

# MMGame

## Tutorial by Deniz Hashimov

### MMGame Tutorial (Flappy Bird Clone/Remake) Introduction

Hi , we will be developing a clone/remake of the sensation that was Flappy Bird, a frustratingly addicting mobile game that has managed to conquer the casual gaming world with its impossibly difficult one-touch gameplay.

#### The Details

- We will be using libGDX - a cross platform game development framework.
- libGDX allows us to simultaneously develop for iPhone, iPad, Android, Mac and PC, along with HTML.
- We will be using Java.
- I will provide all the code, art and other assets. You can use them from the links in the tutorial.

#### Plan of Action

1. I will begin with an in-depth analysis of Flappy Bird. Getting the feel/gameplay right is our first priority.
2. I will then discuss how to setup the libGDX framework, and give you a quick tour of this new environment.
3. Next, I will code the game and its logic, adding artwork as necessary.
4. With the gameplay finished, I will add buttons and make ther final touches, such as sound effects.

### Part 1 - Flappy Bird - An In-depth Analysis

To clone a game properly, we must understand its behavior perfectly. In this section, we are going to study various gameplay elements of *Flappy Bird* so that we can emulate the game more accurately.

I am going to try to determine how each gameplay element was implemented. Of course, this is just an approximation -- I may be completely wrong, but it should be close enough for us to mirror the game. If any drastic changes need to be made, I will let you know in a future section.



A quick tracing of the bird revealed its dimensions to be 17 pixels (width) x 12 pixels (height). It makes use of just seven colors. The bird takes up 1/8 of the game width, which seems to be 135 or 136 pixels at quick estimation. The bird is scaled accordingly to fit the device width. The bird also comes in three different color schemes, alternating randomly.

## Bird Physics

It's difficult to experiment with physics in this game without dying, but from my attempts, I have discovered the following:

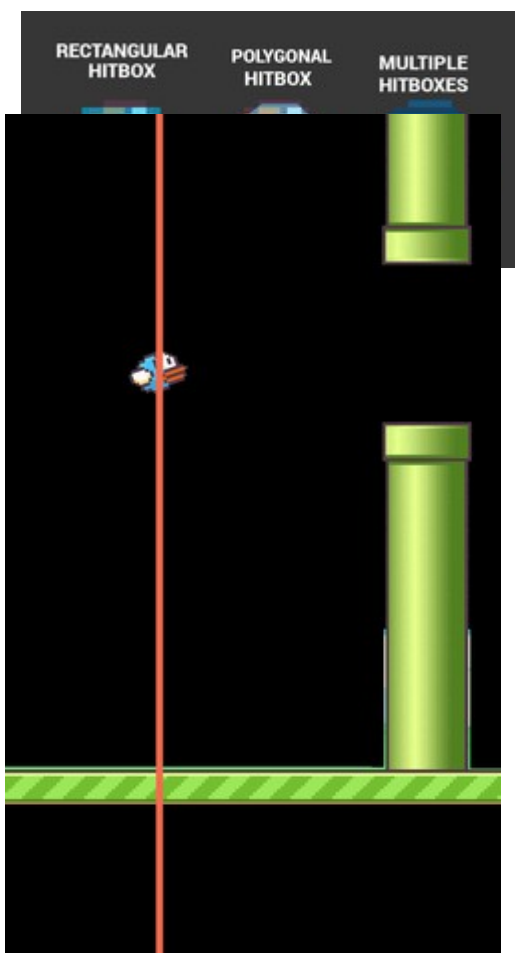
1. The bird accelerates due to gravity; i.e. vertical velocity is always increasing downwards.
2. But there's a cap. You can't go faster than this velocity cap.
3. No matter the current velocity, the bird will gain the same amount of height when the screen is tapped.
4. The rotation of the bird is correlated to its vertical velocity. Animation (flapping) only occurs when moving upwards.

Our primary focus will be to get these things implemented as closely as possible, as the *feel* of the gameplay depends primarily on the physics.

## Collision Detection

When should the bird die? I have no idea how it was implemented in the actual game. From what I can see, however, pixel-perfect collision seems to be the way to go. We are going to create a hit box for our bird, which will be used to check for collisions with the pipes.

We don't want a small hit box, as that would make the game too easy. We don't want to make a large hit box, because people will get angry if they die without hitting anything.



The way I'm going to implement this is to create a single hitbox using a Rectangle object.

## The Pipes

The Pipes may be the most difficult part of the game to get perfectly right, but it is essential that we do so. A large part of the appeal of this game is its difficulty. If the difficulty somehow changes, by our miscalculating the speed or generating the pipes inconsistently, the game will not feel right. It won't have that frustration-reward-addiction system.

You never see more than 6 pipes at once, so we will create 6 pipes. The pipes seem to come at the same time interval every time, so the distance between each set of pipes will be constant. As soon as one set of pipes becomes invisible (moves beyond the left edge of the screen), we will redetermine the height (more on that below) and move the set to appropriate location beyond the right edge of the screen.

