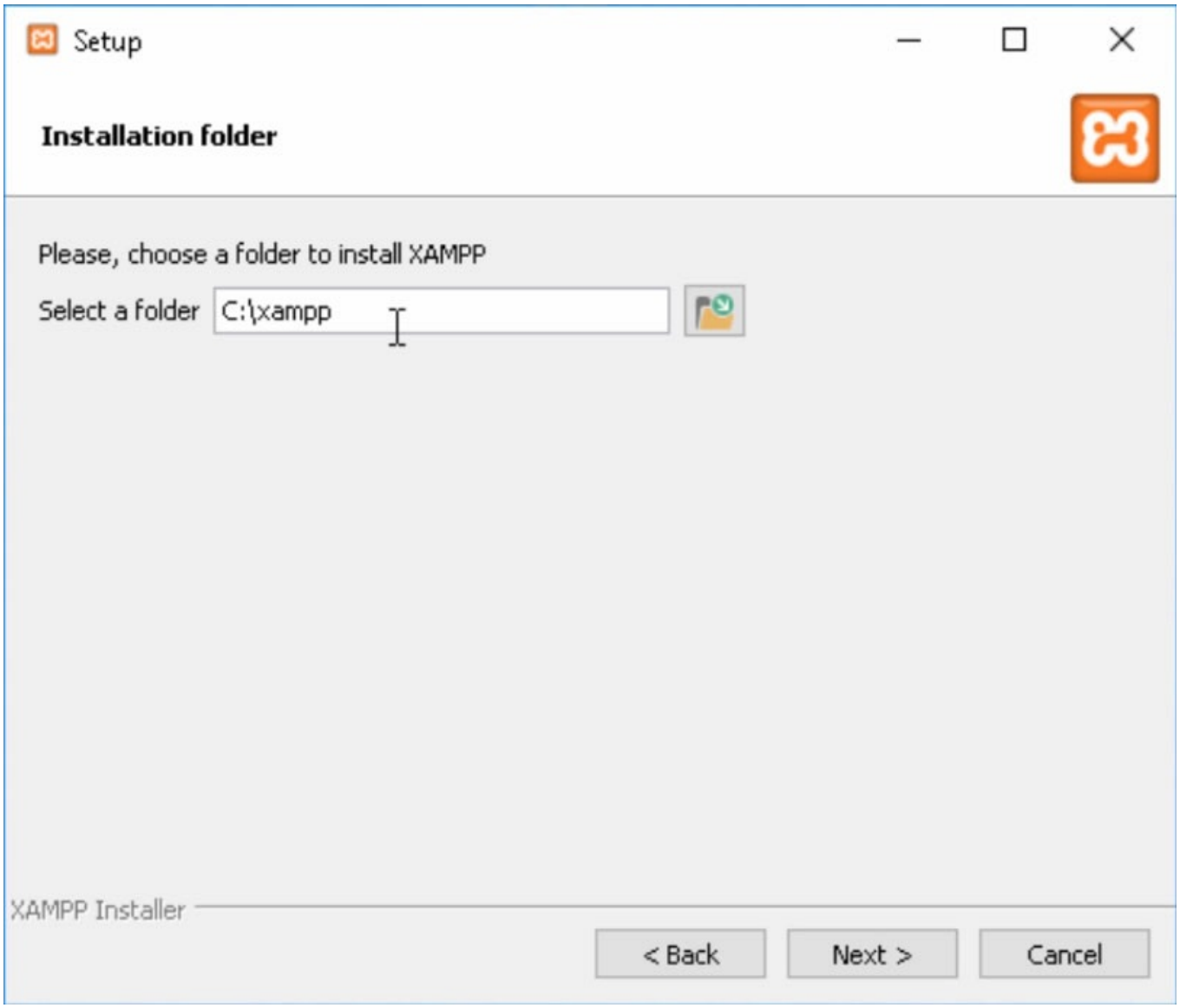



The welcome screens appears, click next to continue.

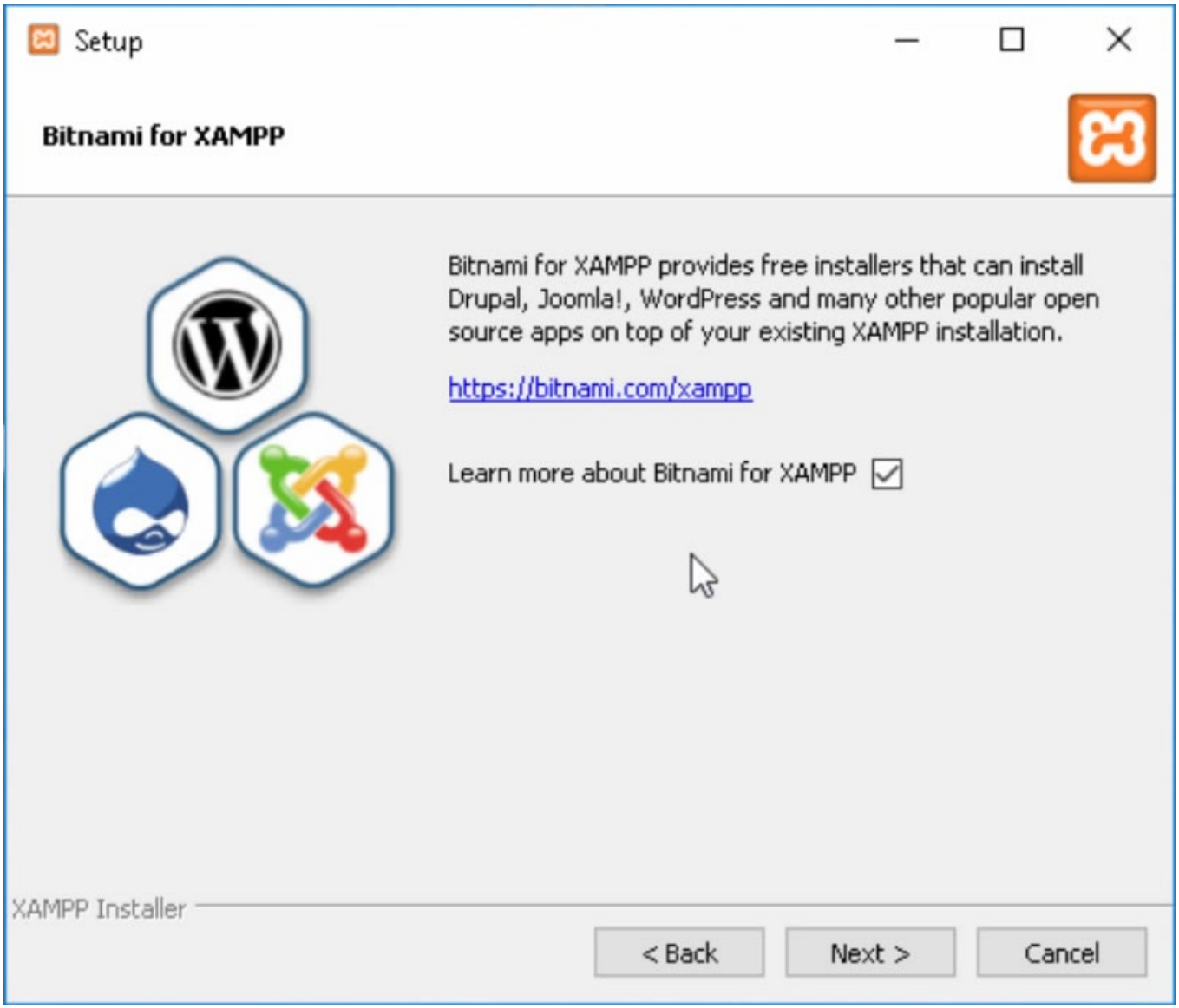



This screen lets you select which components you want to install.

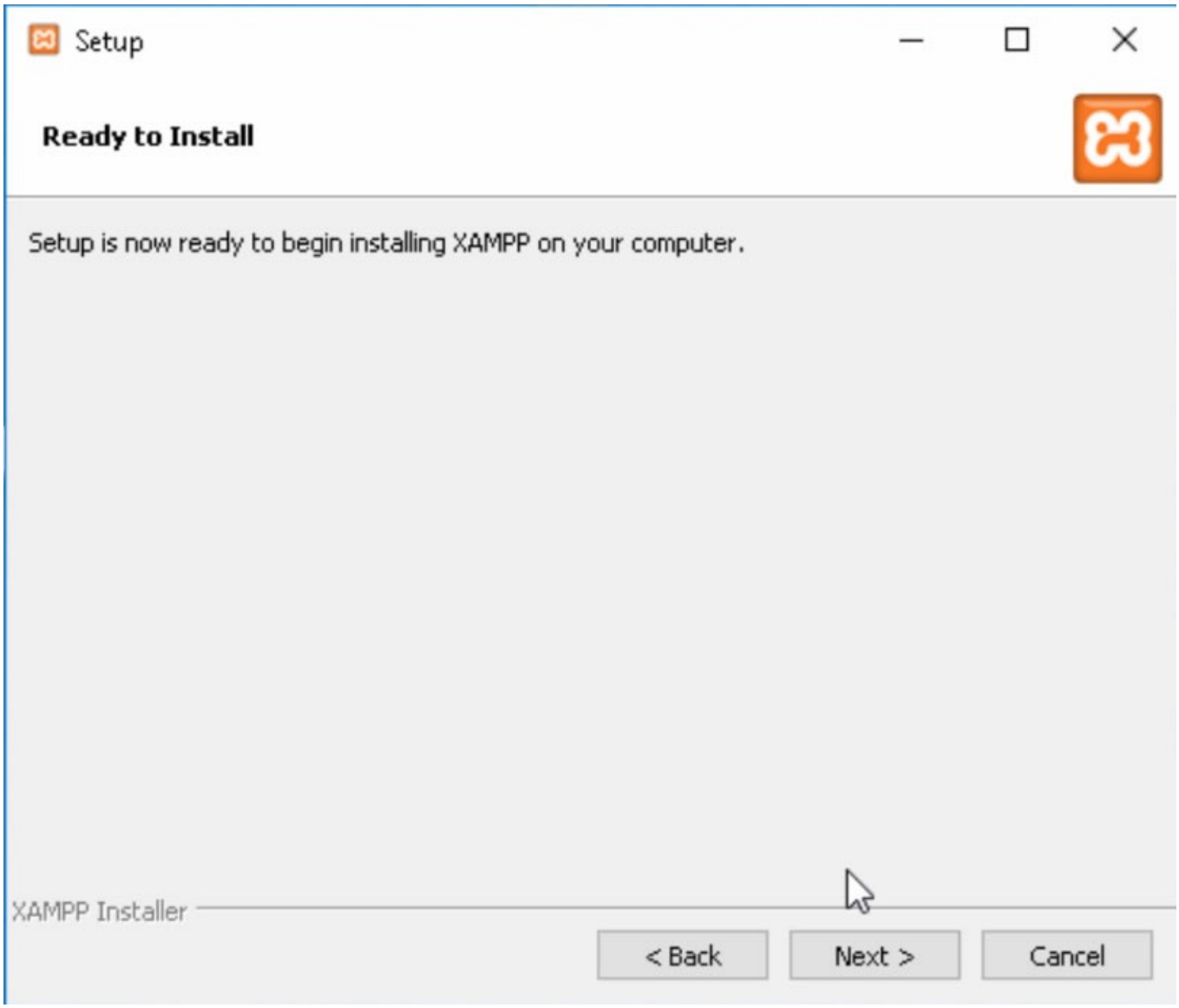
The required components are Apache and PHP. You may uncheck all the optional components, but there is no harm in keeping them, especially if you plan to do any other web development in the future. A local server saves a lot of time in development work.




The next screen is where you'd like to install the server. If you change the default location, or are on a Mac or Linux,please make note of the address, because you will need to place files at that location later.

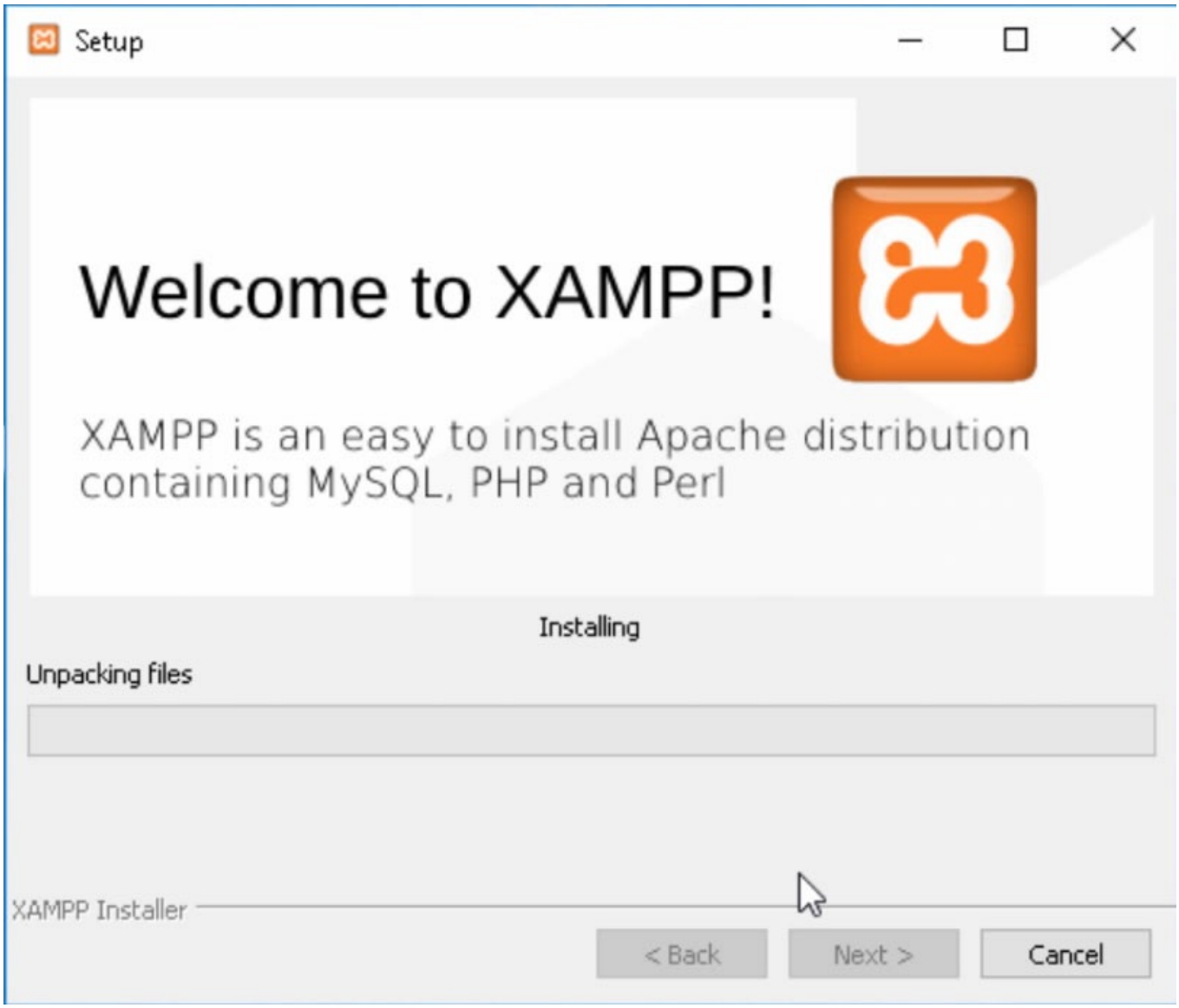



This screen talks a little about some of the frameworks that you can install on Xampp. We will not be needing this for our projects. Make sure you uncheck the more information box before clicking next to avoid a website being opened.

