Aston University
Birmingham

Tutorial 5

Advanced Hit Detection

In previous tutorials we have used simple hit detection, in this tutorial we shall be looking at deciding what to do when two objects collide. In this case we shall be creating bullets in our space invaders game.

Step 1

Lets get some bullets in this! Using the *ReusableObject* we can create bullets that can move around the game and do something when they collide with something else.
The first thing you will want to do is create a variable to store your *ReusableObject*, then assign the *ReusableObject* to that variable in the *preload* function.

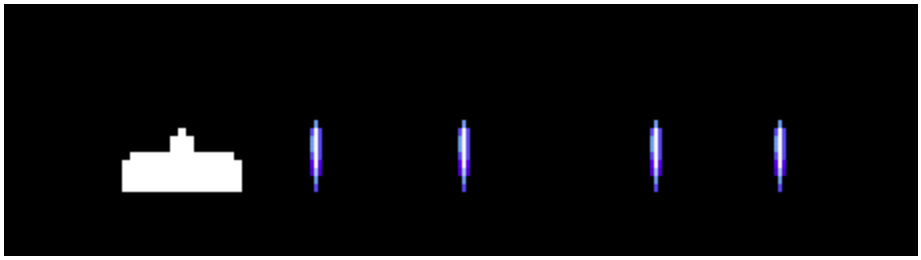
Remember you need to pass through the game object, the name of it and the image to represent it. We will create the bullets for our player so we shall give it the name "playerBullets" and we will pass through the image *"img/bullet.png"*

The next thing we need to do is fire the bullet when the up key is pressed, look in the *update* function you will see an 'if statement' that is activated when the up button is pressed.

You will want to create a variable to store this bullet and then use the *create* function on the *playerBullets* variable you created for these bullets. You can see the documentation for the *create* function here (http://eas-fyphost-02.aston.ac.uk/~smalljh/docs/ReusableObject.html#create) but it takes two parameters, the x and y coordinates. As the player is firing the bullet we want to fire from their x coordinate and just above their y coordinate. So the code will be similar to:

```
var currentBullet = playerBullets.create(player.getX(), player.getY()-20);
```

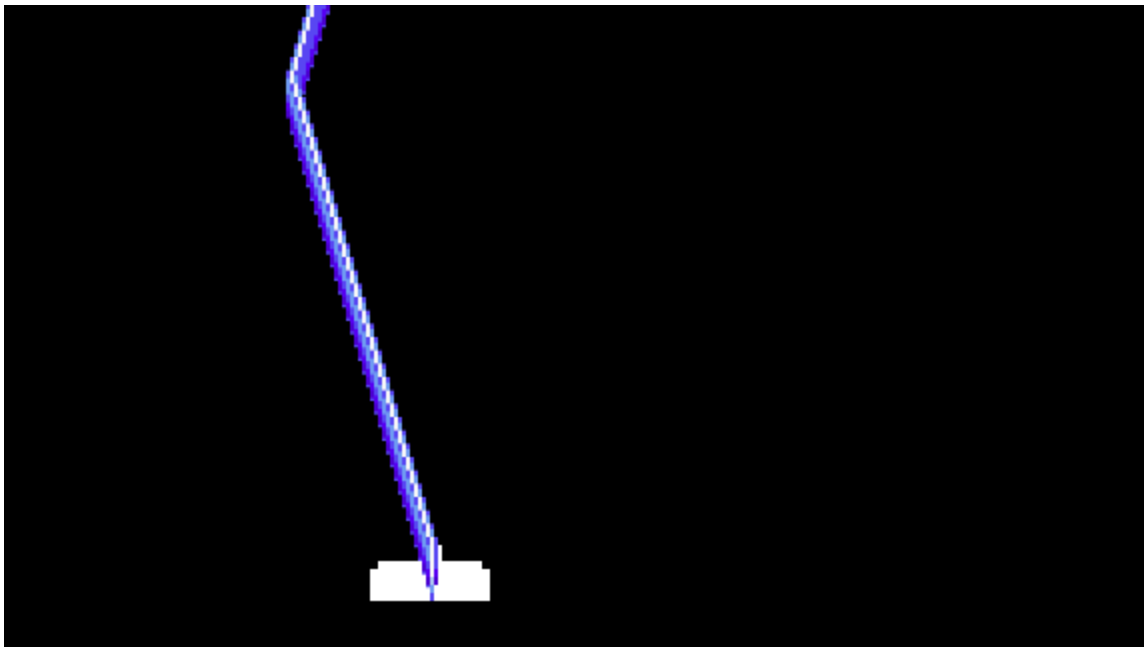
Now if you reload the browser and press up you will create the bullet but see that it does not move.



To get it to move upwards you will want to give it a y velocity of around -400. This can be done by calling the *setVelocityY* function on the previously created variable. E.g.:

```
currentBullet.setVelocityY(-400);
```

Now if you reload the game and press up your bullet will move upwards! Now there is just one problem, if you hold the up key we have a stream of bullets! It looks like a lazer - something we do not want!