The height of the opening varies, but the size of the opening does not. The easiest way to implement this would be to simply move the column vertically to a random Y position when we reset its X position (within parameters). When we implement the pipes, I will examine the pattern in more detail to determine whether they truly follow a random pattern and how much they are able to shift up and down.

## Animations

This is an extremely simple game. I have blacked out the static elements in this game -- the background and the sand. These will never change.

The bird stays fixed horizontally, at approximately 1/3 of the screen width.

The grass(?) and the pipes are the only elements that need to scroll horizontally, and they do so at the same speed. The grass will be extremely easy to implement, so we will not discuss that here.

## Handling Different Screen Sizes



On my device, the bird seems to be centered vertically (see red line in image to the left). Looking at this, I predicted that the game stretches equally at the top and bottom, so that the size (or ratio) of the essential gameplay area remains the same. No device confers an advantage, as you can never see beyond a fixed width. Instead, the two reddish regions at the top and the bottom of the screen seem to stretch if you have a taller device, rather than show you a smaller width.

To confirm this, I had a look at the game's 3.5 inch iPhone version, which I believe the game was originally built for, and the essential gameplay area has a similar size to the non-shaded region on the image to the left.

So, we are going to stick with the following assumptions when building the game, to maintain a level of consistency:

- We will use the 3.6 inch Retina iPhone (640x960) as our baseline.
- We assume that all the essential gameplay happens in the rectangle of that ratio.
- Bird Width is 17 pixels (scaled accordingly).
- Game width is ~135 pixels, scaled accordingly (by factor of 4.75x on iPhone)
- Game height will vary with the device, but the essential height (where all the gameplay actually happens) will be  $(960/640) * 135 = 203$  pixels.

# MMGame Tutorial (Flappy Bird Clone/Remake with extras)

## Part 2 - Setting up libGDX



Welcome to Part 2. In this section, we will be setting up the libGDX framework, which will do a lot of the low-level work for us, so that we can focus on what matters: gameplay.

Before we move any further, have a look at MMGame to the left. MMLogo will be the main character of our game.

As always, the installation/setup is the most boring part of the tutorial. Thankfully, this is very fast and easy! Kudos to the libGDX team for making this easy

## Setup Java, Download ADT

If you do not have Java installed on your machine, and you do not have Eclipse up and running with Android Development Tools, click [here](#) to install those.

## Downloading libGDX / Creating Projects

LibGDX offers cross-platform development that lets you write code once and run on multiple platforms. This is possible because in the libGDX architecture, you have a core Java project in which you write all of your high-level source code (typically using some kind of interface). Whatever code you write in the core Java project will interface with platform-specific code via a variety of platform-specific Java

projects (this is the core functionality offered by libGDX).

To set up the core Java projects and the supporting platform-specific projects:

1. Click [here](#) to Download the Setup App.

2. Once downloaded, you must do one of the following:

3a. If on a Mac, try double clicking the .jar file.

3b. If on a PC, COPY the downloaded file to the Desktop and then open up a Command Prompt. Type the following:

**(DO NOT TYPE THE '>')**

```
> cd desktop
```

```
> java -jar gdx-setup.jar
```

4. Once this has been done, you will see the following screen:



6. Enter the information shown below (and in the image above): you may change the Destination to any folder you wish.

**Name:** MMGame

**Package:** com.mentormate.mmgame

**Game class:** MMGame

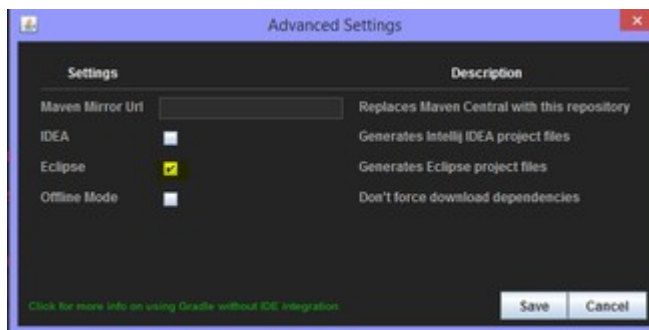
**Destination:** Your choice. Remember which folder you choose.

**Android SDK:** The location of your Android SDK installation. If you are using the ADT Bundle (**Android**

**Developer Tools:** Eclipse + Android SDK, etc.) this is located inside **sdk** inside the **adt-bundle** folder.

Make sure that the *Sub Projects* **Desktop and Android** are checked, and uncheck any *Extensions* (support classes that offer additional functionality for libGDX).

This will automatically setup five Java projects for us in the destination you provided. The **core project** will be where we write our game code. The Android project will access the code in our core project and execute it with platform-specific implementation in order to make the game work in each platform.

