Space Shooter Tutorial

# Project Set-Up

Download the asset folder from Blackboard. You will notice that the folder includes 2 subfolders (content with all the resources and js with the phaser.js file).

## The html file

Create an index.html file and include the following code in it:

<!DOCTYPE html>

<html>

<head>

<title>Space Shooter</title>

</head>

<body>

<script src="js/phaser.js"></script>

<script src="js/game.js"></script>

</body>

</html>

## The JavaScript files

Next, create a JavaScript file and name it game.js. Make sure to save it in the js folder.

We will start by initializing our Phaser game inside game.js. Include the code below for the config object and for the game.

var config = {

type: Phaser.AUTO,

width: 512,

height: 512,

backgroundColor: "black",

physics: {

default: "arcade",

arcade: {

gravity: {x: 0, y: 0 }

}

},

scene: [

SceneMainMenu,

SceneMain

],

pixelArt: true,

roundPixels: true

};

var game = new Phaser.Game(config);

We are now going to work on the different scenes for the game. We will create two different JavaScript files, one for each scene we intend on using. Make sure to link the scripts in your index.html file.

Let’s begin with the SceneMainMenu.js. We are going to declare a class named SceneMainMenu and extend the Scene class of Phaser. Basically our SceneMainMenu will be a Phaser Scene with additional functionality. The constructor function is called immediately when instantiating the class. Within the constructor, we are going to set a key SceneMainMenu for the scene. We do that by calling the constructor of the Phaser Scene class (with super).

Add the code below to the SceneMainMenu.js file.

class SceneMainMenu extends Phaser.Scene {

constructor() {

super({ key: "SceneMainMenu" });

}

preload() {

}

create() {

}

}

We are going to add one more JavaScript file to our project. Make sure to include the necessary script tags in the html file.

In the SceneMain.js file, add the code below:

class SceneMain extends Phaser.Scene {

constructor() {

super({ key: "SceneMain" });

}

preload() {

}

create() {

}

}

The project is set up. It’s time to add some functionality to the scenes.

# Menu Scene

## Loading assets

Let’s start by preloading some assets in the SceneMainMenu, in the preload function. We will load two images for the Start button and a sound effect for the click:

this.load.image("sprBtnPlay", "content/sprBtnPlay.png");

this.load.image("sprBtnPlayHover", "content/sprBtnPlayHover.png");

this.load.audio("sndBtn", "content/sndBtn.wav");

## Adding a Title and Button

The create function within our SceneMainMenu class will be called as soon as the scene is started. We will add a button. We first add the sound effect. Then, we add a text for the title of the game. Then we add the play button. There are several events handled for that button. If the mouse is over the button, we change the image used for the button. If the mouse leaves the button’s area, the button is set back to the initial image. If the player clicks on the Start button, the sound effect is played and the SceneMain starts. When the player releases the button, it goes back to the initial image (we don’t really get to see that). Enter the code below in the create function of the SceneMainMenu class:

this.sfx = {

btn: this.sound.add("sndBtn")

};

this.textTitle = this.add.text(

this.game.config.width \* 0.5,

64,

"SPACE SHOOTER",

{

fontFamily: "Arcadepix",

fontSize: 32,

align: "center"

}

);

this.textTitle.setOrigin(0.5);

this.btnPlay = this.add.sprite(

this.game.config.width \* 0.5,

this.game.config.height \* 0.5,

"sprBtnPlay"

);

this.btnPlay.setInteractive();

this.btnPlay.on("pointerover", function() {

this.btnPlay.setTexture("sprBtnPlayHover");

}, this);

this.btnPlay.on("pointerout", function() {

this.setTexture("sprBtnPlay");

});

this.btnPlay.on("pointerdown", function() {

this.sfx.btn.play();

this.scene.start("SceneMain");

}, this);

this.btnPlay.on("pointerup", function() {

this.setTexture("sprBtnPlay");

});

That’s all the SceneMainMenu is going to need for the game. It is now time to add some action to the game in the SceneMain.