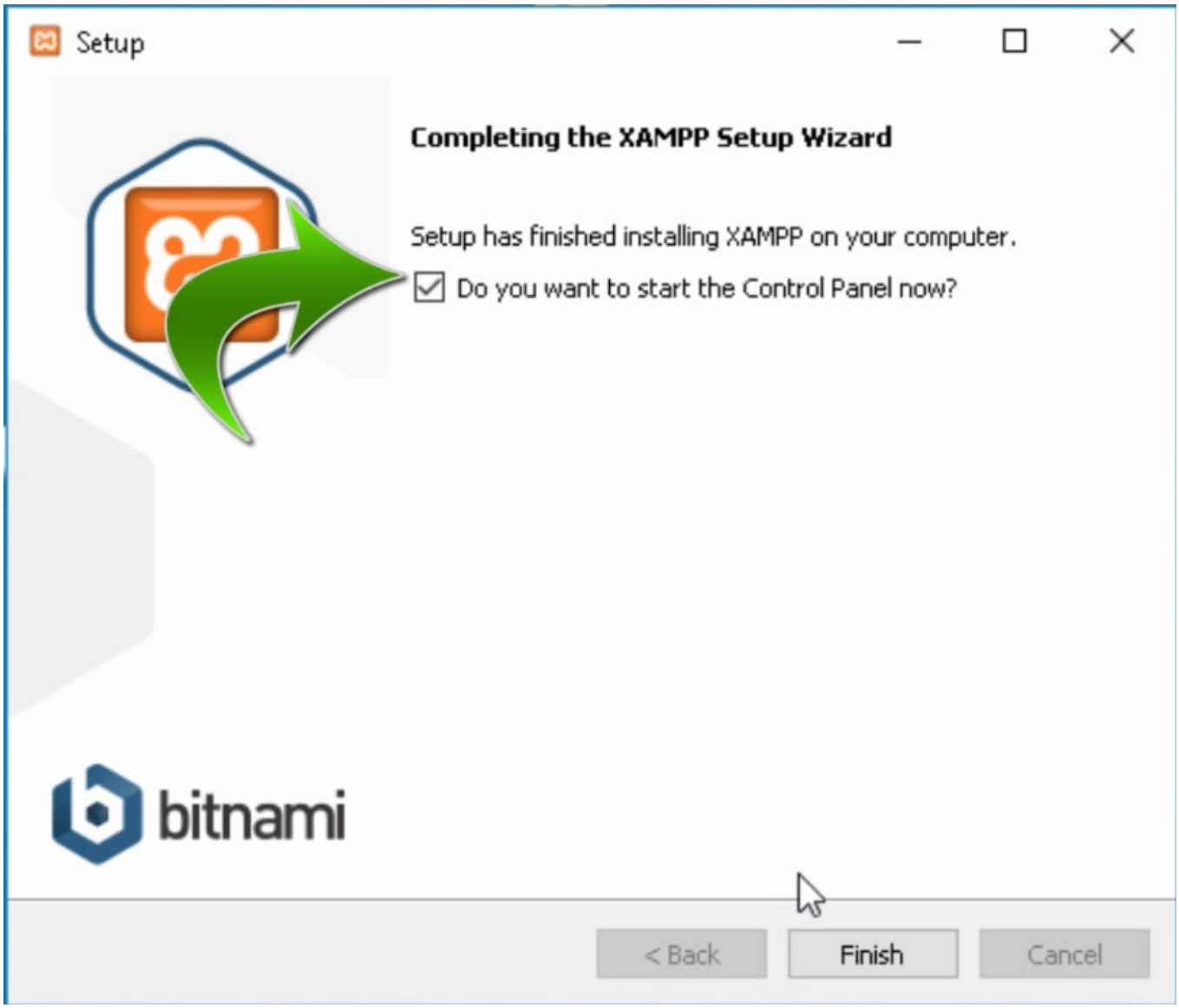



Now Xampp has the information it needs to install the server.

Click Next.



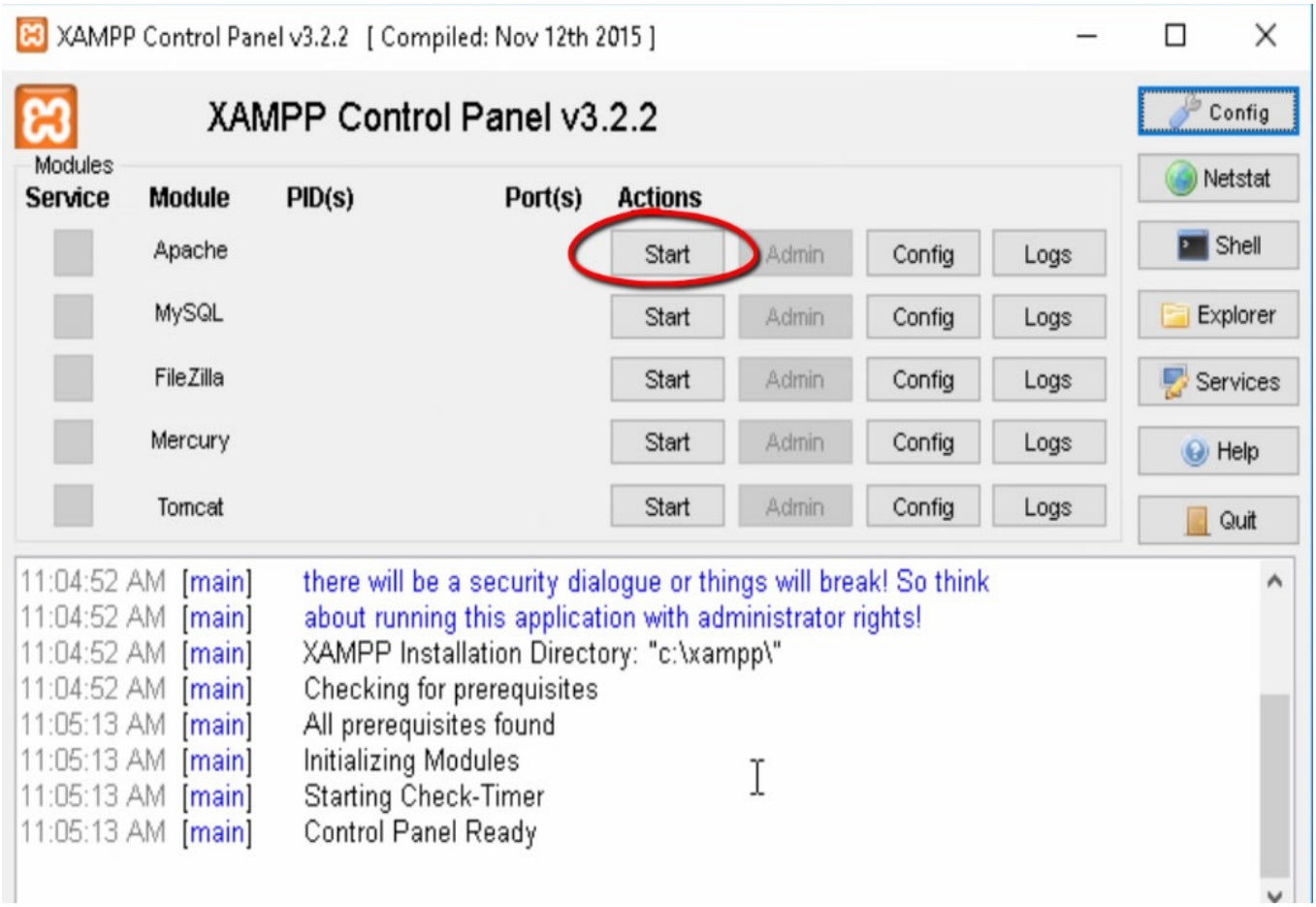

Xampp will now start installing, this can take between 10 to 20 minutes, depending on the speed of you computer.



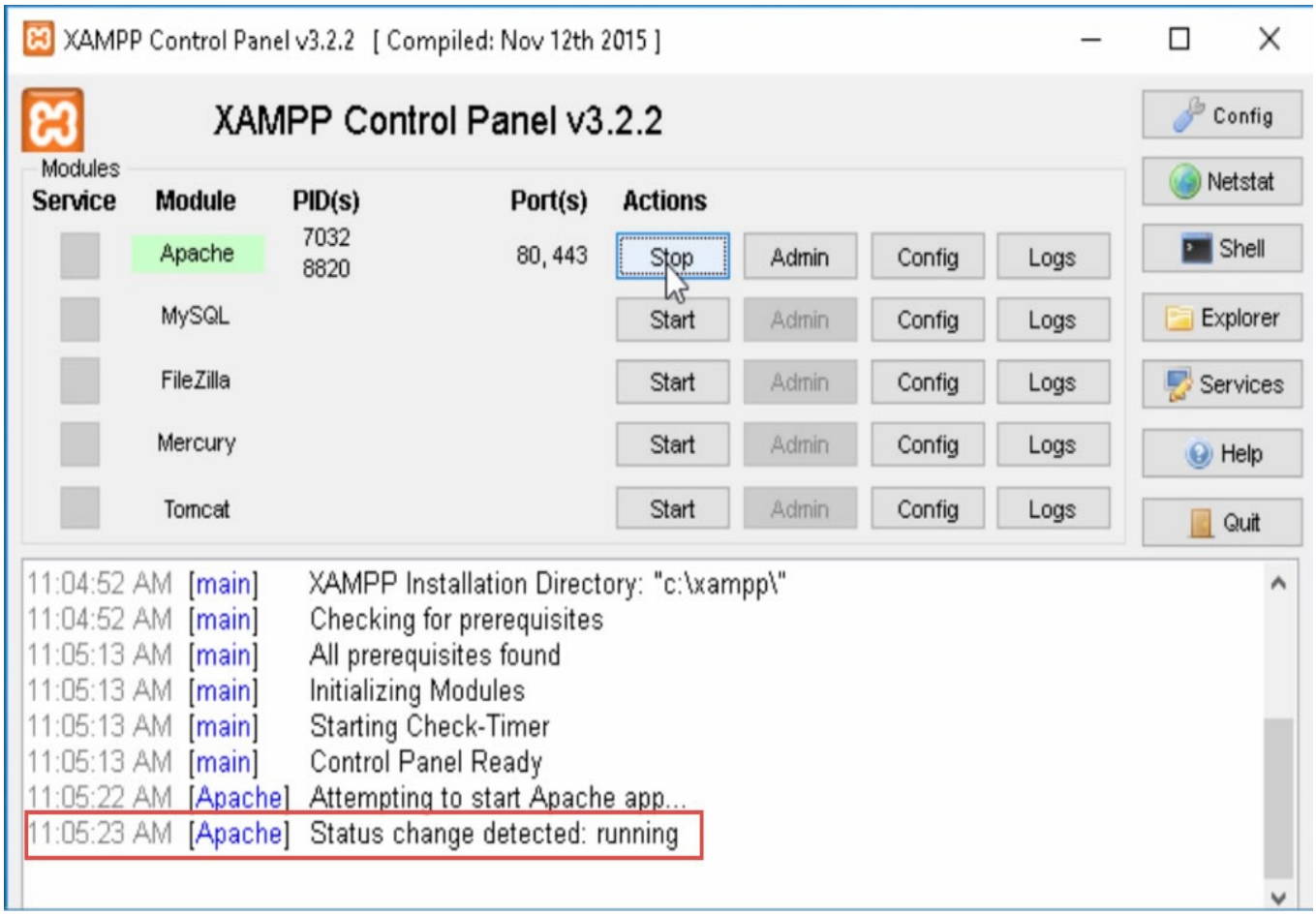

Now the server is installed. Make sure you keep the checkbox checked, so you can start the server now.


The first time you run Xampp you will get a language box like this. Click English and

Save.



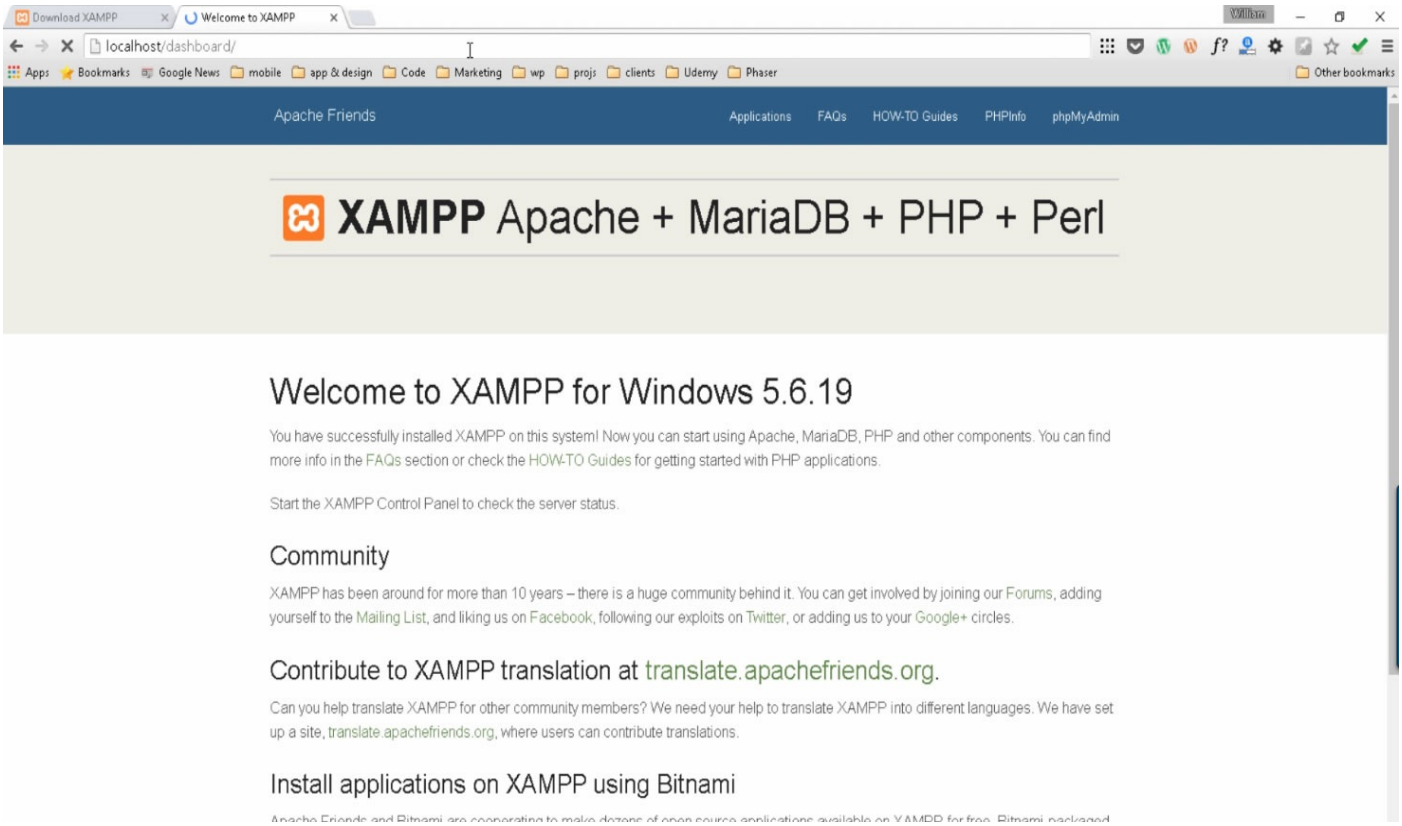

The Xampp Control Panel appears. Click the Apache start button.




If everything is set up correctly you will see the message “Status change detected:running”

One problem that can occur at this point, is if you are running Skype. If you get a red warning message mentioning Skype, simply shut down Skype, and press Start again.

Enter http://localhost into your browser. The following screen will appear.




Congratulations! Your local server is up and running!

# CHAPTER FIVE

## *Make the blank game*

Phaser uses objects called states. We can think of states in terms of screens.

For example a typical casual game might contain the following states

- Splash Screen
- Title Screen
- Instruction Screens
- Main Game
- Game Over/Play Again Screen

A state simply starts out as a simple object like this:

**var MyState = {**

**}**

Phaser has several built in functions for every state

The 3 most common are

- Preload
- Create
- Update

So a basic blank state would look like this

```
var MyState = {

preload: function () {

},
create: function () {

}
, update: function () {

}
}
```

**Preload**

The preload function is where we can set up a library of images, sounds and data.

**Create**

The create function is the place to create objects for text,sounds and images.

**Update**

The update function is a constant running loop. It is commonly used for checking collisions or updating variables

**Setting Up The Folder Structure**

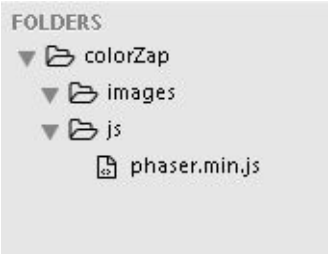
If you are using web space open the folder where you'd like to create your game

If you are using the xampp server open the htdocs folder

create a folder called games and inside games create another folder called colorZap

inside the colorZap copy the image folder and js from the resources you downloaded.

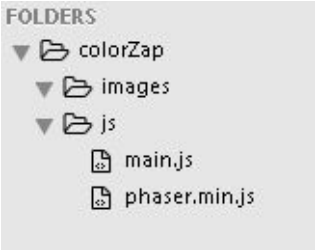
Place the Phaser.js file inside the js folder




**Declare the game**

We need a main file to launch the game

Create a new .js file called main.js




In the main.js file declare the following variable

**var game;**

This will be our global variable that will hold the entire game

We now need to make that variable into a phaser game when the page is loaded.

Place the following code into the main.js file

```
window.onload = function()
{
game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);
}
```

This code breaks down to

game=new Phaser.Game(game width,game height,render mode,div tag);

**Game width and height**

This is the size of the game to be drawn

**Render Mode**

There are 3 drawing modes for phaser

1.Phaser.Canvas

Uses the standard Html5 canvas.

2.Phaser.WebGL

Uses the Web Graphics Library for drawing.

WebGL is a JavaScript API for rendering interactive 3D and 2D graphics withing any web browser without the use of plug-ins.