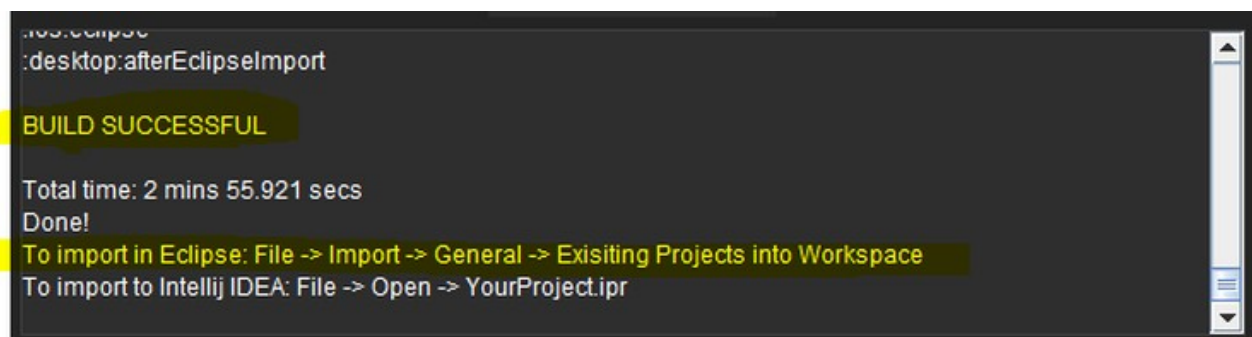


7. Next, we will generate Eclipse projects by pressing Advanced, and selecting "Eclipse"

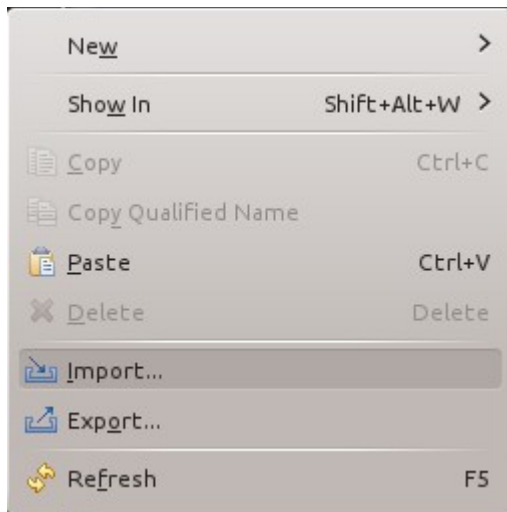
Note: libGDX uses a build tool called Gradle which automates the task of building your application, manages .JAR dependencies (for adding extensions) and makes it easy collaborate with others. Gradle is a large topic of its own, and you need to have some background in using build tools such as Ant or Maven. Rather than spending a week discussing that topic, we will pretend Gradle does not exist in this tutorial.

8. Once the downloads are finished, simply press the "Generate" button.

9. The setup app will take a minute to complete downloading and configuring your project. Once you see the following message, you may close out of the Project Setup.

