Mobile Game Development for BlackBerry PlayBook with Adobe AIR

Lab #5: Game Engine

Introduction

The objective of this lab is to get our spaceship game engine up and running. We'll learn about the basics of game logic, while building a game of our own.

Where we left off

Continuing where Lab 4 left off, you should have your game set up, with an accelerometer controlled ship on stage. I let you control how the ship will be affected by the accelerometer, by letting you write the code. This is because The game is built by you, not me. So it should be a reflection of how you think the game should react when things happen. The ship is controlled, and moved, the why you think it should be – and this is going to be a directly related to peoples opinions of your game. The ship is going to be the first thing the player interacts with, and if they don't like the way it reacts, it'll ruin their entire experience.

With that said, you may want to take some time to go back, modify your code, and get the ship controller the way you want it.

Accelerometer vs. Button Control

This is a highly debated topic in mobile development. I won't get too much into it, but this is crucial when it comes to designing games. I chose accelerometer, because it is more intuitive to pick up and use, there is less clutter on the screen, makes it more challenging to control, and finally – makes the user feel like they are actually steering the ship rather than playing a game. (Ever see your parents play a video game? They don't just push the buttons; they turn the controller, as if they were driving a car.) But like I said before, if you don't like my decision, feel free to change it to button controls. This is *your* game.

Asteroids

Now that you have a controllable ship, you'll need to set up some enemies. Start by creating a new class called *AsteriodClass.as*. Copy the skeleton from *Asteroid.txt*. Read the comments for explanations, and fill in the required code where it is needed.

Spawning Asteroids

In your *init* function (gameboard.as), you'll want to add an event listener (on frame). Event listeners are set up in ActionScript like so:
*object.***AddEventListener(***EventType.Event, functionName);*

So, if we want a function named *newframe*, to fire every frame change, we'll add our event listener like so:

```
this.addEventListener(Event.ENTER_FRAME, newframe);
```

Now that our initializer function adds an event listener, we'll need to create the function it fires, but first we'll need some class variables. Create a *private* int called *onscreen*, and initialize it to 0. As well as a *public static* var as a new array, called *rocks*.

Create a private function called *newframe*. **Ensure it accepts an event (event:Event), as an input parameter, or it will not run**. Inside this function, put the following:

```
if (onscreen < 6)
                        {
                                var temp:AsteriodClass = new AsteriodClass(stageR, 
ourShip, this);
                                temp.addEventListener(Event.REMOVED_FROM_STAGE, 
removeEnemy, false, 0, true);
                                rocks.push(temp);
                                this.addChild(temp);
                                onscreen++;
                        }
```

This will check if there are 5 asteroids on screen every frame. If there are less, then it will add one. Notice we added an event listener to the asteroid. If the asteroid is ever removed from the display list (by calling *asteroid.removeself();*), then this event will fire. Build a private function called *RemoveEnemy,* don't forget to include an event as an input parameter, that will be called when this event fires. Add the following code to it:

```
rocks.splice(rocks.indexOf(event.currentTarget), 1);
                                onscreen--;
```

This code will remove the reference to the asteroid from our Array, and then decrease our onscreen counter so another asteroid can be added to the stage next frame.

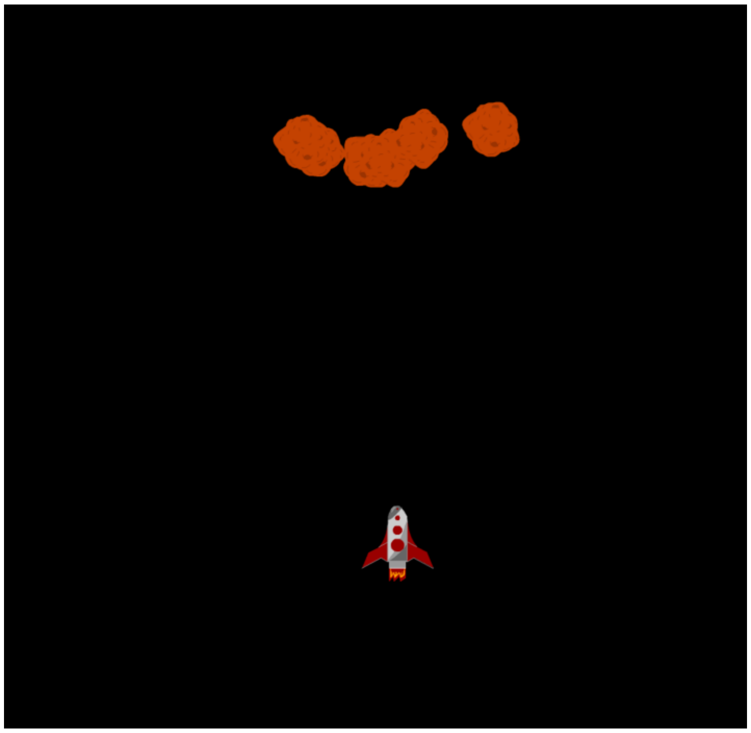

Figure 1

Run your game, and you should see something similar to Figure 1. When an asteroid hits you, it should disappear, **DAMAGED** should appear onscreen, the asteroid should disappear, and another spawn at the top. When an asteroid runs off screen, another should spawn at the top.