3.Phaser.Auto

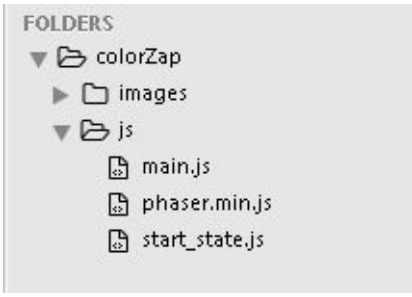
Uses the WebGL if available and falls back to the canvas if not.

**Div Tag**

A div tag to insert the game into. If the tag already exists in the html file the game is placed in this tag. If it does not exist, Phaser will create it. If the parameter is omitted Phaser will automatically create a new div tag and place the game inside.

**Making the Start State**

Make a .js file in the js folder called start_state.js and make a blank state inside.




Your start_state.js file should look like this:

```
var StartState = {

preload: function () {

},
create: function () {

}
, update: function () {

}
}
```

**Adding A State to the game**

We can add states to the game and then start them to change screens.

We add a state to the game by using this format:

game.state.add(Key,Object);

To load a state:

Game.state.start(Key);

To add a state to our game place the following code into main.js

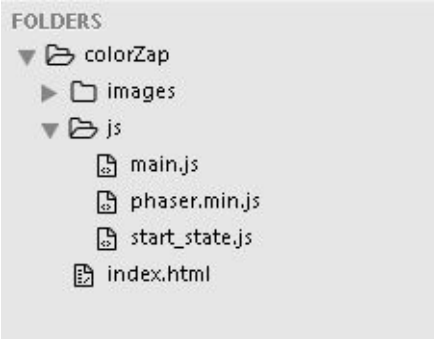
**game.state.add(“StartState”,StartState);**
**game.state.start(“StartState”);**

The first line adds a state definition, linking the string StartState to the object StartState. The two parameters don’t have to match,

Your main.js file should look now look like this

```
var game;
window.onload = function()
{
game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);
game.state.add(“StartState”,StartState);
game.state.start(“StartState”);
}
```

In your game folder make an file called index.html




Place this html code inside it.

This code loads the JavaScript and sets up div tag to place the game.

**<!DOCTYPE html>**
**<html lang=””>**
**<head>**
**<meta charset=“UTF-8”>**
**<title>Game Title Here</title>**
**<script src=“js/phaser.min.js”></script>**
**<script src=“js/main.js”></script>**
**<script src=“js/start_state.js”></script>**
**</head>**
**<body>**

** **

**</body>**
**</html>**

**Set up logging**

Inside of the create function of the start_state.js file add this line of code

**console.log(“Ready!”);**

This will show that everything is working right.

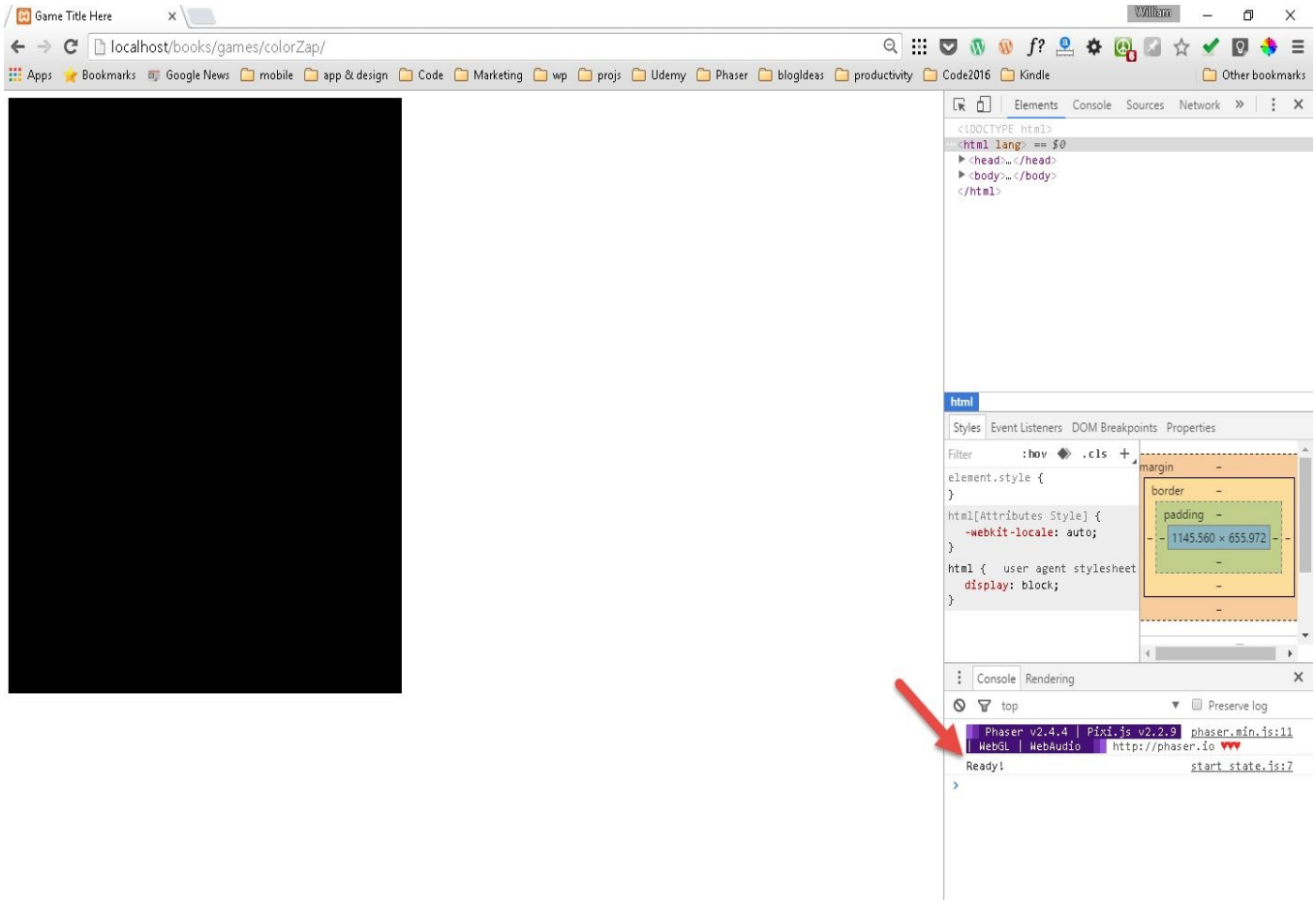
**Open the index file**

Browse to the index file on your web server.

If using a local server the path will be something like

Http://localhost/games/colorZap/index.php

You should see the word ready in the logging console



# CHAPTER SIX

## *Make the Title Screen*

**Loading an image into the library**

Phaser keeps a library of images and sounds, by caching those items by a key. To place an image into the library we use code in the following format

**game.load.image(key_name, path_to_image);**

These load statements generally are placed in the preload functions.

Type this code into your preload function

preload: function () {

**game.load.image(“gameLogo”, “images/title/gameLogo.png”);**

}

**Place an image on the canvas**

After you cache an image in the library you can place it on the canvas.

All images and other objects are defined in the create function

Here is the format for placing an image on the canvas

**this.imageObject=game.add.sprite(x,y,key_name);**

And here is the code for our game

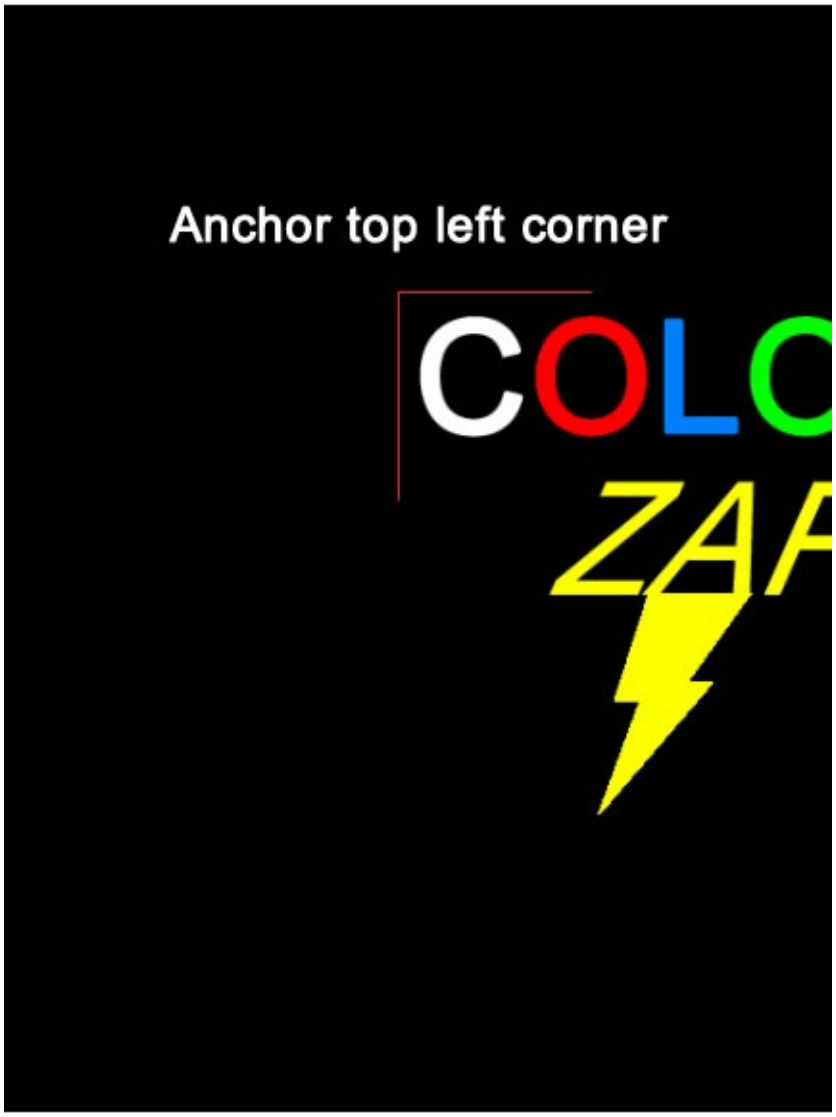
create:function()
{
**this.logo = game.add.sprite(game.world.centerX, 180, “gameLogo”);**
}

game.world.centerX is a shortcut in phaser to find the center of the canvas

However as we see, the image is not centered to the canvas


This is due to where the image’s anchor is set. An anchor is where the image’s x and y positioning starts.

By default it is set to the top left-hand corner.

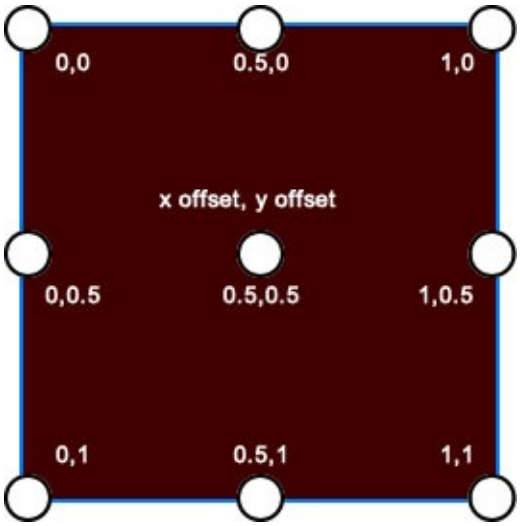



The formula for setting an anchor is:

this.image.anchor.set(xOffset,yOffset);

xOffset and yOffset are numbers between 0 and 1.

While an anchor can technically start anywhere, there are 9 most popular points.




I personally either use 0,0 (top left) or 0.5,0.5 (center) for 99% of my projects.

We need to set the anchor to the center of the image.

To do this we can use the following code in the create function

**this.logo.anchor.set(0.5,0.5);**

Refresh the page and we see that the image is now correct

**Adding a sprite sheet**

Sprite sheets are image files that contain multiple images that can be separated and called in code

Here is the outline of the code needed to load a sprite sheet in phaser

game.load.spritesheet(“libary_key”, “path_to_image”, cell_width, cell_height);

We are going to load a sprite sheet and use the image to make a button

Here is the button image and each frame is 264 wide by 75 high


In your preload function place the following code:

**game.load.spritesheet(“buttons”, “images/ui/buttons.png”, 264, 75);**

A loaded sprite sheet is split into frames.

The frame indexes always start with 0

The loaded button image would be split as follows:


Buttons

To set up a button with rollover and pressed states, can use the following

ButtonObject=game.add.button(x,y,library_key,function_when_pressed,scope,over_fran

This is the code we will place in the create function to place the button and center it:

**this.buttonStart = game.add.button(game.world.centerX, game.world.height-100, 'buttons', this.startTheGame, this, 7, 6, 7);**

**this.buttonStart.anchor.set(0.5,0.5);**

This will place the button in the center horizontally and vertically place it 100 pixels from the bottom of the screen.

Examining the properties one by one

x and y is the left and top position respectively

Library_key('buttons') is the key used to preload the sprite sheet

Function_when_pressed (this.startTheGame) is the function called when the button is pressed

Over_frame is the frame display when the mouse is rolled over the button

Normal_frame is the frame normally displayed

Pressed_frame is the frame displayed when

Add the function for when the button is pressed to the end of the title screen code

**startTheGame: function () {**

**console.log("start button pressed");**

**}**

**Title Screen Code**

Your code should now look like this:

var StartState = {

preload: function () {

game.load.image("gameLogo", "images/title/gameLogo.png");
game.load.spritesheet("buttons", "images/ui/buttons.png", 264, 75);
},

```
    create: function () {


    this.logo=game.add.sprite(0, 0, “gameLogo”);
    this.logo.anchor.set(0.5,0.5);
    this.buttonStart = game.add.button(game.world.centerX, game.world.height-100, ‘buttons’, this.startTheGame, this, 7, 6, 7);
    this.buttonStart.anchor.set(0.5,0.5);


    },
    update: function () {


    }
    , startTheGame: function () {


    console.log(”Button Pressed”);


    }
```

Run and test the game.

Press the start button.

Expected Result:You will see the message “*Button Pressed”* in the console.

If there are any errors, compare your code with the code above to look for any differences


COLOR
ZAP
START

# CHAPTER SEVEN

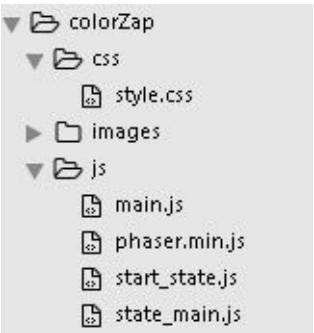
## *Make the Main Game*

### Add the Main State

The title screen being built, we now need to add the main game screen.

This is where all the action will take place.

Create a new file in the js folder called **state_main.js**




We will call this state StateMain

Set up the starter code

```
var StateMain={
preload:function()
{
```

**},**
**create:function()**
**{**

**},**
**update:function()**
**{**
**}**

**}**

Now we need to add the main state to the game

Open Main.js and add the line

**game.state.add(“StateMain”,StateMain);**

Just before the line

**game.state.add(“StartState”,StartState);**

Your Main.js file should now look like this

```
var game;
window.onload = function()
{
game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);
game.state.add(“StateMain”,StateMain);
game.state.add(“StartState”,StartState);
game.state.start(“StartState”);
```

}

Open your start_state.js file

Replace the line

**console.log(”Button Pressed”);**

With this line of code

**game.state.start(“StateMain”)**

In the create function of state_main.js place the code

**console.log(”Game Started!”);**

In the index.html file we need to add the new javaScript file to the code

Add the line **<script src=“js/state_main.js”></script>** to your index.html file so that it now looks like this:

New code is in bold

<!DOCTYPE html>
<html lang=””>
<head>
<meta charset=“UTF-8”>
<title>Game Title Here</title>
<script src=“js/phaser.min.js”></script>
<script src=“js/main.js”></script>
**<script src=“js/state_main.js”></script>**
<script src=“js/start_state.js”></script>

```
</head>
<body>

<div id="ph_game"></div>

</body>
</html>
```

Run and test the game.

Press the start buttons.