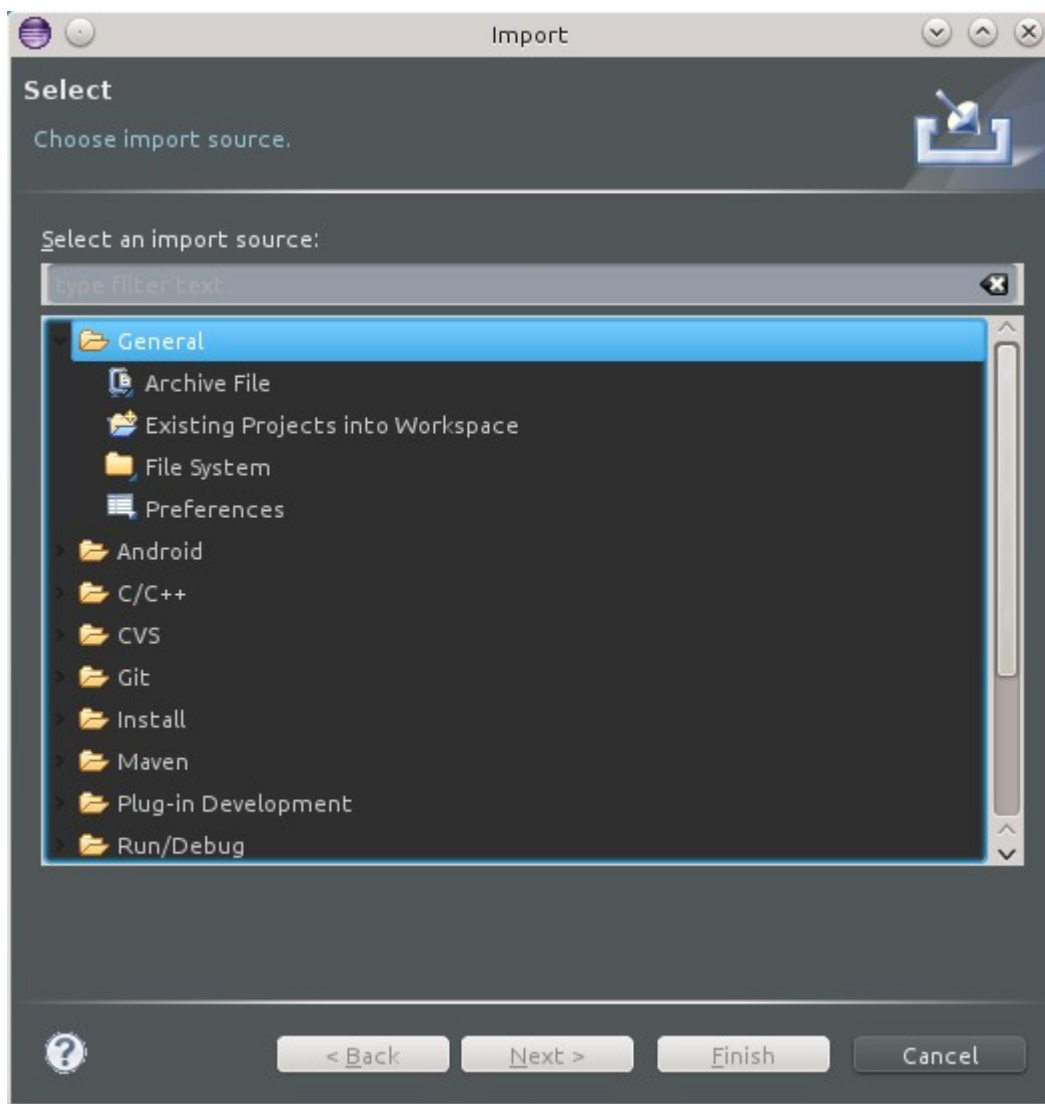


10. Now that we have our three projects generated inside the Destination folder from Step 6, we can import them into Eclipse. Open up **Eclipse**.

