Mobile Game Development for BlackBerry PlayBook with Adobe AIR

Lab 4: Setting Up the Environment

Introduction

The objective of the next set of labs is to create a basic top-down spaceship shooter. This will be a simple game, where a spaceship will be on the bottom of the screen. It will be able to move horizontally, and will take accelerometer readings as a control input. Asteroids will spawn at the top of the screen, and move towards the spaceship. The player will then have to either destroy the asteroids, or dodge them.

Software Requirements

-Flash Builder 4.6 (http://www.adobe.com/products/flash-builder.html)
-Adobe Flex SDK (Included with Flash Builder)
-Playbook SDK for Adobe Air 2.0
(https://bdsc.webapps.blackberry.com/air/download/sdk/)
    -Playbook SDK must be integrated into Flash Builder.
-Playbook Simulation Environment

Skill Requirements

-This lab assumes basic knowledge of AS3
-This lab assumes basic understanding of the Flash Builder IDE
-This lab assumes you have completed lab sections *gravity ball*, and *BMI Calculator*.

Set-up

Included in this labs folder is *Lab_Blasteriods.zip*. Import this into Flash Builder, as a *Flash Builder Project*.

This project contains a .swc with a number of graphically assets (both in MovieClip format, and Button format), that you will make use of during the development of your

game, as well as a few functions in the main AS file that will set up a main menu, about menu, how to play menu, as well as instantiate *gameboard.as* when the play game button has been clicked. The game currently has auto-roatate on (For testing purposes, simulators do not work well with non-default orientations, and we'll be working in portrait mode). It will be turned off later. It is highly recomended you test this game using the VM Image simulator.

Ship Class

Create a new class, entitled *Ship.as*. Paste the skeleton from *Ship.txt* into *Ship.as*. Refer to the comments in the skeleton. You are required to write code that will make the ship move left and right. The accelerometer will return values -1 < AccelerationX < 1.

(Further exploration) If you're feeling creative, since the Ship class extends the MovieClip class, play with the *this.rotate* and *scale* properties to make your ship look like it is turning.

Setting up gameboard, adding our Ship

Once your ship class is done, we'll need to add it to our gameboard in order to test it. Before we do this though, we'll need to set up our gameboard class so it is easy to wortk with. First, notice that our gameboard construct accepts *_stage* (stage temp variable) as a parameter? Similar to our ship class, we want to set up a stage reference. This will make it easy to manage objects we have added to the stage. But we want our local to function stage variable, to be globally (class) usable. So create a private class variable, called stageR (stage reference), and assign our temp stage variable to it in the constructor.

Create another class variable. This one will be public. Call it *ourShip*, give it the type declaration of *Ship*. But do not instaniate it. We will instaniate it later.

Next, create a new private function called *init*. Make it accept 0 parameters, and return void. This will act as our initalizer function. The reason we will not use our constructor function as an initalizer, is *init* can be called much easier than the constructor, any time we want (say if we want to restart the game, because the player died), and this will make reinitalizing everything easier than reinstanitating. Now, inside our *init* function, instantiate *ourShip* as a new *Ship* (don't forget it requires the stage reference as input). Set it's *x* value to the middle of the stage. Set it's *y* value to 800. Then add the ship to the gameboard's display list. It's import you add it to the gameboard's display, and not the stage. This is so if the gameboard is removed, so is the ship. You can do this, by using *this.addChild();*. *this* being the class.

Run your game. You're ship should be visible. And you should be able to steer your ship by tilting your device. (Of course you won't be able to tilt your device in the simulator, the accelerometer can only be tested using a real device. But the simulator will allow us to get a general idea of what's going on, and ensure our display list is being updated properally.)

You running game should look like Figures 1 and 2.

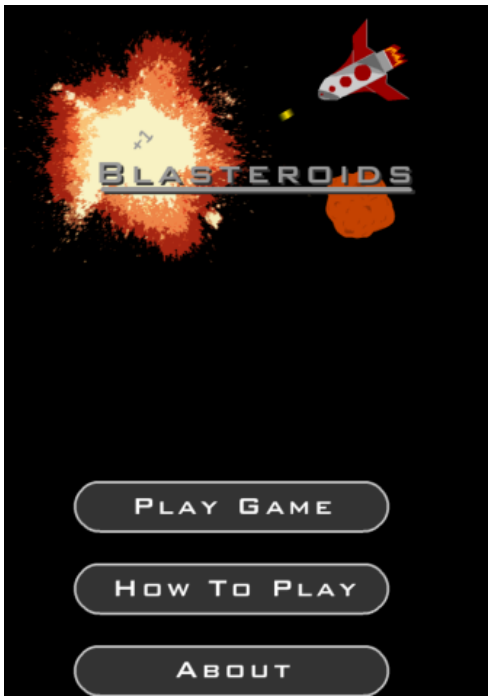

Figure 1

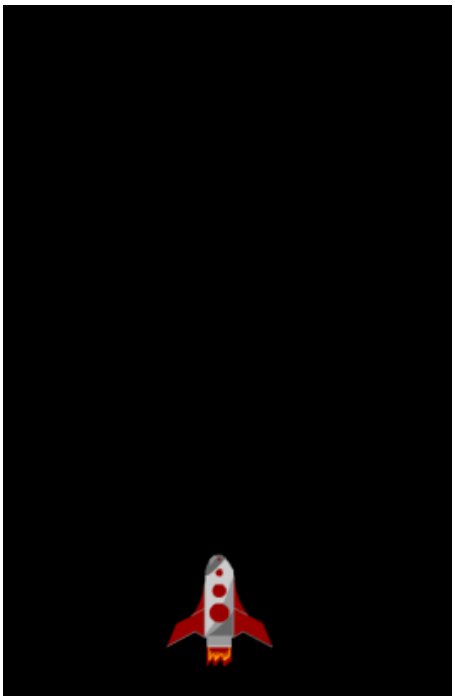
Figure 2

Next Lab
Congrats. You have the basic structure for your game set up. In the next lab, we'll be creating our game engine, while talking about basic game logic and design.