# Main Scene

## Loading Assets

For the main scene, we are going to start by loading the necessary assets. In the preload of the SceneMain class, add the following code:

this.load.image("background", "content/starfield.png");

this.load.image("player", "content/player.png");

this.load.image("bullet", "content/bullet.png");

this.load.image("enemy", "content/enemy-green.png");

this.load.spritesheet("explosion", "content/sprExplosion.png", { frameWidth: 128, frameHeight: 128});

this.load.audio("explosionSound", "content/sndExplode.wav");

this.load.audio("laserPlayer", "content/sndLaserPlayer.wav");

## Scrolling Background

Let’s start by working on the scrolling background. To do that, we will instantiate two background sprites, one on the canvas and the other to the right. They will both move to the left. When one of the background sprite is out of sight, we move it to the right side of the screen.

In order to do that, we will need to add code to the create and the update functions. Let’s begin with the create function. Here, we will instantiate two background sprites and store their references in an array named background:

this.background = [];

for(var i=0; i<2; i++) {

var x = config.width\*(0.5+i);

var y = config.height/2;

var bckg = this.add.sprite(x, y ,"background");

this.background.push(bckg);

}

In the update, we move the background sprite by decrementing their x coordinate and then checking if they have left the screen:

for(var i=0; i<2; i++) {

this.background[i].x -= 1;

if(this.background[i].x < -config.width/2) {

this.background[i].x = config.width\*1.5;

}

}

If you run the game now, click on the Play button when it becomes available and you should see the scrolling background.

## The Player

We are now ready to add the player to the main scene. We are going to set it up on the left side of the screen. We are going to allow the WASD keys to move it and make sure it cannot leave the game area.

Let’s begin by initializing the keys we want to use in the create function:

this.keyW = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.W);

this.keyA = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.A);

this.keyS = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.S);

this.keyD = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.D);

this.keySpace = this.input.keyboard.addKey(Phaser.Input.Keyboard.KeyCodes.SPACE);

Now, we are going to add the player’s sprite and make sure it remains in the game area by setting a collision with the world boundaries.

this.player = this.physics.add.sprite(40, config.height/2, "player");

this.player.setCollideWorldBounds(true);

In the update, we are now going to handle the different keys and move the player as needed. Note that for every frame, we will reset the velocity to 0.

this.player.setVelocity(0,0);

if(this.keyW.isDown) {

this.player.setVelocityY(-160);

}

if(this.keyA.isDown)

{

this.player.setVelocityX(-160);

}

if(this.keyS.isDown) //down

{

this.player.setVelocityY(160);

}

if(this.keyD.isDown)//right

{

this.player.setVelocityX(160);

}

if(this.keySpace.isDown)

{

console.log("Shooting");

}

Try to run the game now. You should be able to move the player around the screen.

## Shooting

Let’s set up its shooting ability. We are going to give the player a couple of properties. The first one will be the fireRate to control how fast the player can shoot. The second one will be named nextFire and will track the next time the player can fire. We will set the fireRate to 200 and the nextFire to 0 (the player can shoot as soon as the game begins).

In the create function, below the two lines of code that initialize the player and set it to collide with the World bounds, add the two lines below:

this.player.fireRate = 200;

this.player.nextFire = 0;

We are now going to create some bullets using a group. We will create 30 bullets and use the bullet image. We are also going to iterate through all the bullets and change their active property to false. Add the code below to the create function:

this.bullets = this.physics.add.group({

key: "bullet",

repeat: 29,

});

this.bullets.children.iterate(function(child) {

child.active = false;

});

Now all we have left to do is allow the player to shoot. We are going to check the time and see if we have passed the nextFire time. We are also going to check if we have some inactive bullets. There is a method that counts the active or inactive objects: countActive(true) will count active objects in a group and countActive(false) will count inactive objects in a group.

If the player is allowed to shoot and we have some inactive bullets, then we can get the first inactive bullet in the group using the getFirstDead() method. We make sure we set the bullet to be active, set its position in front of the player, and set its x velocity to 160. We also need to make sure that we get the new nextFire time by adding the fireRate to the current time.

In the update function, where we checked for the space key being pressed, add the following code:

if(this.keySpace.isDown)

{

if(this.time.now > this.player.nextFire && this.bullets.countActive(false) > 0) {

this.player.nextFire = this.time.now + this.player.fireRate;

var bullet = this.bullets.getFirstDead();

bullet.active = true;

bullet.setPosition(this.player.x+bullet.width, this.player.y);

bullet.setVelocityX(160);

}

}

We are almost done with this portion of the project. All we need to do now is to make sure that the bullets are set inactive when they leave the game area. In the update, we are going to iterate through all the bullets and if their x coordinate is greater than the width of the game, then we change them to inactive.