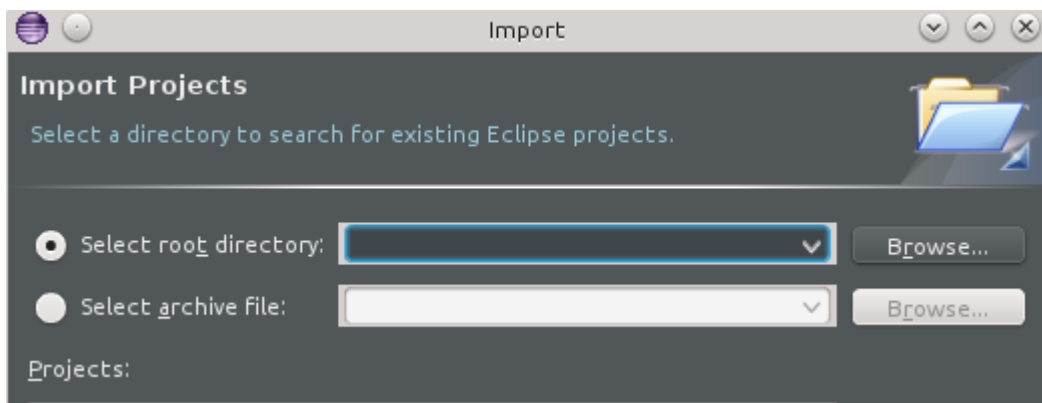


11. Right click in the Package Explorer and press Import, as shown below.

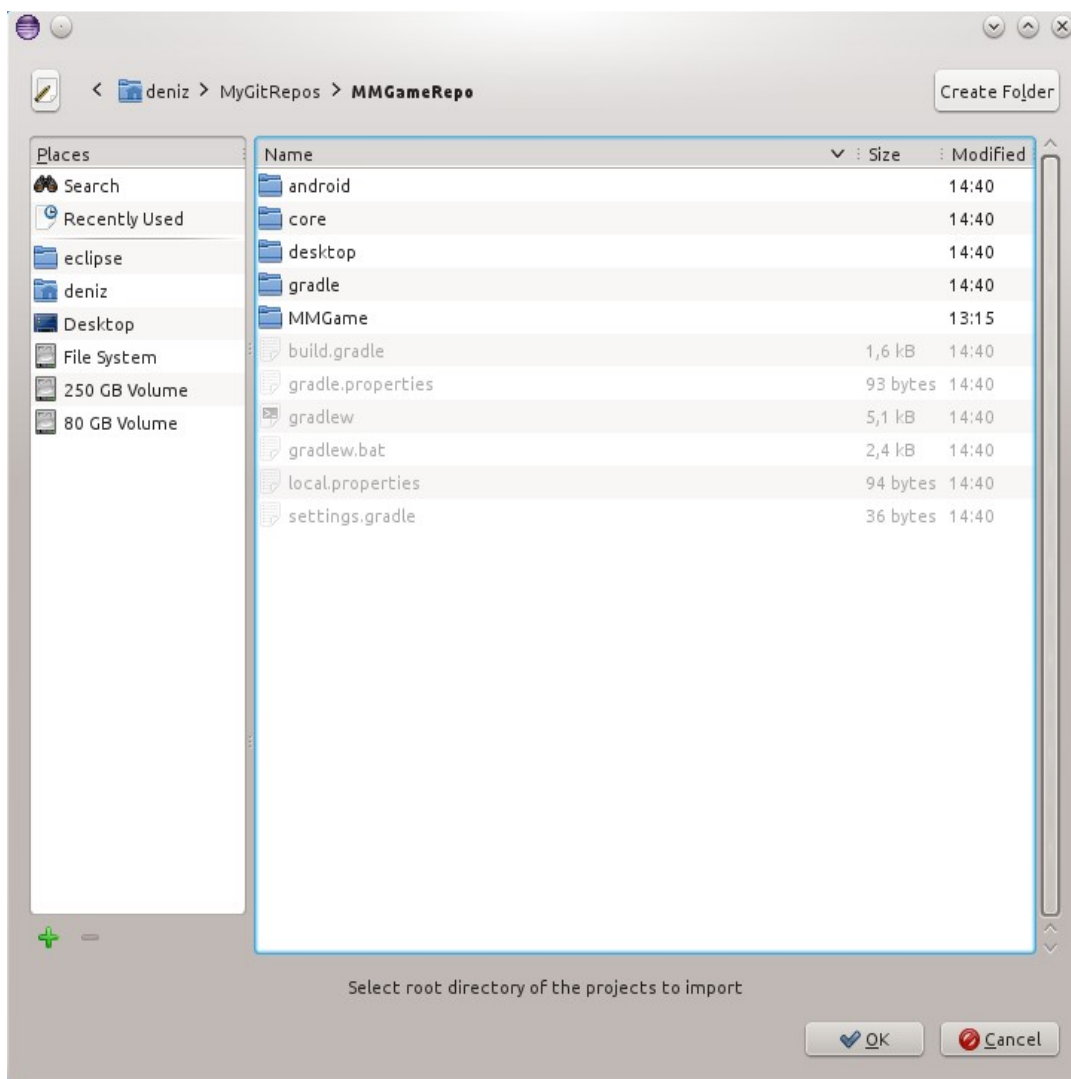
12. Choose General > Existing Projects into Workspace



13. Click "Browse" next to select root directory



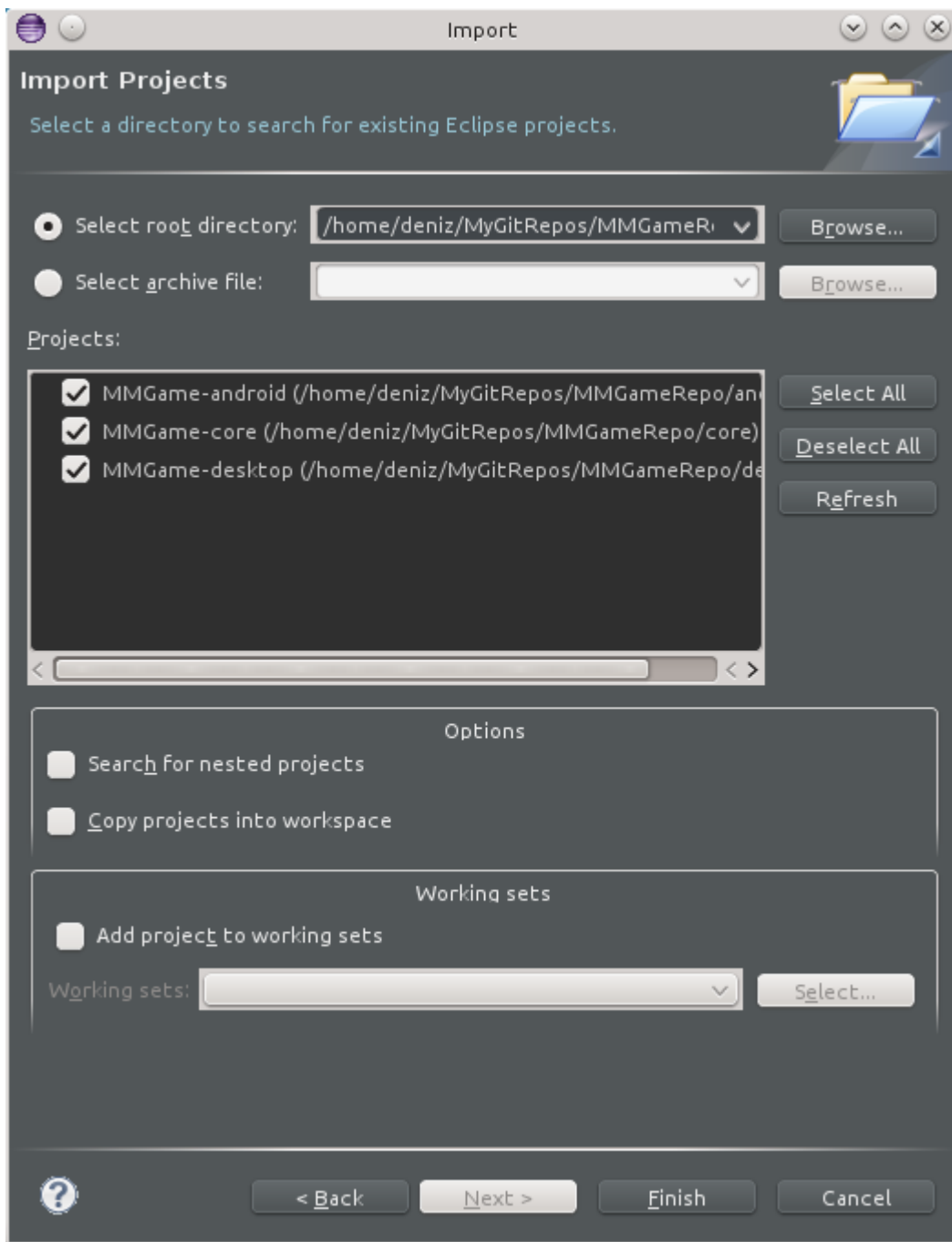
14. Navigate to the Destination folder from Step 6, and click Open.