Lasers
Now you have the structure of the two main classes in your game, the Ship, and the Asteroid. You also have basic interaction (hittest) between the asteroid and the ship, but we'll need to add an extended interaction between the ship and the asteroid – a laser.

Create a new class called *Laser.as*. Copy the code from *laser.txt*, and fill in the *removeSelf()* code.

FIRE THE LASER!!!!111!!!1!
Now that we have a laser, we need to implement a way to actually fire it. Throughout this lab, you may have wondered why we had a blank space at the bottom off the screen. This empty space is going to be for our attack button.

In *gameboard.as*, declare a new private class variable called button, and give it the type Sprite.

Add the following code to your *init* function.

```
//Create a new instance of a Sprite to act as the button graphic.
                    button = new Sprite();
                    button.x = 0;
                    button.y = 875;
                    //Set the color of the button graphic
                    button.graphics.beginFill(0xFFFFFF);
                    //Set the X,Y, Width, and Height of the button graphic
                    button.graphics.drawRect(0, 0, stageR.stageWidth, 250);
                    //Apply the fill
                    button.graphics.endFill();
                    button.addEventListener(MouseEvent.MOUSE_DOWN, 
ourShip.fireBullet);
                    this.addChild(button);
```

Once again, notice the event listener. This time, it calls a function inside the *ship* class. So open up *Ship.as*, and create a new public function called *fireBullet*. Once again, do not forget to include the event as an input parameter. This type the event type is a *MouseEvent*. Put the following code inside the function:

```
stageRef.addChild(new Laser(stageRef, x, y));
```

Running your game should look like Figure 2.

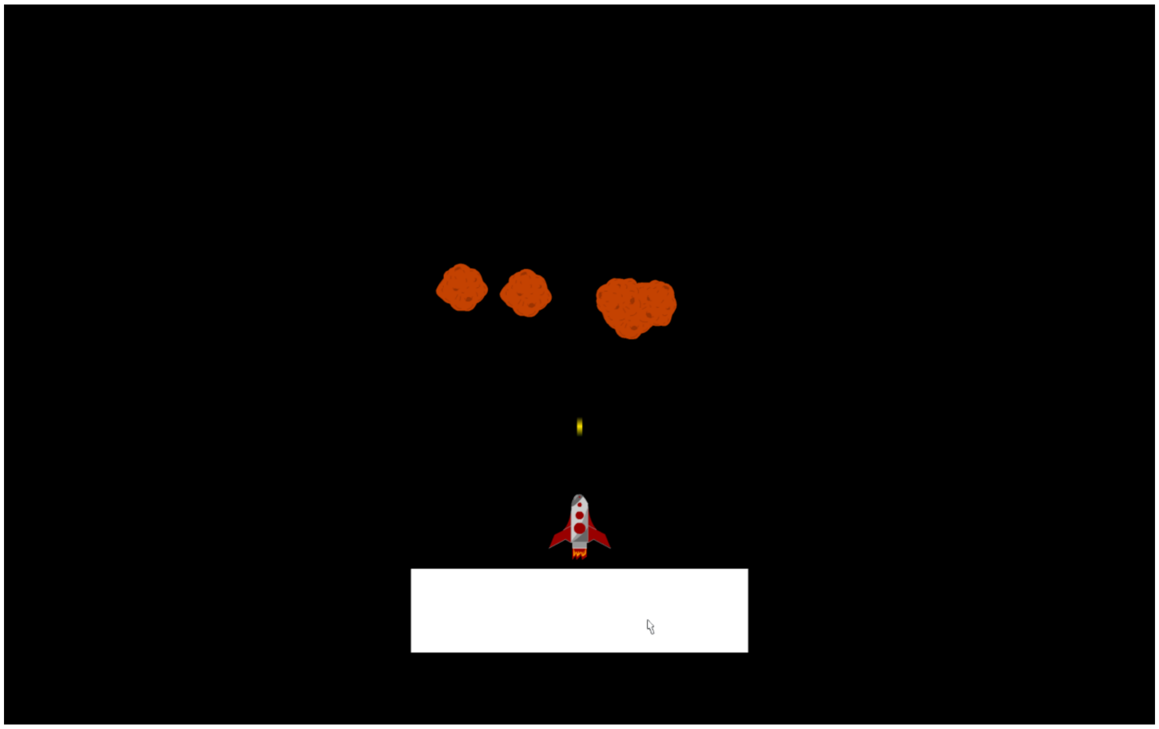
Figure 2

Pushing the game should fire a bullet. When a bullet hits an asteroid, it should disappear and another should respawn.

And there you have it. Your first, very basic, top-down space-shooter game engine.

Polishing your game, testing yourself, and expanding your knowledge
Now that you have the basics of your game engine designed, consider doing the following to polish your game, test yourself, and expand your knowledge.

1. Change the fire button graphic, it something both intuitive, and sleek in appearance
2. Research methods to make your hitboxes more accurate. Research multiple hitboxes, radial hitboxes, and pixel masks.
3. Add sounds. (This should be very easy, import, instantiate, object.play)
4. Add scores/health that will cause "Winner" or "Gameover".
5. Add explosions. (Change the display list when an object is hit, remove explosion after a short timer has expired)