Expected result:The game will change to a black screen and “Game Started!” will displayed in the console

**Adding the Squares**

Now let’s add the color squares to the game

First we need to load the four square images into the library

In the preload function of main_state.js add the following lines:

```
game.load.image("yellow","images/main/blocks/yellow.png");
game.load.image("blue","images/main/blocks/blue.png");
```

**game.load.image(“red”,“images/main/blocks/red.png”);**
**game.load.image(“green”,“images/main/blocks/green.png”);**

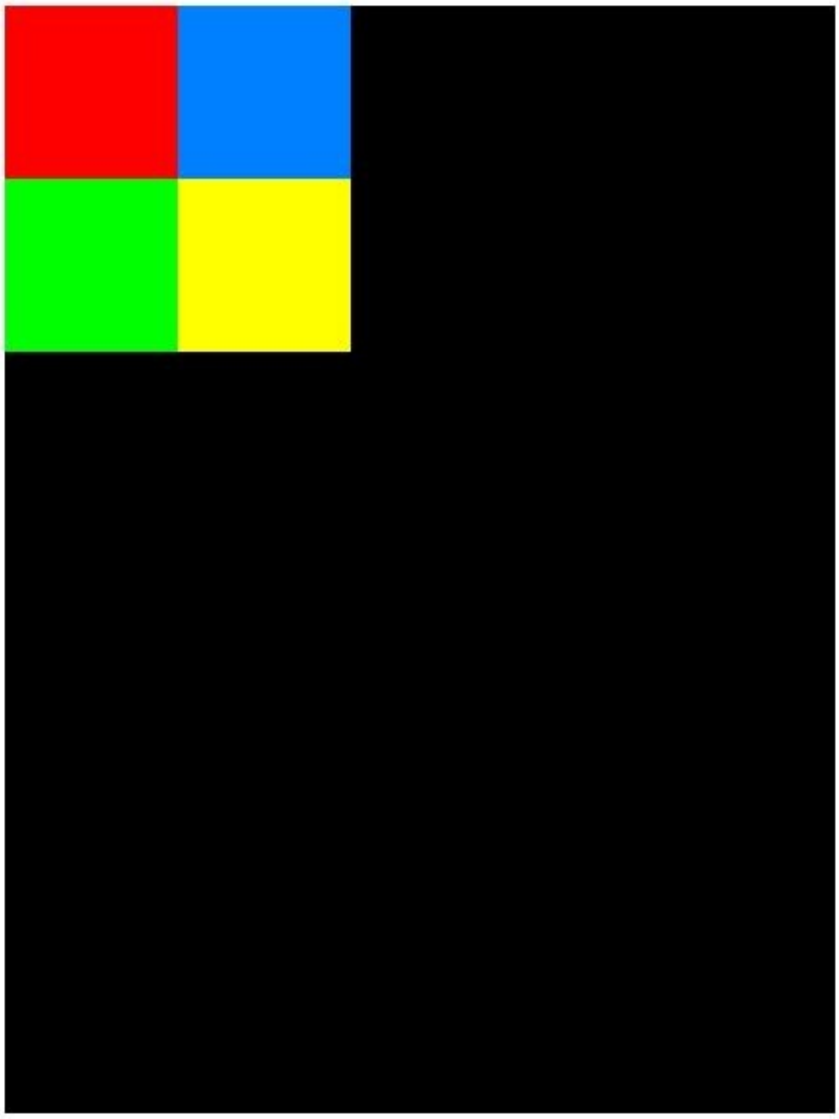
This will load each of the images in the library.

Now we can place the images on the canvas

In the create function place the following code:

**this.red=game.add.sprite(0,0,“red”);**
**this.blue=game.add.sprite(100,0,“blue”);**
**this.green=game.add.sprite(0,100,“green”);**
**this.yellow=game.add.sprite(100,100,“yellow”);**

This will set all the images in the top left corner of the game



**Working with Groups**

Phaser groups allow you to combine objects and position them as one.

We need to place the colored blocks in the center bottom of the screen, we could just set each x and y of each block, but it would be better to line it up using maths so if we change the size of the screen for a mobile device, which we will do later in this book, we can always put it in the center.

To easily accomplish this we need to group all the color blocks together

Adding a group to a game is very simple, place this code in the create function of main_state.js

**this.blockGroup=game.add.group();**

is all that it takes to add a group to Phaser

To add the images to the group add the following:

**this.blockGroup.add(this.red);**

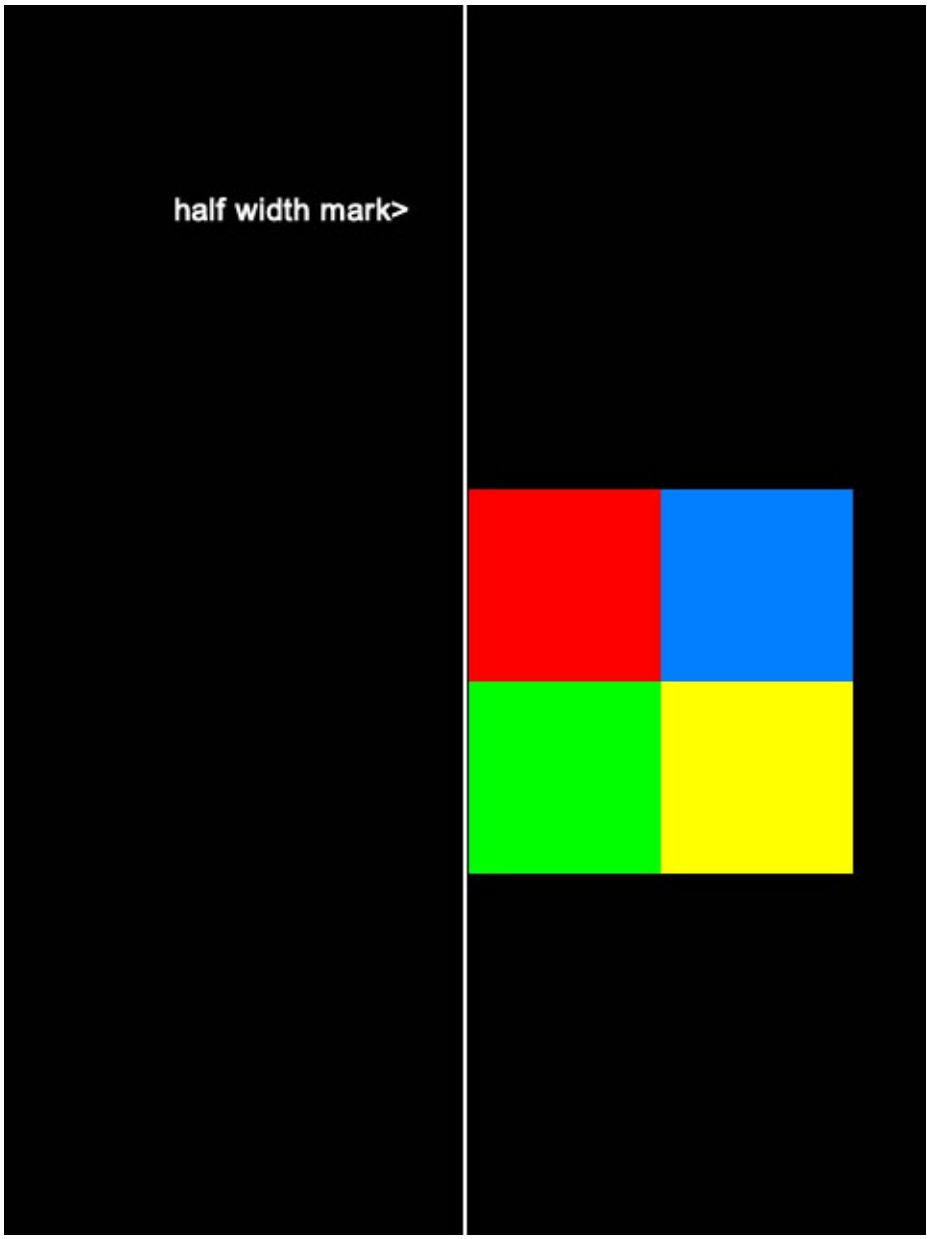
**this.blockGroup.add(this.blue);**

**this.blockGroup.add(this.green);**

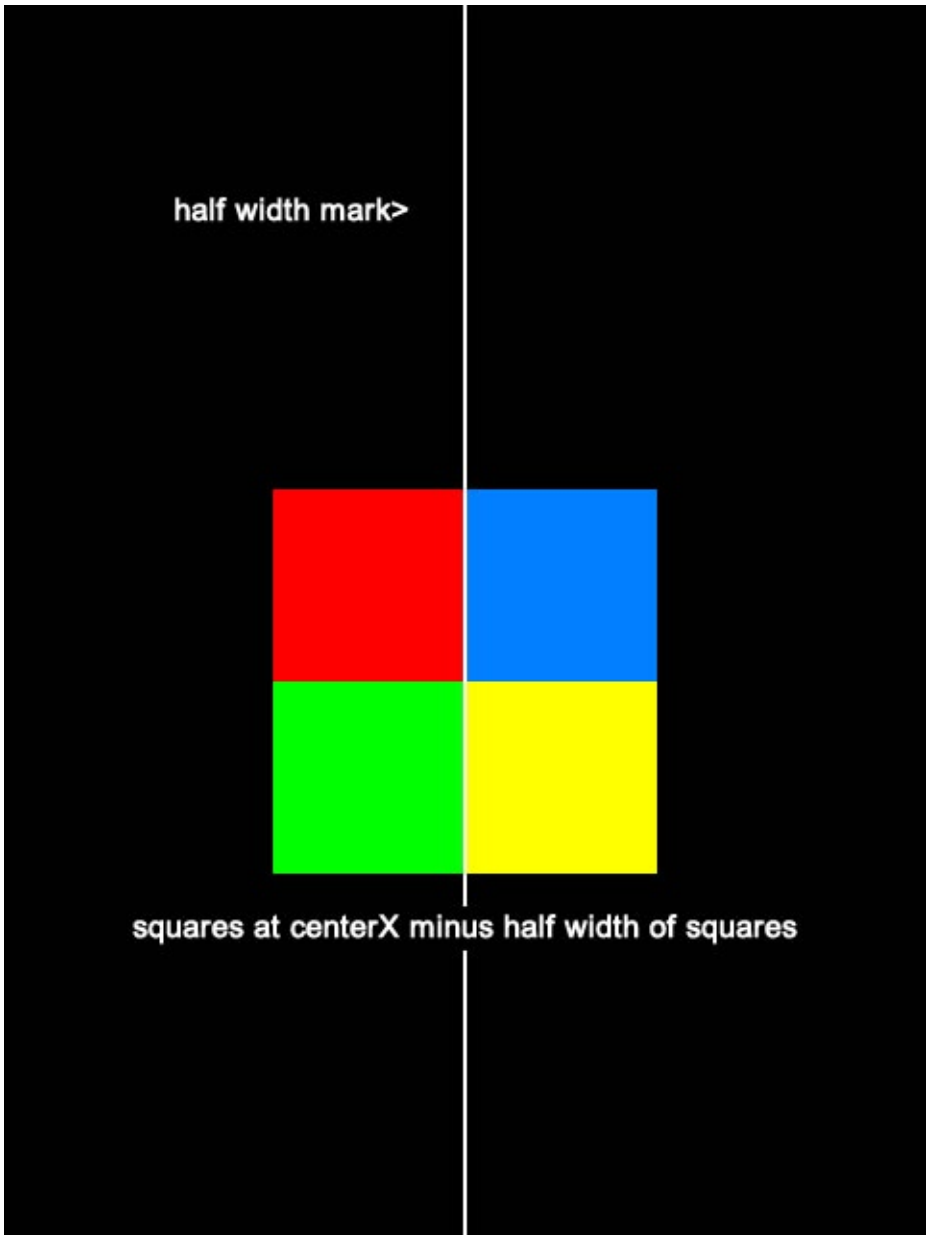
**this.blockGroup.add(this.yellow);**

Phaser Groups have no anchor setting so to place it in the center bottom. What we can do is use a little math. Take the centerX (game.world.centerX) and subtract half the width of the group to align it to the center horizontal. Then take the game height and subtract the height of the group plus a small amount to give it a little margin.

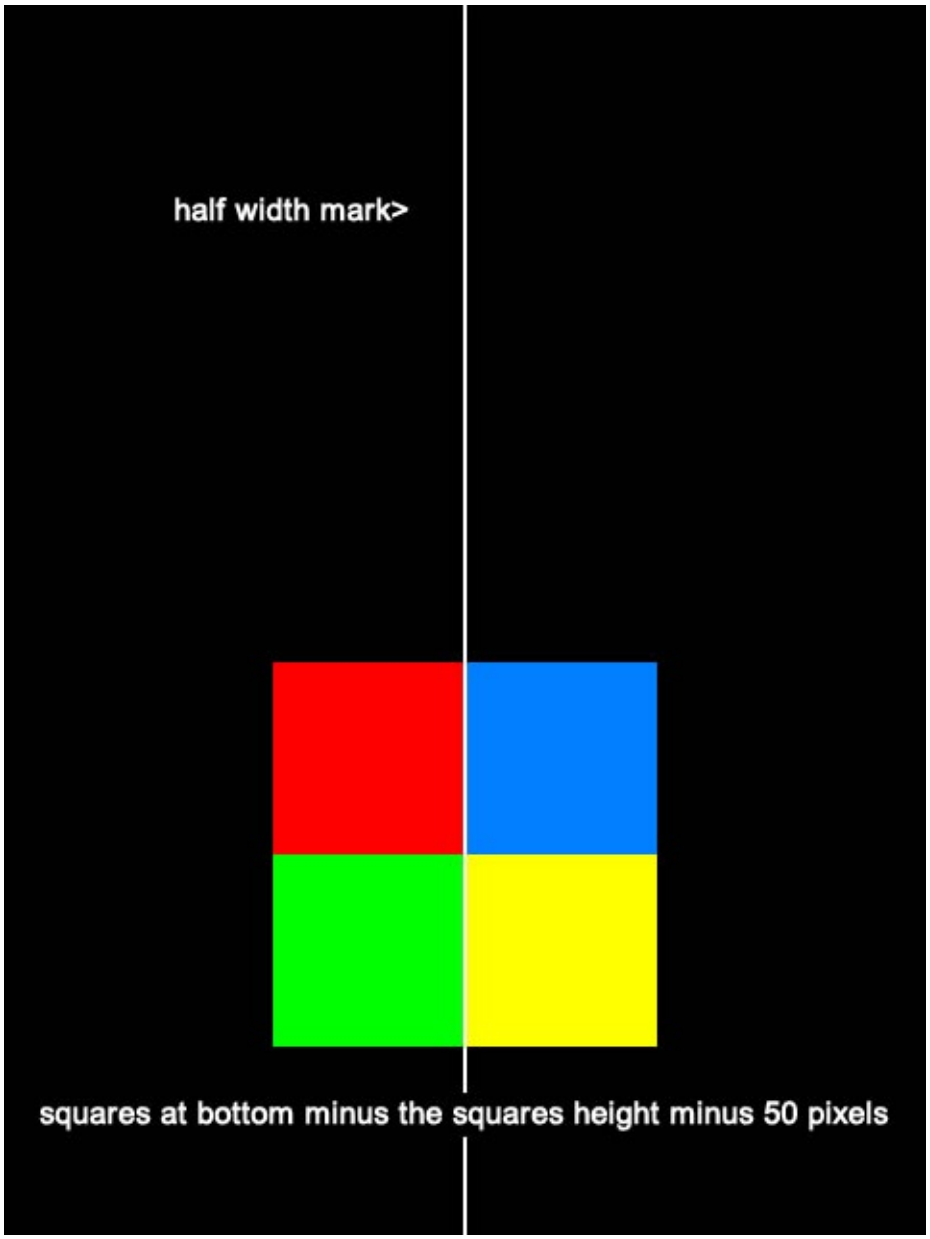
Squares at centerX




Now we need to take the groups width, divide it by two and subtract that number from the game’s half width




And then we need to set this at the bottom, minus the height of the squares and a small buffer of 50 pixels.




Here is what that looks like in code:

**//take the center of the game and subtract 50% of the group's width**

**this.blockGroup.x=game.world.centerX-this.blockGroup.width/2;**

**//set the group at the bottom of the game, subtracting the group height plus a little to create a margin**

**this.blockGroup.y=game.height-this.blockGroup.height-50;**

**Clicking Objects**

Setting up for Phaser to accept clicks is a three step process. The first thing is to tell Phaser which item you want to listen to.

We do that by setting the inputEnabled property of that object to true. By default every new object's inputEnabled property is set to false. This is to reduce the runtime load. If every object was listened to automatically it could seriously slow down your code.

Add this code to the end of your create statement.

**this.red.inputEnabled =true;**

**this.blue.inputEnabled =true;**

**this.green.inputEnabled =true;**

**this.yellow.inputEnabled =true;**

This will enable each object for input

The second step is to add a listener to each object.

This is the format for adding a listener.

//mouse down

**this.object.events.onInputDown.add(function,scope);**

//mouse up

**this.object.events.onInputUp.add(function,scope);**

Let's make a new function to hold all our listeners, so they are easy to find later.

**setListeners:function()**

**{**

**this.red.events.onInputDown.add(this.changeColor,this);**

**this.blue.events.onInputDown.add(this.changeColor,this);**

**this.green.events.onInputDown.add(this.changeColor,this);**

**this.yellow.events.onInputDown.add(this.changeColor,this);**

**}**

At the end of your create function call the setListeners function

**this.setListeners();**

We could set up a different function for each object, changeToRed(), changeToBlue(), etc. But it will be more efficient in this case to use the same function for each object.

The third and find step is to set up the function to be called when clicked.

Add this function to the end of your file. The target parameter is the object that is clicked.

**changeColor:function(target)**

**{**

**}**

We need a way to tell the objects apart. We can do this by simply setting a unique name for each object.