15. Select all three projects

as shown below, then click "Finish"





16. Now we are done! We have imported our Java projects into Eclipse, and we are ready to start writing code

17. To make sure everything is setup properly, open the **MMGame – desktop** project and navigate to its DesktopLauncher.java class. Update it as shown below:

```

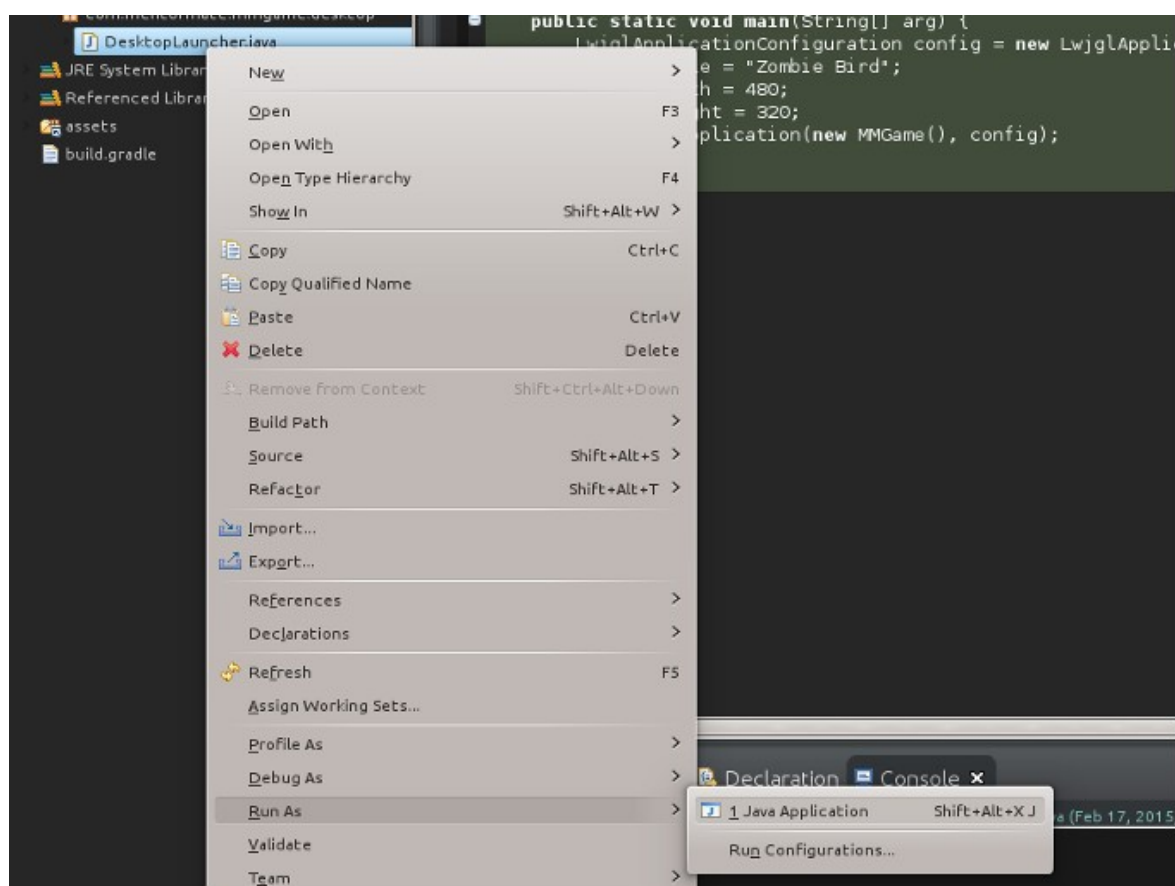
package com.mentormate.mmgame.desktop;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;
import com.mentormate.mmgame.MMGame;

public class DesktopLauncher {
    public static void main(String[] arg) {
        LwjglApplicationConfiguration config = new
        LwjglApplicationConfiguration();
        config.title = "MMGame";
        config.width = 480;
        config.height = 320;
        new LwjglApplication(new MMGame(), config);
    }
}

