**HTML5 (Project1):**

For my learning object for Project 1, I have decided to write about a simple Image Resource Loader that allows you load multiple images at the one time. The code in itself is very simple.

var imageRepository = new function() {

// Define images

this.star = new Image();

this.background = new Image();

this.spaceship = new Image();

// Ensure all images have loaded before starting the game

var numImages = 3;

var numLoaded = 0;

function imageLoaded() {

numLoaded++;

if (numLoaded === numImages) {

window.init();

}

}

this.background.onload = function() {

imageLoaded();

}

this.spaceship.onload = function() {

imageLoaded();

}

this.star.onload = function(){

imageLoaded();

}

// Set images src

this.background.src = "imgs/bg.png";

this.spaceship.src = "imgs/SpaceShip.png";

this.star.src = "imgs/Star.png";

}

First off we create the objects that will hold the image. For this example we have a ship, a background and a star image respectively. We designate them the type Image().

* + this.star = new Image();
  + this.background = new Image();
  + this.spaceship = new Image();

Next we check to make sure that the images have loaded before the game begins to stop problematic game crashes.

var numImages = 3;

var numLoaded = 0;

function imageLoaded() {

numLoaded++;

if (numLoaded === numImages) {

window.init();

}

}

So what we do here is define a variable with the value of our total images and set a second variable outside the function to keep track of how many are loaded. Within the function, we increase the variable that keeps track of how many images we’ve loaded.

We then check and see if the amount of images loaded equals the amount of our total number of images. When it does we tell the page to initialise.

Next we actually check that our images have loaded using our function:

this.background.onload = function() {

imageLoaded();

}

this.spaceship.onload = function() {

imageLoaded();

}

this.star.onload = function(){

imageLoaded();

}

So we can determine that if all images load successfully that our counter will indeed match the total image number. However if the image does not load successfully, we can tell which one it is by outputting our counter.

Finally we specify the location of our file to be loaded in:

this.background.src = "imgs/bg.png";

this.spaceship.src = "imgs/SpaceShip.png";

this.star.src = "imgs/Star.png";

Now you have three loaded images and you can now use them as a draw-able object to be easily displayed on the screen.

**Android (Project 2):**

In this learning object, I’m going to be using Box2D to control Sprite Movement.

For this example I will be using my current Player class. The code is as follows:

**public** **class** Player **extends** AnimatedSprite {

**private** Body body;

**public** Player(**float** pX, **float** pY, VertexBufferObjectManager vbo, Camera camera, PhysicsWorld physicsWorld)

{

createPhysics(camera, physicsWorld);

}

**private** **void** createPhysics(**final** Camera camera, PhysicsWorld physicsWorld)

{

body = PhysicsFactory.*createBoxBody*(physicsWorld, **this**, BodyType.*DynamicBody*, PhysicsFactory.*createFixtureDef*(0, 0, 0));

body.setUserData("player");

body.setFixedRotation(**true**);

physicsWorld.registerPhysicsConnector(**new** PhysicsConnector(**this**, body, **true**, **false**)

{

@Override

**public** **void** onUpdate(**float** pSecondsElapsed)

{

**super**.onUpdate(pSecondsElapsed);

camera.onUpdate(0.1f);

body.setLinearVelocity(**new** Vector2(5, body.getLinearVelocity().y));

}

});

}

In our constructor we call the function createPhysics(). As you can probably guess, this sets up our Physics World and the parameters for the Body object which will represent the player’s entity (in this case an animated Sprite).

Within our function we invoke the following commands:

body = PhysicsFactory.*createBoxBody*(physicsWorld, **this**, BodyType.*DynamicBody*, PhysicsFactory.*createFixtureDef*(0, 0, 0));

//This creates a new BoxBody, assuming the object fits better into a square. There are other types of bodies too

body.setUserData("player");

Next we tell the body who exactly it’s representing.

body.setFixedRotation(**true**);

Finally we tell the body that it is not allowed to rotate from its current orientation (as in this case, there’s no particular need).

Next comes the update code where we can apply our transitions to the player’s position.

physicsWorld.registerPhysicsConnector(**new** PhysicsConnector(**this**, body, **true**, **false**)

Here we tell our physics world to register a connection between the body and the Sprite in order to move the sprite.

{

@Override

**public** **void** onUpdate(**float** pSecondsElapsed)

{

Next we override the native update method with the above one(The IDE will usually add them in for you.

**super**.onUpdate(pSecondsElapsed);

Next we tell the super(or base) class to update and send it the current time that has elapsed.

body.setLinearVelocity(**new** Vector2(5, body.getLinearVelocity().y));

}

});

}

Next we set our Body’s velocity in the x Direction.

We can use this velocity to apply it to the player’s position.