Place this code just after the inputEnabled lines of code

**this.red.name=“red”;**

**this.blue.name=“blue”;**

**this.green.name=“green”;**

**this.yellow.name=“yellow”;**

The last thing is to set up some logging for testing

Add this line to your changeColor function.

**console.log(target.name);**

**Testing the Click Events**

Your code should now look like this

```
var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
},
create: function() {
this.blockGroup = game.add.group();

this.red = game.add.sprite(0, 0, “red”);
this.blue = game.add.sprite(100, 0, “blue”);
this.green = game.add.sprite(0, 100, “green”);
this.yellow = game.add.sprite(100, 100, “yellow”);

this.red.inputEnabled = true;
this.blue.inputEnabled = true;
```

```
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.setListeners();
    },
    setListeners: function() {
    this.red.events.onInputDown.add(this.changeColor, this);
    this.blue.events.onInputDown.add(this.changeColor, this);
    this.green.events.onInputDown.add(this.changeColor, this);
    this.yellow.events.onInputDown.add(this.changeColor, this);
    },
    changeColor: function(target) {
    console.log(target.name);
    },
    update: function() {
```

}

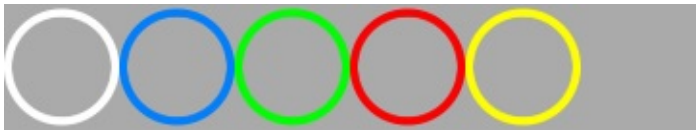
}

Testing-Refresh the window. Click each of the colors

Expected result:The names of the blocks will appear in the console window.

**Add the Ring**

Now let’s add the ring that will catch the ball. This ring is a sprite sheet containing 5 colored rings 60 pixels wide by 65 pixels high.



First preload the sprite sheet. Add this code to the preload function.

**game.load.spritesheet(“rings”,“images/main/rings.png”,60, 65, 5);**

To place the ring in place we use the following code. This places the ring in the center horizontally and 100 pixels above the blockGroup.

Place this code in your create function just above the call to the setListeners function

**this.ring=game.add.sprite(game.world.centerX,this.blockGroup.y-100,“rings”);**

**this.ring.anchor.set(0.5,0.5);**

**Select the Ring’s Frame**

When we add a sprite that is a sprite sheet, by default it displays the first frame.

It is very easy to change the frame with one line of code

this.sprite.frame=1;
this.sprite.frame=2;

It is important to note that the first frame is 0, not 1.

To go to the first frame of a sprite use:

this.sprite.frame=0;

Let’s set up a function to change frame of the ring

**drawRing:function(color)**
**{**
**this.ring.frame=color;**
**}**

In this function, the parameter *color* indicates which frame number.

We can now select a frame based on the name of the object clicked.

We will use a switch statement to select the correct frame.

Change your ChangeColor function to look like this.

**changeColor:function(target)**
**{**
**switch(target.name)**

**{**
**case “red”:**
**this.drawRing(3);**
**break;**
**case “blue”:**
**this.drawRing(1);**
**break;**
**case “green”:**
**this.drawRing(2);**
**break;**
**case “yellow”:**
**this.drawRing(4);**
**break;**
**}**
**}**

**Testing the Ring**

Your code should now look like this. New code in bold.

var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
},
create: function() {
this.blockGroup = game.add.group();

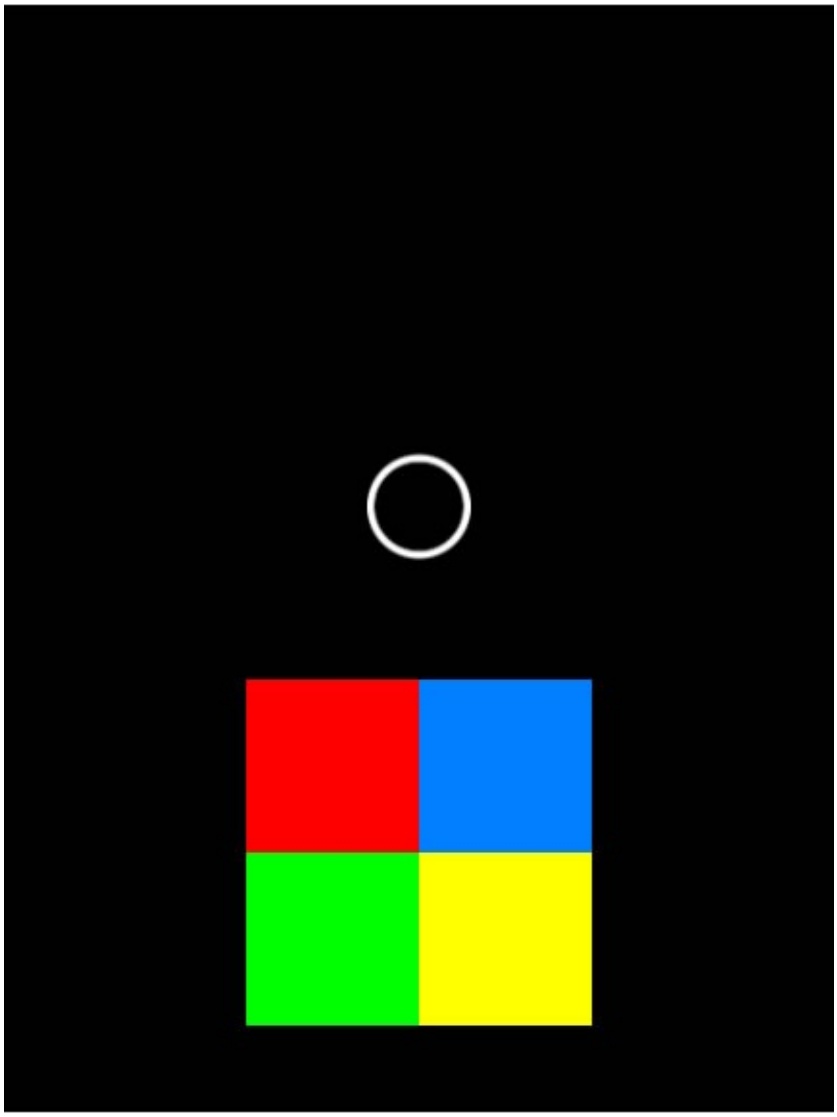
```
    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);

    this.setListeners();
```

```
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
update: function() {

}
```

}

Testing-Refresh the window. Click each of the colors

Expected result:The ring will change to the color pressed.



Resetting the Ring

Right now when we click a color the ring changes to the correct color, but there is no way currently to get back to white.

Instead of adding another button we can select white when all blocks are released, or in code terms, when a mouse up is called.

To reset the ring we can use this simple function

**resetRing:function()**
**{**
**this.ring.frame=0;**
**}**

To call the function we could add a listener to each block, but since it doesn't matter which block is released, we can simply listen to the entire game for a mouse up event.

Add this line of code to your setListeners function

**game.input.onUp.add(this.resetRing,this);**

**Testing the Mouse Up Event**

Your code should now look like this.

New code is in bold

var StateMain = {

preload: function() {
game.load.image("yellow", "images/main/blocks/yellow.png");
game.load.image("blue", "images/main/blocks/blue.png");
game.load.image("red", "images/main/blocks/red.png");
game.load.image("green", "images/main/blocks/green.png");
game.load.spritesheet("rings", "images/main/rings.png", 60, 65, 5);
},
create: function() {
this.blockGroup = game.add.group();

```
    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);

    this.setListeners();
    },
```

```
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
```

update: function() {

}

}

Testing-Refresh the window. Click each of the colors, and release.

Expected result:The ring will change to the color pressed and to white when the mouse is released.

**Add the Ball**

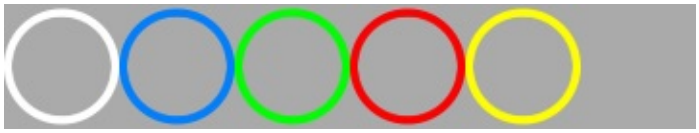
Now that we have the ring and inputs working, let's add the ball.

The ball is a sprite sheet just like the rings, with the frames of the ball sprite sheet lining up the same as the rings.

The individual images are 35 pixels wide and high.

[ball sprite sheet]

[ring sprite sheet]



Add the sprite sheet in your preload function

**game.load.spritesheet('balls', 'images/main/balls.png', 35, 35, 5);**

In the create function we add the ball to the canvas

**this.ball=game.add.sprite(0,0,“balls”);**

We are placing it in the corner temporarily. In the next step we will randomly place the ball.

**Using random numbers**

Phaser has a nice built in random number generator.

With most languages to get a number between 1 and 10 we would call something like:

**var num=Math.floor((Math.random() * 10) + 1);**

Which works fine, but with Phaser we can call:

**var num=game.rnd.integerInRange(1, 10);**

Either code is acceptable, but the Phaser example is just a bit cleaner and easier to read.

Now we will add the function to randomly set the position of the ball and randomly set the color of the ball

This will place the ball at a position between 0 and 100 pixels above the top of the game and somewhere between 100 pixels to the left edge of the game to 100 pixels to the right edge of the game. It will also set a random color for the ball.

**resetBall:function()**

**{**

**var colorIndex = game.rnd.integerInRange(0, 5);**

**var yy=game.rnd.integerInRange(0,100);**

**var xx=game.rnd.integerInRange(-100,game.width+100);**
**this.ball.frame=colorIndex;**
**this.ball.x=xx;**
**this.ball.y=yy;**
**}**

In the create function call the resetBall function

**this.resetBall();**

**Testing the Mouse Up Event**

Your code should now look like this.

New code is in bold.

var StateMain={
preload:function()
{
game.load.image(“yellow”,“images/main/blocks/yellow.png”);
game.load.image(“blue”,“images/main/blocks/blue.png”);
game.load.image(“red”,“images/main/blocks/red.png”);
game.load.image(“green”,“images/main/blocks/green.png”);
**game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);**
},
create:function()
{
this.red=game.add.sprite(0,0,“red”);
this.blue=game.add.sprite(100,0,“blue”);
this.green=game.add.sprite(0,100,“green”);
this.yellow=game.add.sprite(100,100,“yellow”);
this.red.inputEnabled =true;

```
this.blue.inputEnabled =true;
this.green.inputEnabled =true;
this.yellow.inputEnabled =true;

this.red.name=“red”;
this.blue.name=“blue”;
this.green.name=“green”;
this.yellow.name=“yellow”;
this.red.events.onInputDown.add(this.changeColor,this);
this.blue.events.onInputDown.add(this.changeColor,this);
this.green.events.onInputDown.add(this.changeColor,this);
this.yellow.events.onInputDown.add(this.changeColor,this);

this.blockGroup=game.add.group();
this.blockGroup.add(this.red);
this.blockGroup.add(this.blue);
this.blockGroup.add(this.green);
this.blockGroup.add(this.yellow);

this.blockGroup.x=game.world.centerX-this.blockGroup.width/2;
this.blockGroup.y=game.height-group.height-50;

this.ball=game.add.sprite(0,0,“balls”);
this.setListeners();
this.resetBall();
},
function setListeners()
{
game.input.onUp.add(this.resetRing,this);
this.red.events.onInputDown.add(this.changeColor,this);
this.blue.events.onInputDown.add(this.changeColor,this);
this.green.events.onInputDown.add(this.changeColor,this);
```

```
this.yellow.events.onInputDown.add(this.changeColor,this);
},
drawRing:function(color)
{
this.ring.frame=color;
},
changeColor:function(target)
{
switch(target.name)
{
case “red”:
this.drawRing(3);
break;
case “blue”:
this.drawRing(1);
break;
case “green”:
this.drawRing(2);
break;
case “yellow”:
this.drawRing(4);
break;
}
},
resetRing:function()
{
this.ring.frame=0;
},
resetBall:function()
{
var colorIndex = game.rnd.integerInRange(0, 6);
var yy=game.rnd.integerInRange(0,100);
```

**var xx=game.rnd.integerInRange(-100,game.width+100);**

**this.ball.frame=colorIndex;**

**this.ball.x=xx;**

**this.ball.y=yy;**

**},**

update:function()

{

}

}

Important-after testing change the line

**var yy=game.rnd.integerInRange(0,100);**

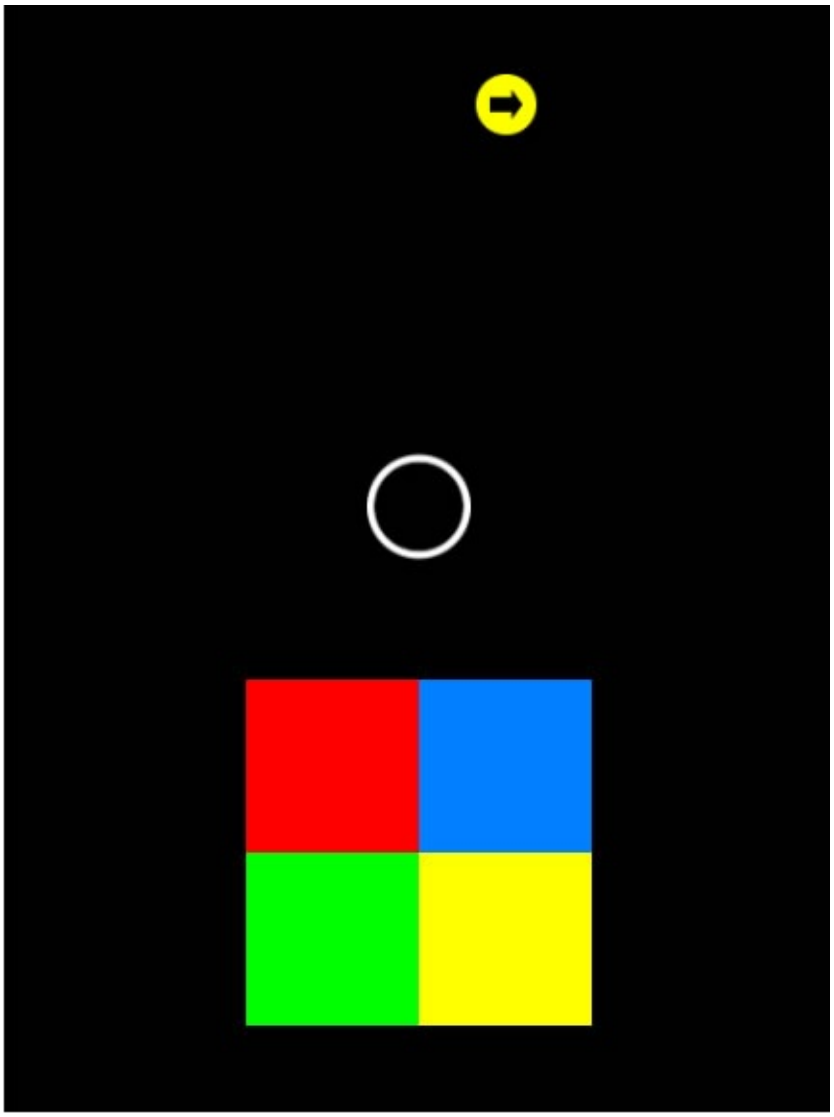
To

**var yy=-game.rnd.integerInRange(0,100);**

note the minus symbol is the only change, this will set the ball in a random position off screen.

Testing-Refresh the window.

Expected result:The ball will appear in a random position and a random color each time the window is refreshed.



**Adding Physics**

Phaser comes with several Physics engines. The one that seems most popular, and the one we will be using is called Arcade.

The first thing needed to be able to use Physics is the start the system.

Add this to the top of your create function.

**game.physics.startSystem(Phaser.Physics.ARCADE);**

The second step is to enable the objects you want use physics

For a single object you can use:

**game.physics.arcade.enable(object);**

or for multiple objects you can use an array of objects:

**game.physics.arcade.enable([object1,object2]);**

Place this code just after the ball is added to the canvas:

**game.physics.arcade.enable(this.ball);**

We need to move the ball from where ever it is to the ring.

We can use this built in physics function:

**game.physics.arcade.moveToXY(Object, TargetX, TargetY, speed);**

This function moves an object from its current position to the target x and y, but it also returns a number that represents the angle that the object is traveling.

Add this code to the resetBall function

**var rot=game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);**

**this.ball.rotation=rot;**

We also need to declare the speed variable.

At the top of the create function add this code:

**this.speed=200;**

Right now the ball speed will remain at 200, but we will be changing it later.

**Testing the Physics**

Your code should now look like this.