```

18. Right click and Run the **DesktopLauncher** class (bottom left), and you will see the following (bottom right):





19.If you've gotten this far, libGDX is working properly, and we are ready to move on. Join me in Part 3!

## Part 3 - Understanding the libGDX Framework

In this section we will be building some of the helper Classes and Methods that will help us when we build our game. Before we write any code, here are a few technical details you should know.

libGDX is under an Apache 2.0 license, which allows you to freely modify and distribute the code, provided that you give credit to the original author. All built-in libGDX classes will have an Apache 2.0 header. As we will not be modifying these files, and since the licenses are already included for us, we do not have to include it ourselves or modify the header in any way.

### Basic Structure (How we will design and implement our game)

Let's take the time to discuss how we will implement our game. Here's a quick diagram I threw together to describe the design.



We are going to be working with the ZBGame branch first. We will start implementing the Framework Helpers and the Screen Classes, among which is the GameScreen.

GameScreen relies on two helpers: the World and the Renderer. The World will need to talk to the Gameplay classes in order to make our game objects.

If that all makes sense, let's move on.

**Warning. The following will be the most conceptually challenging section in the whole tutorial.**

## Conventions used in this Tutorial (Read!)

There will be a few times when I reference existing code in the libGDX library, such as the Game class below. These classes are already built-in to your project, and you do NOT need to write any code. Simply refer to the classes as I ask, so that we can build around them. All of these classes will have the Apache 2.0 header in their original form, with the following authors cited:<https://github.com/libgdx/libgdx/blob/master/gdx/AUTHORS>.

I will use a WHITE background for built-in classes, and a dark background for code that you need to write. I will make this clear as we progress.

Have a look at the Game class really quickly. You do NOT need to write any code. Just skim through it.

## Game Class (Built-in)

```
public abstract class Game implements ApplicationListener {
    private Screen screen;

    @Override
    public void dispose () {
        if (screen != null) screen.hide();
    }

    @Override
    public void pause () {
        if (screen != null) screen.pause();
    }

    @Override
    public void resume () {
        if (screen != null) screen.resume();
    }

    @Override
    public void render () {
        if (screen != null) screen.render(Gdx.graphics.getDeltaTime());
    }

    @Override
    public void resize (int width, int height) {
        if (screen != null) screen.resize(width, height);
    }

    /** Sets the current screen. {@link Screen#hide()} is called on any old screen, and {@link
    Screen#show()} is called on the new
    * screen, if any.
```

```

* @param screen may be {@code null}
*/
public void setScreen (Screen screen) {
    if (this.screen != null) this.screen.hide();
    this.screen = screen;
    if (this.screen != null) {
        this.screen.show();
        this.screen.resize(Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
    }
}

/** @return the currently active {@link Screen}. */
public Screen getScreen () {
    return screen;
}
}

```

## Examining the Game class (Difficult - skip if necessary)

The above Game class is an implementation of the ApplicationListener, which will be the interface between our code and the platform-dependent application that is directly launched by our target device.

For example, when Android launches our app, it will look for an ApplicationListener. We can easily provide one by providing a Game object, as declared above.

There is one minor issue. Notice that Game is an abstract class. This means that it is choosing to NOT implement some of the methods that are listed in our ApplicationListener interface. Because of this, to get a true ApplicationListener we must implement these missing methods ourselves.

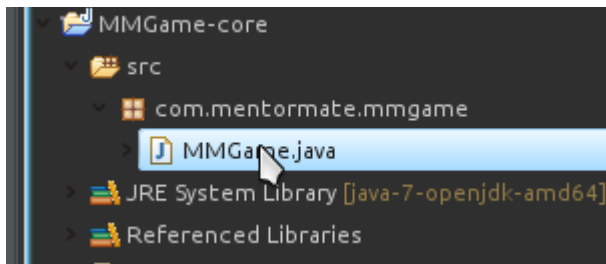
We could do this by copying the Game class and then providing the implementation to the missing method - it is actually just one, called create().