To fix this we can set a time limit between each bullet fired, the game object keeps track of the current time. Using this we can see if enough time has passed since the last bullet has been fired. You will want to create a global variable (*lastShotTime)* to check when the next bullet can be fired, you can first set it to 0 as it can be fired straight away.

Note this value is in milliseconds so 1000 equals 1 second!

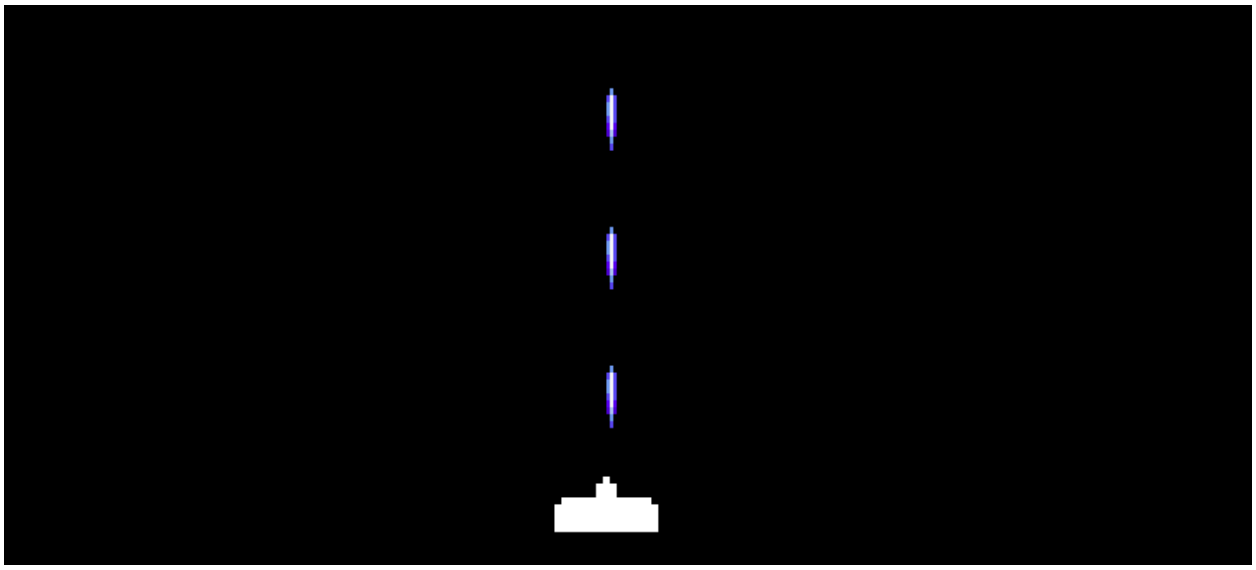
You will want to put a second ‘if statement’ inside the current one that checks if the current game time is greater than the next allowed time to fire the bullet. I.e enough time has passed since the last time the bullet was fired.

Hint: *if(game.getGameTime() > lastShotTime)*

Inside this ‘if statement’ you will want to move (cut and paste) the code you did above that fired the bullet into it. You will also want to increment your next allowed fire time variable by the current game time and then however many milliseconds you want the delay to be. E.g:

lastShotTime = game.getGameTime() + 200;

Now our bullets with have a slight delay between firing!



Step 2

The last thing to do is the hit detection when your bullet hits one of the invaders! This is a very simple thing to do thanks to the API! All that you need to do is use the *checkCollision* function (http://eas-fyphost-02.aston.ac.uk/~smalljh/docs/ReusableObject.html#checkCollision) on the *ReusableObject* you created for your bullets. This function takes 2 parameters:

1. The other object to check a collision against, so in this case its the enemy

2. The function to call when a collision is detected

So a new function is needed, this means that we must create one! You can call the function anything you want but it must take two parameters, the first one will be the object that collides with something, so in this case it is the bullet. The second object is what the first object hit, so in this case it is the invader.

Create this function at the bottom of the file and ensure it has the two parameters as defined above. You should have something similar to this:

```
function hitEnemy(bullet, enemy) {

}
```

Ensure you are invoking the *checkCollision* function *on the ReusableObject* you created for the bullets in the *update* function. We place it here as we want to check for collisions on every update:

```
playerBullets.checkCollision(enemy, hitEnemy);
```

If you run the game again now and a bullet hits an invader you will notice that nothing happens! This is because you have not put anything in the function you created, the simplest thing you can do is call the *kill()* function on the two parameters passed through your *hitEnemy* function you created earlier.

```
function hitEnemy(bullet, enemy) {

  bullet.kill();
  enemy.kill();

}
```

This will delete the objects from the game. In the future you could also implement some sort of score counter or maybe even change the invaders image to one showing that it has been damaged.

If you reload the game now the enemy will be destroyed when hit by a bullet!

Aston University
Birmingham

Step 3

There are a few extra bits that you can do:

1. You can animate the enemies (in exactly the same way you did your player!)
2. Make the invaders move left and right and also descend, this can be done with the *moveGroupX* and *moveGroupY* functions on the enemy object
3. You can get the enemy to fire bullets and detect if they hit the player
4. Show an explosion animation when something is hit with the Animation Object! (http://eas-fyphost-02.aston.ac.uk/~smalljh/docs/Animation.html)