New code is in bold.

```
var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
},
create: function() {
this.speed = 200;

game.physics.startSystem(Phaser.Physics.ARCADE);
```

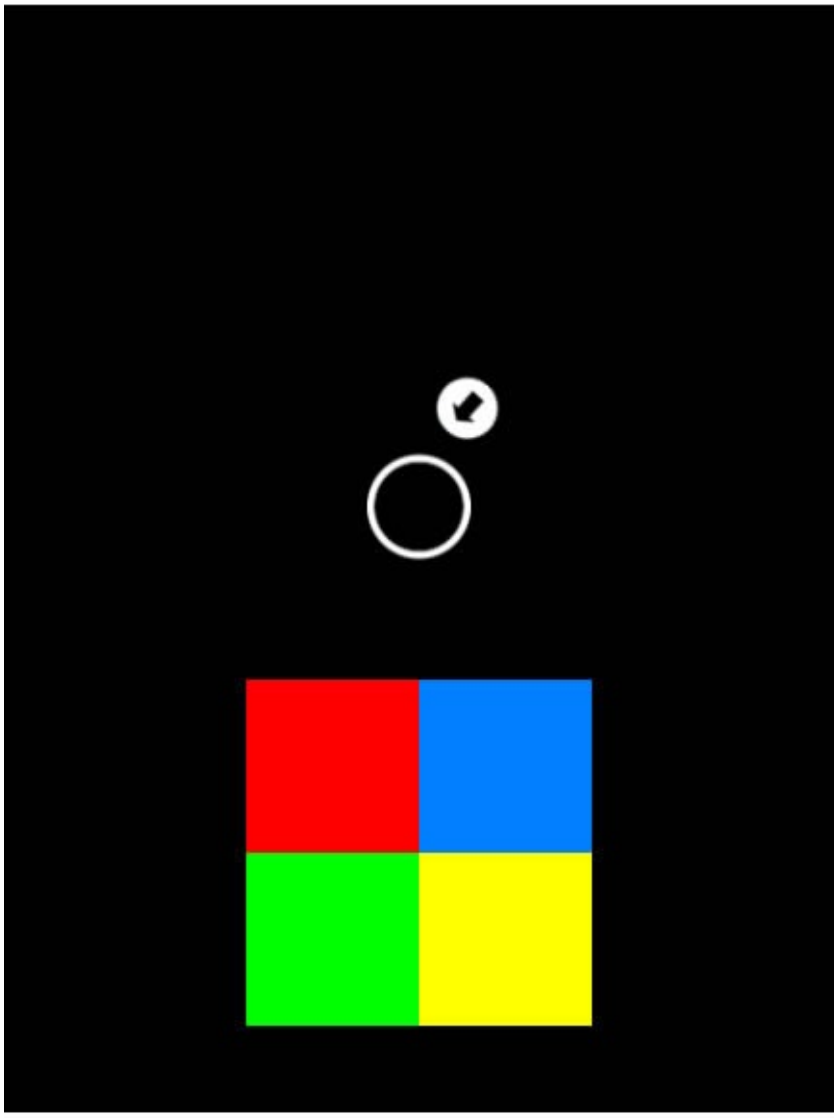
```
    this.blockGroup = game.add.group();

    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);
```

```
this.ball = game.add.sprite(0, 0, “balls”);
game.physics.arcade.enable(this.ball);


this.setListeners();
this.resetBall();
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
```

```
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {

}

}
```

Testing-Refresh the window.

Expected result:The ball will appear in a random position and a random color and fly to the ring and continue past the ring, with the arrow pointing towards the ring.



**Using the Update Function**

The update function constantly executes any code placed in it. You should be careful not to overload this function with too much code as it can slow down your game.

We will use this update function to check the position of the ball and if it is inside the ring we will reset the ball.

The first thing we will do is measure the distance between the ball and the ring. We do this by subtracting both the y and x position of the ring from the x and y position of the ball.

Place this code inside your update function

**var diffX = Math.abs(this.ring.x - this.ball.x);**

**var diffY = Math.abs(this.ring.y - this.ball.y);**

This will get the distance between the ring and the ball by subtracting the x or y positions of each object.

We are using Math.abs to get the absolute value, which simply means that if a number is negative it turns positive.

So the absolute value of -10 is 10.

Then we can check to see if the variables fall below a certain value

**if (diffX < 10 && diffY < 10) {**

**}**

If both values are less than 10 then the first thing we will do is to stop the ball by setting the velocity to 0. While we did not explicitly set the velocity before, it was set by the MoveTo method we called earlier.

The format for velocity on a sprite is

sprite.body.velocity.setTo(x_velocity, y_velocity);

Add this code inside the if statement

**this.ball.body.velocity.setTo(0, 0);**

We also want to check to see if the color of the ring is the same color as the ball

We can do this by comparing frames

**if (this.ring.frame == this.ball.frame) {**

**}**

If the ring’s and ball’s color matches then we just call this.resetBall();

The update function should look now like this:

```
update: function () {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
this.resetBall();
}
}
}
```

**Testing Reseting the Ball**

Your code should now look like this.

New code is in bold.

```
var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
```

```
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
},
create: function() {
this.speed = 200;

game.physics.startSystem(Phaser.Physics.ARCADE);

this.blockGroup = game.add.group();

this.red = game.add.sprite(0, 0, “red”);
this.blue = game.add.sprite(100, 0, “blue”);
this.green = game.add.sprite(0, 100, “green”);
this.yellow = game.add.sprite(100, 100, “yellow”);

this.red.inputEnabled = true;
this.blue.inputEnabled = true;
this.green.inputEnabled = true;
this.yellow.inputEnabled = true;

this.red.name = “red”;
this.blue.name = “blue”;
this.green.name = “green”;
this.yellow.name = “yellow”;



this.blockGroup.add(this.red);
this.blockGroup.add(this.blue);
this.blockGroup.add(this.green);
```

```
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group's width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, "rings");
    this.ring.anchor.set(0.5, 0.5);

    this.ball = game.add.sprite(0, 0, "balls");
    game.physics.arcade.enable(this.ball);



    this.setListeners();
    this.resetBall();
    },
    setListeners: function() {
    this.red.events.onInputDown.add(this.changeColor, this);
    this.blue.events.onInputDown.add(this.changeColor, this);
    this.green.events.onInputDown.add(this.changeColor, this);
    this.yellow.events.onInputDown.add(this.changeColor, this);
    game.input.onUp.add(this.resetRing, this);
    },
    changeColor: function(target) {
    switch (target.name) {
    case "red":
    this.drawRing(3);
    break;

    case "blue":
```

```
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
```

```
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
this.resetBall();
}
}

}

}
```

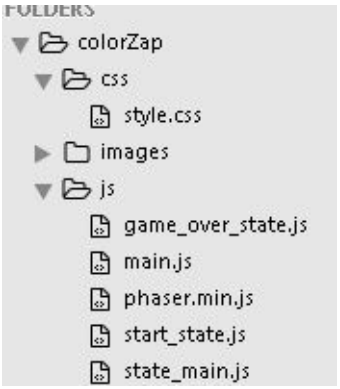
Testing-Refresh the window. Press the color buttons.

Expected result:The ball will appear in a random position and a random color and fly to the ring, if the color of the ball matches the color of the ring. If not it will simple stop in place until the correct color is pressed.

**Game Over Screen**

At this point we need to be able to stop the game and allow the user to play again. For that we will need the game over screen, or in Phaser terms the gameOver state.

In the js folder make a new .js file called **game_over_state.js**




Create the blank state

**var StateOver = {**

**preload: function () {**

**},**
**create: function () {**

**}**
**, update: function () {**
**}**
**}**

First let’s make a button to let the player restart the game

Place this code in the create function

**this.buttonPlayAgain = game.add.button(game.world.centerX, game.height-200, ‘buttons’, this.replay, this, 1, 0, 1);**
**this.buttonPlayAgain.anchor.set(0.5,0.5);**

This button uses the same sprite sheet that we used for the button on the title screen, so there is no need to preload the image here.

This time we will be using frames 0 and 1.


Now let’s add the replay function

**replay:function()**

**{**

**game.state.start(“StateMain”);**

**}**

Your game_over_state file should now look like this.

**var StateOver = {**

**preload: function () {**

**},**

**create: function () {**

**this.buttonPlayAgain = game.add.button(game.world.centerX, game.height-200, ‘buttons’, this.replay, this, 1, 0, 1);**

**this.buttonPlayAgain.anchor.set(0.5,0.5);**

**}**

**, update: function () {**

**},**

**replay:function()**

**{**

**game.state.start(“StateMain”);**

**}**

**}**

Now we need to add the new state into the game.

In the main.js file add the following code:

**game.state.add(“StateOver”,StateOver);**

Your code in the main.js file should now look like this:

var game;

window.onload = function()

{

game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);

game.state.add(“StartState”,StartState);

**game.state.add(“StateOver”,StateOver);**

game.state.start(“StartState”);

}

In the index.html file we need to add the new javaScript file to the code

Add the line **<script src=“js/game_over_state.js”></script>** to your index.html file so that it now looks like this:

<!DOCTYPE html>

```
<html lang="">
<head>
<meta charset="UTF-8">
<title>Game Title Here</title>
<script src="js/phaser.min.js"></script>
<script src="js/main.js"></script>
<script src="js/state_main.js"></script>
<script src="js/game_over_state.js"></script>
<script src="js/start_state.js"></script>
</head>
<body>

<div id="ph_game"></div>

</body>
</html>
```

Calling the Game Over

In the update function we check to see if the ball frame matches the ring frame and if it does we reset the game

If not we need to call the game over screen.

We simply need to add this code to the if statement

```
else
{
game.state.start("StateOver");
}
```

So it now looks like this:

```
if (this.ring.frame == this.ball.frame) {
this.resetBall();
} else {
game.state.start(“StateOver”);
}
```

**Testing the game over**

The main state should now look like this

New code is in bold.

```
var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
},
create: function() {
this.speed = 200;

game.physics.startSystem(Phaser.Physics.ARCADE);

this.blockGroup = game.add.group();
```

```
    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);

    this.ball = game.add.sprite(0, 0, “balls”);
    game.physics.arcade.enable(this.ball);
```

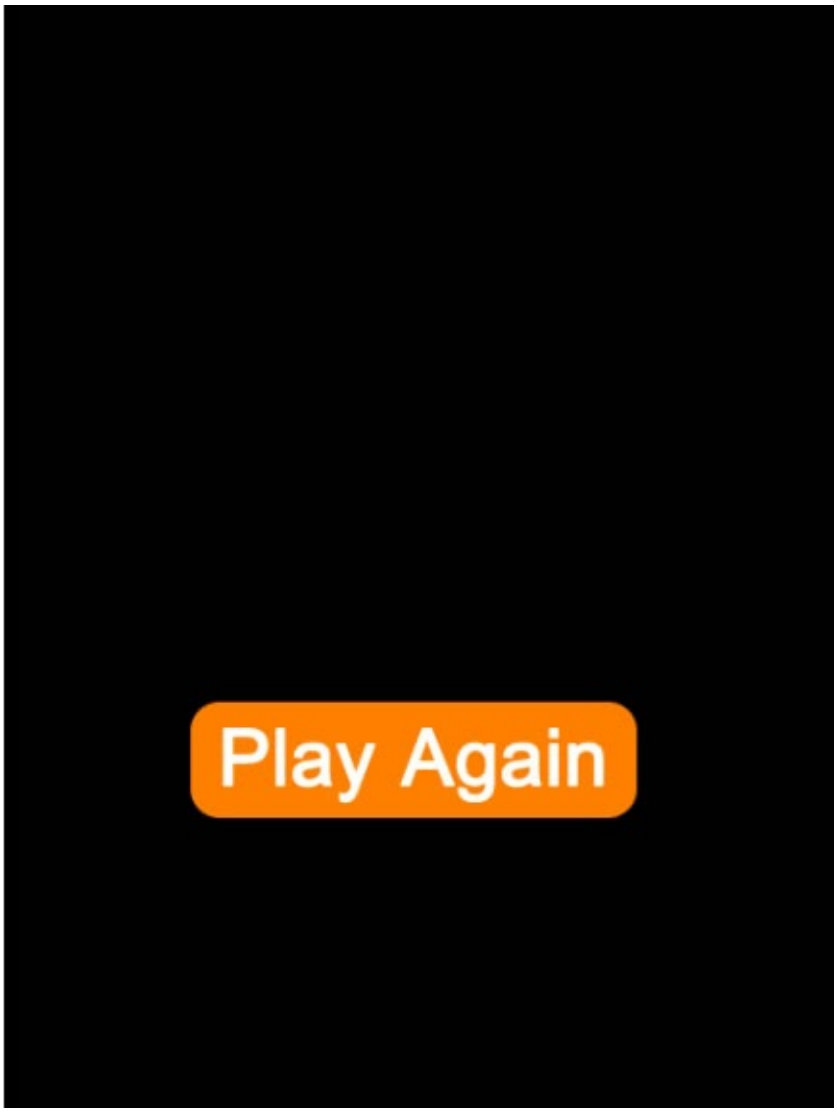
```
this.setListeners();
this.resetBall();
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
```

```
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
this.resetBall();
} else {
game.state.start("StateOver");
}
}

}
```

}

Testing-Refresh the game. Press the squares to NOT match the ball.
Expected result-The game over screen will appear

Testing-Press the play again button
Expected result-The game will reset




**Adding text to your game**

Now we have a playable game! Let’s take the next step by adding score.

First we need to have a text box to hold the current score.

You can add text to a game in Phaser using a single line of code.

**game.add.text(x,y,“message”);**

That works fine for static text, but will a bit more of typing we can have a lot more control over text.

First let’s add the text box

This will place the number zero, 150 pixels from the top, and in the center horizontally.

Add this code near the bottom of your create function but above this.setListeners();

**this.scoreText = game.add.text(game.world.centerX, 150, “0”);**

This will not show up if you run the game now because text in Phaser is black by default, and so is the background color, so even though the text is there, it cannot be seen.

We can use the fill property to change the text color. The property take hex colors like #ffffff(white) #ff0000(red) #000000(black)

If you are not familiar with hex colors you can find an online color picker to get the color you want.

Place this code just after the code above.

**this.scoreText.fill = “#ffffff”;**

To change the size of the text we can set the fontSize property. This is set in pixels. For normal reading text we generally use between 12 and 16 pixels.

But sometimes in games we need big text that will stand out, so we use 64 pixels here.

Add this line of text below the last two lines you just placed in the create function.

**this.scoreText.fontSize = 64;**

Because we are placing the text in the center of the screen, we also need to set the anchor on the text.

**this.scoreText.anchor.set(0.5, 0.5);**

That will give us a nice large number that is easy to see place in the top middle of the screen.

To make sure the player know that it is the score, we can place a label

This code will place the label 50 pixels above the text, and set it at half the size of the score.

**this.scoreLabel = game.add.text(game.world.centerX, 100, “score”);**
**this.scoreLabel.fill = “#ffffff”;**
**this.scoreLabel.fontSize = 32;**
**this.scoreLabel.anchor.set(0.5, 0.5);**

**Calculating the Score**

Now that we have a place to display the score, we can start awarding the score to the player.

In the main.js file declare a variable named score

**var score;**

By declaring the variable in main.js we will have access to that variable in every state.

At the top of your create statement place this code:

**score=0;**

This will reset the score to 0 every time the StateMain is loaded.