In the update, add the following code:

this.bullets.children.iterate(function(child) {

if(child.active) {

if(child.x > config.width) {

child.active = false;

}

}

});

Try the game now. The player should be able to move around the screen and shoot. Feel free to adjust some of the values (the position of the bullets, or the fire rate) as you see fit.

## Enemies

We are now going to add some enemies to the game. We will create a group of enemies and use a time event to add an enemy every second. We will give them a random velocity. The image is a bit large, so we will need to modify the scale. Note that we set the loop property to true since we want to keep instantiating enemies.

In the create function, add the following code:

this.enemies = this.physics.add.group();

this.time.addEvent( {

delay: 1000,

callback: function() {

var y = Phaser.Math.Between(50, config.height - 50);

var enemy = this.enemies.create(config.width, y, "enemy");

enemy.setScale(0.5);

enemy.setVelocityX(Phaser.Math.Between(-200,-100));

},

callbackScope: this,

loop: true

});

Now, we need to make sure to remove the enemies once they have left the screen. We cannot use the iterate method to go through all the children because we are going to destroy the objects and that cannot be done with iterate (the children array will be changed). In the update, add the following code:

this.enemies.children.each(function(enemy) {

if(enemy.x < -enemy.width)

{

this.enemies.remove(enemy);

enemy.destroy();

}

}, this);

Try to game and notice that the enemies are removed completely once they leave the screen.

## Handling Collisions

We need to handle collisions between the bullets and the enemies and the collisions between the enemies and the player. We don’t really want them to collide as a ball would collide against a wall. We just want to know when they are overlapping so we can handle the event.

Add the following code in the create function, after the player, bullets, and enemies have been created.

this.physics.add.overlap(this.bullets, this.enemies, this.destroyEnemy, null, this);

this.physics.add.overlap(this.player, this.enemies, this.gameOver, null, this);

We also need to define the destroyEnemy and the gameOver functions as they will be called when the collision events occur. When a bullet collides with an enemy, we will destroy the enemy. When the player collides with an enemy, the game is over and we go back to the main menu.

Let’s first create the sound effects and the explosion effect in the create function by adding the code below:

this.sfx = {

explosionSound: this.sound.add("explosionSound"),

laserSound: this.sound.add("laserPlayer"),

};

this.anims.create({

key: "explosion",

frames: this.anims.generateFrameNumbers("explosion"),

frameRate: 16,

repeat: 0,

});

Now we are ready to handle the collisions. After the definition of the update, we are going to add two other functions, destroyEnemy and gameOver. The destroyEnemy function includes two parameters, for the two objects involved in the collision. Since the overlap is defined between the bullets and the enemies (in that order), then the parameters will be bullet and enemy in that order. Automatically, the bullet object involved in the collision will be assigned to the bullet variable and the same goes on for the enemy. We will create an explosion, play the animation, and when the animation is completed, we will destroy the explosion. We will also play a sound effect and then destroy the enemy and disable the bullet, reset its position to the origin and stop it.

destroyEnemy(bullet, enemy) {

var explosion = this.add.sprite(enemy.x, enemy.y, "explosion");

explosion.play("explosion");

explosion.on("animationcomplete", function() {

if(explosion) {

explosion.destroy();

}

});

this.sfx.explosionSound.play();

enemy.destroy();

bullet.active = false;

bullet.setPosition(0,0);

bullet.setVelocityX(0);

}

For the gameOver function, the parameters will be the player and the enemy involved in the collision. We will destroy the enemy, play the explosion sound, and start the main menu scene.

gameOver(player, enemy) {

enemy.destroy();

this.sfx.explosionSound.play();

this.scene.start("SceneMainMenu");

}

We also need to play the sound effect when the player shoots. This happens in the update, when the space key is pressed. See the code below to add the line of code that plays the sound effect:

if(this.keySpace.isDown)

{

if(this.time.now > this.player.nextFire && this.bullets.countActive(false) > 0) {

this.player.nextFire = this.time.now + this.player.fireRate;

var bullet = this.bullets.getFirstDead();

if(bullet) {

bullet.active = true;

bullet.setPosition(this.player.x+bullet.width, this.player.y);

bullet.setVelocityX(160);

**this.sfx.laserSound.play();**

}

}

}

This concludes the tutorial. Submit the zipped project folder (with the js and content folders and html file) in Blackboard.