In the update function find the if then statement that checks to see if the ring frame and the ball frame match

```
if (this.ring.frame == this.ball.frame) {
this.resetBall();
}
```

Inside the if then statement let's up the score by one.

```
if (this.ring.frame == this.ball.frame) {
score++;
this.resetBall();
}
```

Next let's add a line to display the updated score

```
if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.resetBall();
}
```

**Testing the score**

The main.js file should now look like this

New Code is in bold.

```
var game;
var score;
window.onload = function()
{
game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);
game.state.add(“StartState”,StartState);
game.state.add(“StateMain”,StateMain);
game.state.add(“StateOver”,StateOver);
game.state.start(“StartState”);
}
```

The state_main.js should now look like this

New code is in bold.

```
var StateMain = {
```

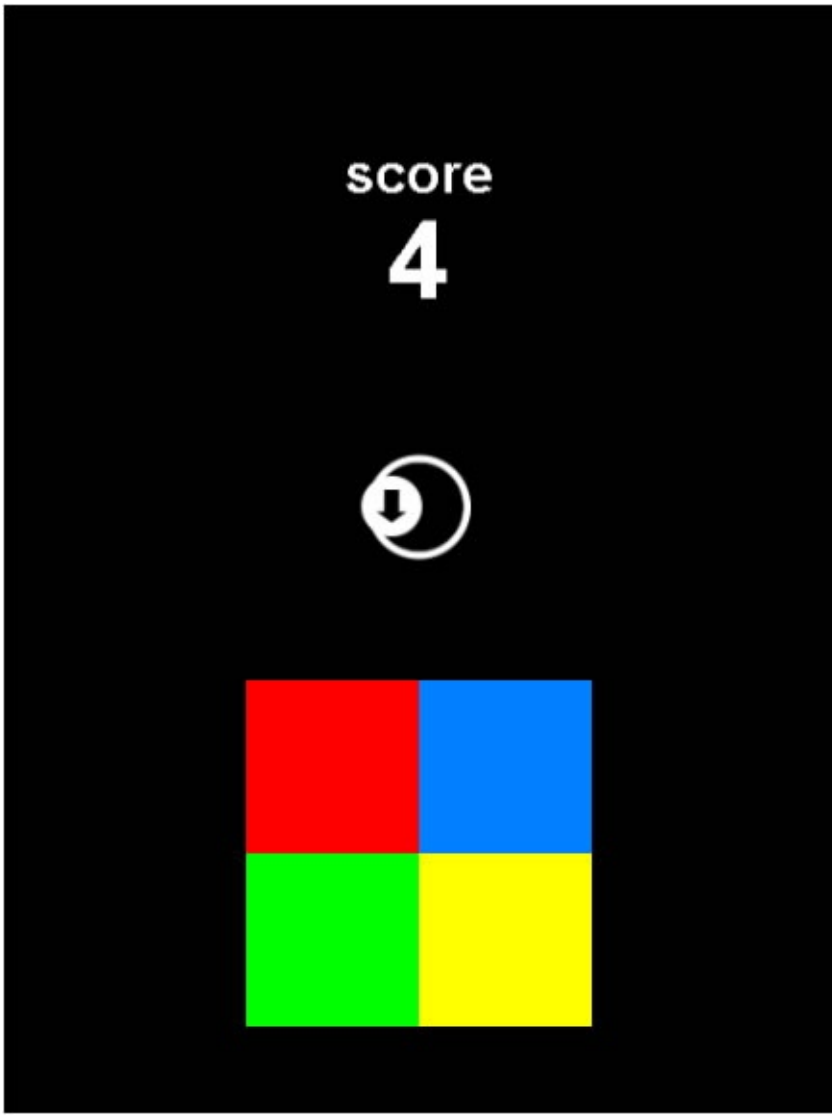
```
preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
},
create: function() {
this.speed = 200;
score=0;
game.physics.startSystem(Phaser.Physics.ARCADE);

this.blockGroup = game.add.group();

this.red = game.add.sprite(0, 0, “red”);
this.blue = game.add.sprite(100, 0, “blue”);
this.green = game.add.sprite(0, 100, “green”);
this.yellow = game.add.sprite(100, 100, “yellow”);

this.red.inputEnabled = true;
this.blue.inputEnabled = true;
this.green.inputEnabled = true;
this.yellow.inputEnabled = true;

this.red.name = “red”;
this.blue.name = “blue”;
this.green.name = “green”;
this.yellow.name = “yellow”;



this.blockGroup.add(this.red);
```

```
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group's width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, "rings");
    this.ring.anchor.set(0.5, 0.5);

    this.ball = game.add.sprite(0, 0, "balls");
    game.physics.arcade.enable(this.ball);

    this.scoreText = game.add.text(game.world.centerX, 150, "0");
    this.scoreText.fill = "#ffffff";
    this.scoreText.fontSize = 64;
    this.scoreText.anchor.set(0.5, 0.5);

    this.scoreLabel = game.add.text(game.world.centerX, 100, "score");
    this.scoreLabel.fill = "#ffffff";
    this.scoreLabel.fontSize = 32;
    this.scoreLabel.anchor.set(0.5, 0.5);

    this.setListeners();
    this.resetBall();
    },
    setListeners: function() {
    this.red.events.onInputDown.add(this.changeColor, this);
    this.blue.events.onInputDown.add(this.changeColor, this);
```

```
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
```

```
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.resetBall();
} else {
game.state.start("StateOver");
}
}

}

}
```

Testing-Refresh the game. Press the squares to match the ball.
Expected result-The score will increase and display at the top of the game.




To add a little challenge to the game let’s speed up the ball every time the player gets a point.

Earlier we added the speed variable to the create function, so now Let’s add two variables just below that.

this.speed = 200;

**this.maxSpeed = 400;**

**this.incSpeed = 25;**

The first new variable is maxSpeed, this will be the upper limit that the ball's speed can reach, without it, the speed would reach an impossible speed. We want to make the game challenging but not impossible.

The second new variable is incSpeed, this is how much the ball will speed up every time the player scores.

Now let's update the if then statement that checks if the ball matches the ring and add the code that will increase the speed

Here is how the if then statement should look, new code in bold.

```
if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed+=this.incSpeed;
if(this.speed>this.maxSpeed)
{
this.speed=this.maxSpeed;
}
this.resetBall();
}
```

**Testing the speed up**

The main state should now look like this

New code is in bold.

```
var StateMain = {
```

```
preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
},
create: function() {
this.speed = 200;
score = 0;
this.maxSpeed = 400;
this.incSpeed = 25;

game.physics.startSystem(Phaser.Physics.ARCADE);

this.blockGroup = game.add.group();

this.red = game.add.sprite(0, 0, “red”);
this.blue = game.add.sprite(100, 0, “blue”);
this.green = game.add.sprite(0, 100, “green”);
this.yellow = game.add.sprite(100, 100, “yellow”);

this.red.inputEnabled = true;
this.blue.inputEnabled = true;
this.green.inputEnabled = true;
this.yellow.inputEnabled = true;

this.red.name = “red”;
this.blue.name = “blue”;
this.green.name = “green”;
```

```
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);

    this.ball = game.add.sprite(0, 0, “balls”);
    game.physics.arcade.enable(this.ball);

    this.scoreText = game.add.text(game.world.centerX, 150, “0”);
    this.scoreText.fill = “#ffffff”;
    this.scoreText.fontSize = 64;
    this.scoreText.anchor.set(0.5, 0.5);

    this.scoreLabel = game.add.text(game.world.centerX, 100, “score”);
    this.scoreLabel.fill = “#ffffff”;
    this.scoreLabel.fontSize = 32;
    this.scoreLabel.anchor.set(0.5, 0.5);

    this.setListeners();
    this.resetBall();
```

```
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
```

```
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed += this.incSpeed;
if (this.speed > this.maxSpeed) {
this.speed = this.maxSpeed;
}
this.resetBall();
} else {
game.state.start("StateOver");
}
}
```

}

}

Testing-Refresh the game. Press the squares to match the ball.
Expected result-The ball will increase speed for every correct match.

Adding Sounds

Adding sound to phaser is fairly simple, but it is a three step process.

The steps are

1. Preload the sound into the library
2. Define the sound in an object
3. Play the sound

We will be using two sounds, one for points and the other for losing the game

First lets preload the sounds.

To preload a sound in Phaser we use the following format:

game.load.audio(‘soundName’, ‘pathToSoundFile’);

Place these lines of code in your preload function in stateMain.js

**game.load.audio(‘points’, ‘sounds/points.mp3’);**
**game.load.audio(“lose”, “sounds/gameOver.mp3”);**

This loads the sound into the library the same way as we did earlier with images.

Next lets create the sound objects.

To create a sound object in phaser use the following formula:

this.soundObjectName = game.add.audio(‘libraryKey’);

Add these lines at the top of your create statement to define the sounds.

**this.lose = game.add.audio(‘lose’);**
**this.points = game.add.audio(“points”);**

These lines create sound objects based on the library key.

The third and final step is play the sound like this:

this.soundObjectName.play();

We need to play the sound in our if the statement that we’ve been working with in the last few steps.

**this.points.play();** should be placed where the score is increased, and the **this.lose.play();** should be called right before we change to the game over screen.

The if then statement should now look list this.

```
if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed += this.incSpeed;
this.points.play();
this.resetBall();
```

} else {
**this.lose.play();**
game.state.start(“StateOver”);
}

**Testing the sound**

The main state should now look like this

New code is in bold.

var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
**game.load.audio(‘points’, ‘sounds/points.mp3’);**
**game.load.audio(“lose”, “sounds/gameOver.mp3”);**
},
create: function() {
this.speed = 200;
score = 0;
this.maxSpeed = 400;
this.incSpeed = 25;

game.physics.startSystem(Phaser.Physics.ARCADE);

```
    this.blockGroup = game.add.group();

    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);
```

```
this.ball = game.add.sprite(0, 0, “balls”);
game.physics.arcade.enable(this.ball);

this.scoreText = game.add.text(game.world.centerX, 150, “0”);
this.scoreText.fill = “#ffffff”;
this.scoreText.fontSize = 64;
this.scoreText.anchor.set(0.5, 0.5);

this.scoreLabel = game.add.text(game.world.centerX, 100, “score”);
this.scoreLabel.fill = “#ffffff”;
this.scoreLabel.fontSize = 32;
this.scoreLabel.anchor.set(0.5, 0.5);

this.lose = game.add.audio(‘lose’);
this.points = game.add.audio(“points”);

this.setListeners();
this.resetBall();
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;
```

```
case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
update: function() {
```

```
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed += this.incSpeed;
if (this.speed > this.maxSpeed) {
this.speed = this.maxSpeed;
}
this.points.play();
this.resetBall();
} else {
this.lose.play();
game.state.start("StateOver");
}
}

}

}
```

Testing-Refresh the game. Press the squares to match the ball, and then press squares to not match the ball

Expected result-Sounds will play for correct match and for losing the game.

**Toggling Sound**

Now that we have the sounds in the game, it will be nice to give the player the option of toggling on and off the sound.

For that we need some toggle buttons. Phaser doesn't have any built in toggle buttons, like it does regular buttons, but we can create our own.

First let's add the toggle buttons as a sprite sheet, in the same way we did the other buttons.

Add this code to then end of your preload function of state_main.js

**game.load.spritesheet("soundButtons", "images/ui/soundButtons.png", 32, 32);**

In the create statement let's define our toggle button as a sprite.

**this.soundButton = game.add.sprite(10, 10, "soundButtons");**

By placing the button at x=10, y=10 (10 pixels from top, and 10 from the left), we can insure that is out of the way of everything else, and because we are using the top left corner, it will be the same regardless of screen size.

Next we have to tell Phaser that we want to listen for events on the sprite.

**this.soundButton.inputEnabled = true;**

And then add the listener itself inside the setListeners function

**this.soundButton.events.onInputDown.add(this.toggleSound, this);**

We need a variable to hold the state of the sound being on or off. It is important to declare this in main.js instead of stateMain.js. The reason for this is that if we only declare the variable in stateMain.js, it will reset every time the state is loaded. That means the player wanted the sound off, they would have to turn off the sound every time they pressed play again.

In main.js place the following code.

**var soundOn=true;**

Now let’s create the new function called toggleSound in state_main.js

```
toggleSound: function () {
soundOn = !soundOn;
if (soundOn == true) {
this.soundButton.frame = 0;
} else {
this.soundButton.frame = 1;
}
}
```

The first line of code **soundOn=!soundOn** reverses the boolean value. It turns false to true, and true to false. The if then statement, sets the frame of the sound button based on the value of soundOn, showing the correct on or off sound symbol

[images of sound buttons]

We also need to change the code were we play the sounds. We will place an if-then statement around each .play() and only call the code if the soundOn variable is equal to true.

For example

```
if (soundOn == true) {
this.points.play();
}
```

The last thing to do is to check if the sound is on when the stateMain is loaded. If it is not on, we need to set the button frame to the off image. In the create function add the following:

```
if (soundOn == false) {
```

**this.soundButton.frame = 1;**
**}**

**Testing the sound**

The main_state.js should now look like this

New code is in bold.

**main_state.js**

var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
game.load.audio(‘points’, ‘sounds/points.mp3’);
game.load.audio(“lose”, “sounds/gameOver.mp3”);
**game.load.spritesheet(“soundButtons”, “images/ui/soundButtons.png”, 32, 32);**
},
create: function() {
this.speed = 200;
score = 0;

```
    this.maxSpeed = 400;
    this.incSpeed = 25;

    game.physics.startSystem(Phaser.Physics.ARCADE);

    this.blockGroup = game.add.group();

    this.red = game.add.sprite(0, 0, “red”);
    this.blue = game.add.sprite(100, 0, “blue”);
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
```

```
this.blockGroup.y = game.height - this.blockGroup.height - 50;

this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
this.ring.anchor.set(0.5, 0.5);

this.ball = game.add.sprite(0, 0, “balls”);
game.physics.arcade.enable(this.ball);

this.scoreText = game.add.text(game.world.centerX, 150, “0”);
this.scoreText.fill = “#ffffff”;
this.scoreText.fontSize = 64;
this.scoreText.anchor.set(0.5, 0.5);

this.scoreLabel = game.add.text(game.world.centerX, 100, “score”);
this.scoreLabel.fill = “#ffffff”;
this.scoreLabel.fontSize = 32;
this.scoreLabel.anchor.set(0.5, 0.5);

this.lose = game.add.audio(‘lose’);
this.points = game.add.audio(“points”);

this.soundButton = game.add.sprite(10, 10, “soundButtons”);
this.soundButton.inputEnabled = true;



this.setListeners();
this.resetBall();

if (soundOn == false) {
this.soundButton.frame = 1;
}
},
```

```
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
this.soundButton.events.onInputDown.add(this.toggleSound, this);
},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
```

```
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
toggleSound: function() {
soundOn = !soundOn;
if (soundOn == true) {
this.soundButton.frame = 0;
} else {
this.soundButton.frame = 1;
}
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed += this.incSpeed;
```

```
if (this.speed > this.maxSpeed) {
this.speed = this.maxSpeed;
}
if (soundOn == true) {
this.points.play();
}

this.resetBall();
} else {
if (soundOn == true) {
this.lose.play();
}

game.state.start(“StateOver”);
}
}

}

}
```

**main.js**

```
var game;
var score;
var soundOn=true;
window.onload = function()
{
game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);
game.state.add(“StartState”,StartState);
game.state.add(“StateMain”,StateMain);
game.state.add(“StateOver”,StateOver);
```

```
game.state.start(“StartState”);
}
```

Testing-Refresh the game. Press the sound toggle button to turn on and off sound
Expected result-Sounds will play only when sound button is on.

# CHAPTER EIGHT

## *MobilePrep*

### Setting up for mobile

Now we have a fully functional game that will do well on any website. That's great! However, with most of the world looking at your game through a mobile device, wouldn't it be nice if they could play your game, as if they had downloaded it? Well, with a few simple code additions you can make that happen. Notice, I didn't write *code changes.* We can use the exact same code for the game, and just add a few sections to make it look close to a mobile game.

There are a few things that will be different. While phaser has a full screen effect built it, it has to be launched by a button. Also, it only works on Android, and not all devices support it. That's why I don't recommend going full screen, but rather full browser. Yes there will be the address bar at the top of the screen, but this will insure it works on all devices.

Make a separate file for mobile

We want to reuse as much code as possible, and I've written all the javaScript in the game to be used on both computer and device. The exception is the main html file. The reason for this is that on the desktop version, I will mostly have the game on a page with other information, or possibly ads. In any case, it will be a part of something bigger, whereas on a phone, it will be take up the entire browser. To do that we will also be adding some html and css that is mobile specific.

Here is the standard html file I use for the mobile versions of the game.

Create a new file called mobile.html

You can copy the code from index.html and just add the two lines in bold below.

```
<!DOCTYPE html>
<html lang="">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="user-scalable=0, initial-scale=1,minimum-scale=1, maximum-scale=1, width=device-width, minimal-ui=1">
<title>Color Zap</title>
script src="js/phaser.min.js"></script>
<script src="js/main.js"></script>
<script src="js/start_state.js"></script>
<script src="js/state_main.js"></script>
<script src="js/game_over_state.js"></script>
<link rel="stylesheet" type="text/css" href="css/style.css">
</head>
<body>
<div id="wrongWay"></div>
<div id="ph_game"></div>

</body>
</html>
```

The main differences between the mobile version and the standard version is the addition of a div tag with the id or wrongWay, and the inclusion of a css file.

We will be using these later in the book.

**Detecting a mobile device**

The first step is to check if the player is on a mobile device, like a phone or tablet, or if they are using a non-mobile machine like a desktop or laptop.

Phaser has a built in way to detect mobile or computer, but checking this property

**game.device.desktop**

If the value is true, the player is on a computer, and if false, then on a phone or tablet.

Outside of the phaser framework we can detect mobile by checking the user agent

We can get the user agent in a string form by checking **navigator.userAgent**

It will return a string like the following.

Mozilla/5.0 (iPhone; CPU iPhone OS 9_1 like Mac OS X) AppleWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13B143 Safari/601.1

Then we only need to check for the word mobile anywhere in the string.

**var isMobile=navigator.userAgent.indexOf(“Mobile”);**

If isMobile is equal to a -1 then we are not on mobile, any other value will means it is a mobile device.

Redirect to Mobile

Ideally we should have one link to the main file, and then with a little javaScript redirect to the mobile file.

Because this is outside of the phaser framework, we need to detect mobile by using the userAgent method

In the header of the index.html file

**<script>**

**var isMobile=navigator.userAgent.indexOf(“Mobile”);**

**if (isMobile!=-1)**

**{**

**window.location=“mobile.html”;**

**}**

**</script>**

Adding some style

To make sure there are no gaps in the game, such as white lines around the edges, we can use a little css.

Make a new folder called css and a new file inside that folder style.css

Place this code inside the style.css file

```
body {
padding: 0;
margin: 0;
}
```

**Fitting the game to the screen**

If they are using a computer, then we want to keep the game dimensions the same, so it will fit nicely on a web page.

The size we set up is 480 x 640 and that will be the default game size.

We created this game size in main.js

**game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);**

However if we detect that it is a mobile device, we want to match the screen's width and height.

In main.js replace

**game=new Phaser.Game(480,640,Phaser.AUTO,"ph_game");**

With

```
var isMobile=navigator.userAgent.indexOf("Mobile");
if (isMobile==-1)
{
game=new Phaser.Game(480,640,Phaser.AUTO,"ph_game");
}
else
{
game=new Phaser.Game(window.innerWidth,window.innerHeight,Phaser.AUTO,"ph_game");
}
```

This code checks the user agent and if mobile, we then set the game size to the inner width and inner height of the screen. We are using innerWidth and innerHeight instead of width and height, because it also includes any borders and padding that may exist as well. Most of the time this is not an issue, and you can use height and width. If you get any white lines around the edges of your game, try using the inner properties instead.

**Aligning and Positioning**

You may have noticed that through out the code in main game, we didn't use may

absolute values like:

this.button.x=100;
this.button.y=300;

Instead we used relative positioning such as

**this.button.x=game.world.centerX;**

OR

**this.blockGroup.x=game.world.centerX-this.blockGroup.width/2;**

The reason we do this is to accommodate different screen sizes. Most of the time we can use this technique of offsetting elements from the center or from the edge of the screen. Sometimes in the case of larger screens such as the IPad other adjustments may be required.

**Detecting Orientation**

Usually a game has a fixed orientation, meaning it is either portrait or landscape.

Sometimes a game will have both orientations, and readjust everything when the phone flips.

Usually my games will be either one or the other. When the player flips the phone to the wrong way I display a graphic to tell them to turn the phone to the correct way for the game.

Sometimes, depending on the game, I also will pause the game until the player turns it the right way.

To set up detecting a right and wrong orientation first we need to tell Phaser what the right way is.

Here is the code for setting up a one way orientation

**game.scale.forceOrientation(landscape, portrait);**

So if you want the game to be portrait only

**game.scale.forceOrientation(false, true);**

Or if you only want landscape

**game.scale.forceOrientation(true, false);**

The forceOrientation is a little misleading, in my opinion, because it doesn't stop the game from rotating, just sets up what is right or wrong.

After defining which is correct or not, there are two listeners to handle when a player turns the device. We only want this if on mobile.

Place this code in the create statement of start_state.js

```
if (!game.device.desktop)
{
game.scale.forceOrientation(false, true);
game.scale.enterIncorrectOrientation.add(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.add(this.handleCorrect, this);
}
```

Showing wrong way graphics

Earlier we add a div tag with the id of wrongWay to the mobile html file. This is a div tag that we will show or hide based on the orientation of the game. Let’s add some style to that tag now.

In the style.css file add this code.

```
#wrongWay {
width: 100%;
height: 100%;
position: fixed;
top: 0px;
left: 0px;
background-color: white;
background-image: url(‘../images/turn.png’);
background-repeat: no-repeat;
background-position: center center;
display: none;
}
```

This will make the tag 100% of the screen, and by default the display property is set to none.

The background image looks like this


And it will show whenever the player turns the device to the landscape mode.

**Showing and hiding the wrong way image**

In the start_state.js file add these two functions.

**handleIncorrect: function () {**

**document.getElementById(‘wrongWay’).style.display = “block”;**
**}**
**, handleCorrect: function () {**

**document.getElementById(‘wrongWay’).style.display = “none”;**
**}**

This will turn on the wrongWay div when the incorrect orientation is triggered and hide it when the device is turned the correct way.

Pausing the game

If the player turns the device while in middle of the game play, we want to stop the ball until the phone or tablet is turned to the correct way.

We are going to use the same lines of code that we just put into start_state.js, in the setListeners function of state_main.js

```
if (!game.device.desktop)
{
game.scale.enterIncorrectOrientation.add(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.add(this.handleCorrect, this);
}
```

And for the callback functions

```
handleIncorrect: function () {
if (this.ball) {
if (this.ball.body) {
this.ball.body.velocity.setTo(0, 0);
}
}
}
, handleCorrect: function () {
if (this.ball) {
game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
}
}
```

These lines of code first check that the ball exists, and if it does, then removes the velocity from the ball in the case of the wrong orientation, and resets the velocity in the case of entering the correct orientation.

The orientation listeners will continue to work, even after the state is change, such as a game over occurring. This can cause unwanted behavior in our game, so it is a good idea to remove those listeners right before calling the game over screen.

Let’s make a new function called removeListeners to clean things up.

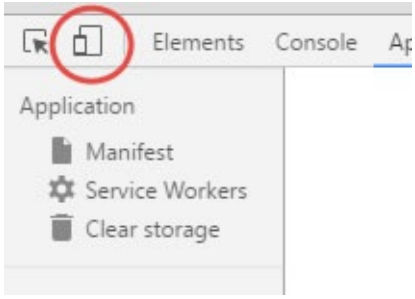
```
removeListeners:function()
{
game.scale.enterIncorrectOrientation.remove(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.remove(this.handleCorrect, this);
}
```

And call it right before changing screens

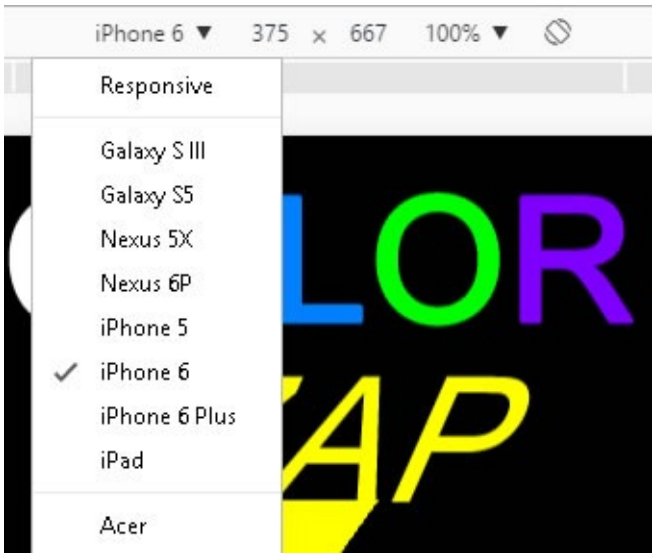
```
this.removeListeners();
game.state.start(“StateOver”);
```

Testing on Mobile

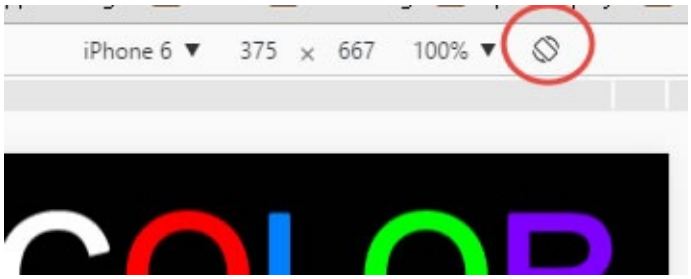
For testing on mobile, the best way is to test on a device itself. If you need to test many devices, for most of us, buying every device on the market just isn’t practical. Fortunately Google has build an emulator into it’s Chrome browser. It is accessed the same way we accessed the debugger by pressing F12.




Click on the toggle device toolbar. This will display the current page on a simulated mobile device




Use the device dropdown to select a device to emulate.




Use the rotate button to flip the phone from portrait to landscape.

**Mobile Testing**

Here is the complete code with new changes in bold

**mobile.html**

**<!DOCTYPE html>**

**<html lang=””>**

**<head>**

**<meta charset=“UTF-8”>**

**<meta name=“viewport” content=“user-scalable=0, initial-scale=1,minimum-scale=1, maximum-scale=1, width=device-width, minimal-ui=1”>**

**<title>Color Zap</title>**
**<script src="js/phaser.min.js"></script>**
**<script src="js/main.js"></script>**
**<script src="js/start_state.js"></script>**
**<script src="js/state_main.js"></script>**
**<script src="js/game_over_state.js"></script>**
**<link rel="stylesheet" type="text/css" href="css/style.css">**
**</head>**
**<body>**
** **
** **

**</body>**
**</html>**

**index.html**

<!DOCTYPE html>
<html lang="">

<head>
<meta charset="UTF-8">
<title>Color Zap</title>

<script>
**var isMobile=navigator.userAgent.indexOf("Mobile");**
**if (isMobile!=-1)**
**{**
**window.location="mobile.html";**
**}**

**</script>**

<script src=“js/phaser.min.js”></script>
<script src=“js/main.js”></script>
<script src=“js/start_state.js”></script>
<script src=“js/state_main.js”></script>
<script src=“js/game_over_state.js”></script>
</head>

<body>

</body>

</html>

**style.css**

**body {**
**padding: 0;**
**margin: 0;**
**}**

**#wrongWay {**
**width: 100%;**
**height: 100%;**
**position: fixed;**
**top: 0px;**
**left: 0px;**
**background-color: white;**
**background-image: url(‘../images/turn.png’);**

**background-repeat: no-repeat;**

**background-position: center center;**

**display: none;**

**}**

**main.js**

var game;

var score;

var soundOn=true;

window.onload = function()

{

**var isMobile=navigator.userAgent.indexOf(“Mobile”);**

**if (isMobile==-1)**

**{**

**game=new Phaser.Game(480,640,Phaser.AUTO,“ph_game”);**

**}**

**else**

**{**

**game=new Phaser.Game(window.innerWidth,window.innerHeight,Phaser.AUTO,“ph_game”);**

**}**

game.state.add(“StartState”,StartState);

game.state.add(“StateMain”,StateMain);

game.state.add(“StateOver”,StateOver);

game.state.start(“StartState”);

}

**start_state.js**

var StartState = {

preload: function () {

```
game.load.image(“gameLogo”, “images/title/gameLogo.png”);
game.load.spritesheet(“buttons”, “images/ui/buttons.png”, 264, 74);
},
create: function () {
this.logo = game.add.sprite(game.world.centerX, 180, “gameLogo”);
this.logo.anchor.set(0.5,0.5);

this.buttonStart = game.add.button(game.world.centerX, game.world.height-100, ‘buttons’, this.startTheGame, this, 7, 6, 7);
this.buttonStart.anchor.set(0.5,0.5);

if (!game.device.desktop)
{
game.scale.forceOrientation(false, true);
game.scale.enterIncorrectOrientation.add(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.add(this.handleCorrect, this);
}

}
, update: function () {

},
startTheGame: function () {

game.state.start(“StateMain”);

},
handleIncorrect: function () {

document.getElementById(‘wrongWay’).style.display = “block”;
}
, handleCorrect: function () {
```

**document.getElementById(‘wrongWay’).style.display = “none”;**
**}**
}

**state_main.js**

var StateMain = {

preload: function() {
game.load.image(“yellow”, “images/main/blocks/yellow.png”);
game.load.image(“blue”, “images/main/blocks/blue.png”);
game.load.image(“red”, “images/main/blocks/red.png”);
game.load.image(“green”, “images/main/blocks/green.png”);
game.load.spritesheet(“rings”, “images/main/rings.png”, 60, 65, 5);
game.load.spritesheet(‘balls’, ‘images/main/balls.png’, 35, 35, 5);
game.load.audio(‘points’, ‘sounds/points.mp3’);
game.load.audio(“lose”, “sounds/gameOver.mp3”);
game.load.spritesheet(“soundButtons”, “images/ui/soundButtons.png”, 32, 32);
},
create: function() {
this.speed = 200;
score = 0;
this.maxSpeed = 400;
this.incSpeed = 25;

game.physics.startSystem(Phaser.Physics.ARCADE);

this.blockGroup = game.add.group();

this.red = game.add.sprite(0, 0, “red”);
this.blue = game.add.sprite(100, 0, “blue”);

```
    this.green = game.add.sprite(0, 100, “green”);
    this.yellow = game.add.sprite(100, 100, “yellow”);

    this.red.inputEnabled = true;
    this.blue.inputEnabled = true;
    this.green.inputEnabled = true;
    this.yellow.inputEnabled = true;

    this.red.name = “red”;
    this.blue.name = “blue”;
    this.green.name = “green”;
    this.yellow.name = “yellow”;



    this.blockGroup.add(this.red);
    this.blockGroup.add(this.blue);
    this.blockGroup.add(this.green);
    this.blockGroup.add(this.yellow);

    //take the center of the game and subtract 50% of the group’s width
    this.blockGroup.x = game.world.centerX - this.blockGroup.width / 2;

    //set the group at the bottom of the game, subtracting the group height plus a little to
create a margin
    this.blockGroup.y = game.height - this.blockGroup.height - 50;

    this.ring = game.add.sprite(game.world.centerX, this.blockGroup.y - 100, “rings”);
    this.ring.anchor.set(0.5, 0.5);

    this.ball = game.add.sprite(0, 0, “balls”);
    game.physics.arcade.enable(this.ball);

    this.scoreText = game.add.text(game.world.centerX, 150, “0”);
```

```
this.scoreText.fill = “#ffffff”;
this.scoreText.fontSize = 64;
this.scoreText.anchor.set(0.5, 0.5);

this.scoreLabel = game.add.text(game.world.centerX, 100, “score”);
this.scoreLabel.fill = “#ffffff”;
this.scoreLabel.fontSize = 32;
this.scoreLabel.anchor.set(0.5, 0.5);

this.lose = game.add.audio(‘lose’);
this.points = game.add.audio(“points”);

this.soundButton = game.add.sprite(10, 10, “soundButtons”);
this.soundButton.inputEnabled = true;



this.setListeners();
this.resetBall();

if (soundOn == false) {
this.soundButton.frame = 1;
}
},
setListeners: function() {
this.red.events.onInputDown.add(this.changeColor, this);
this.blue.events.onInputDown.add(this.changeColor, this);
this.green.events.onInputDown.add(this.changeColor, this);
this.yellow.events.onInputDown.add(this.changeColor, this);
game.input.onUp.add(this.resetRing, this);
this.soundButton.events.onInputDown.add(this.toggleSound, this);

if (!game.device.desktop) {
```

```
game.scale.enterIncorrectOrientation.add(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.add(this.handleCorrect, this);
}

},
changeColor: function(target) {
switch (target.name) {
case “red”:
this.drawRing(3);
break;

case “blue”:
this.drawRing(1);
break;

case “green”:
this.drawRing(2);
break;

case “yellow”:
this.drawRing(4);
break;
}
},
drawRing: function(color) {
this.ring.frame = color;
},
resetRing: function() {
this.ring.frame = 0;
},
resetBall: function() {
var colorIndex = game.rnd.integerInRange(0, 5);
```

```
var yy = -game.rnd.integerInRange(0, 100);
var xx = game.rnd.integerInRange(-100, game.width + 100);

this.ball.frame = colorIndex;
this.ball.x = xx;
this.ball.y = yy;

var rot = game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
this.ball.rotation = rot;
},
toggleSound: function() {
soundOn = !soundOn;
if (soundOn == true) {
this.soundButton.frame = 0;
} else {
this.soundButton.frame = 1;
}
},
handleIncorrect: function() {

if (this.ball) {
if (this.ball.body) {
this.ball.body.velocity.setTo(0, 0);
}
}
},
handleCorrect: function() {

if (this.ball) {
game.physics.arcade.moveToXY(this.ball, this.ring.x, this.ring.y, this.speed);
}
},
```

```
removeListeners: function() {
game.scale.enterIncorrectOrientation.remove(this.handleIncorrect, this);
game.scale.leaveIncorrectOrientation.remove(this.handleCorrect, this);
},
update: function() {
var diffX = Math.abs(this.ring.x - this.ball.x);
var diffY = Math.abs(this.ring.y - this.ball.y);

if (diffX < 10 && diffY < 10) {
this.ball.body.velocity.setTo(0, 0);

if (this.ring.frame == this.ball.frame) {
score++;
this.scoreText.text = score;
this.speed += this.incSpeed;
if (this.speed > this.maxSpeed) {
this.speed = this.maxSpeed;
}
if (soundOn == true) {
this.points.play();
}

this.resetBall();
} else {
if (soundOn == true) {
this.lose.play();
}
this.removeListeners();
game.state.start(“StateOver”);
}
}
```

```
}

}
```

**game_over_state.js**

```
var StateOver = {

preload: function () {

},
create: function () {
this.buttonPlayAgain = game.add.button(game.world.centerX, game.height-200, ‘buttons’, this.replay, this, 1, 0, 1);
this.buttonPlayAgain.anchor.set(0.5,0.5);
}
, update: function () {
},
replay:function()
{
game.state.start(“StateMain”);
}
}
```

Testing-Refresh the game. Turn the device from portrait to landscape,by pressing the rotate button, before and while playing

Expected result-The image telling the player to turn the device to the correct position will appear. The game will pause and resume.

# CHAPTER NINE

## *conclusion*

### Conclusion

Now we have a fully functional game!

I hope you enjoyed it, and had fun learning the code. If you've any questions further, please contact me through my website http://www.williamclarkson.net