Rather than doing that, however, the better thing to do is to apply inheritance.

Inheritance is a simpler concept than interface. We just take the abstract Game class and create an extension (called a subclass), which inherits all the non-private methods and variables from the Game class as its own. Then, we can provide the subclass with its own methods and variables.

Let's implement this.

Open up the MMGame.java from Part 2. I want you to remove ALL the variables and ALL the methods and the inheritance declaration (**extends...**). Your code should look like this:



```
package com.mentormate.mmgame;

public class MMGame {
}
```

We are going to **extend** the **Game** class, which will serve as an **ApplicationInterface** between our code and platform-dependent code (on iOS, Android, etc).

1. Add **extends Game**
2. Add the following import:  
**import com.badlogic.gdx.Game;**

Importing means telling the compiler, "here is the full address to the Game class I am referring to." You need to do this because there are (potentially) many Game classes, and you want to specify this one specifically by telling the compiler where it is.

```
package com.mentormate.mmgame;

import com.badlogic.gdx.Game;

public class MMGame extends Game {

    @Override
    public void create() {
        // TODO Auto-generated method stub
    }

}
```

This is saying for MMGame to become a Game, there is a requirement: it must have a method called create().

We simply click on "Add unimplemented methods," and it will be automatically generated. Let's add one line of code to this auto-generated method:

(Note that we use Gdx.app.log instead of System.out.println(). The Gdx.app.log method is used for printing values and is implemented specifically for each platform (in Android, this would use the [Log class](#) to implement printing. On Java, this uses System.out.println()). The parameters for Gdx.app.log() should be the name of the current class and the message you would like to print).

```
package com.mentormate.mmgame;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;

public class MMGame extends Game {

    @Override
    public void create() {
        Gdx.app.log("MMGame", "created");
    }

}
```

## Let's slow things down for a minute...

Why did we want MMGame to become a Game object again?

Reason #1:

As I've previously mentioned, libGDX hides the platform-dependent code from us. All the code we would otherwise need to write in order to create a game on iOS/Android/HTML/Windows/Mac are already written for us. As game developers, we just need to provide the high-level stuff, and we do that by creating an ApplicationInterface.

By extending Game (a subclass of ApplicationInterface), MMGame becomes this interface between our code and our target platform. Now the behind-the-scenes code from Android, iOS, HTML and etc. can talk to ZBGame and work some magic together.

Reason #2:

In addition to the above, ZBGame also gains access to all the useful methods that belong to the Game class (scroll up if you want to see these again).

This is actually related to Reason #1 above. These are the methods that will eventually be called by the platform-dependent code.

Now when we run our game on one of our target platforms, the platform-dependent code will execute, and begin by calling the create() method above, printing "created."

Let's see what this all means.

We are going to attach our first Screen (which will soon be our GameScreen) to MMGame to explore this.



# Creating the GameScreen

Right click on the src folder inside the CORE MMGame project and create a new Java package called com.mentormate.mmgame.screens

Inside it, create a new class like so, importing Screen:

```
package com.mentormate.mmgame.screens;

import com.badlogic.gdx.Screen;

public class GameScreen implements Screen {

}
```

We must implement the methods in the **Screen** interface. You can do the auto-fix (as we did with MMGame) by clicking "Add unimplemented methods," or by adding the methods as I have below. Add the simple System.out.println() code that I have added to each method:

```
package com.mentormate.mmgame.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.GL20;

public class GameScreen implements Screen {

    public GameScreen() {
        Gdx.app.log("GameScreen", "Attached");
    }

    @Override
    public void render(float delta) {
        // Sets a Color to Fill the Screen with (RGB = 10, 15, 230), Opacity
        // 1 (100%)
        Gdx.gl.glClearColor(10 / 255.0f, 15 / 255.0f, 230 / 255.0f, 1f);
        // Fills the screen with the selected color
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    }

    @Override
    public void resize(int width, int height) {
        Gdx.app.log("GameScreen", "resizing");
    }

    @Override
    public void show() {
        Gdx.app.log("GameScreen", "show called");
    }

    @Override
```

```

    public void hide() {
        Gdx.app.log("GameScreen", "hide called");
    }

    @Override
    public void pause() {
        Gdx.app.log("GameScreen", "pause called");
    }

    @Override
    public void resume() {
        Gdx.app.log("GameScreen", "resume called");
    }

    @Override
    public void dispose() {
        // Leave blank
    }
}

```

## Attaching GameScreen to our MMGame Class

We want to set the current screen in the MMGame class to the GameScreen that we have just created. To do so, we go back to MMGame.java.

1. Add the following to the create() class:

```
setScreen(new GameScreen());
```

Note: This setScreen() method is available via inheritance!

2. Import GameScreen:

```
import com.mentormate.mmgame.screens.GameScreen;
```

```

package com.mentormate.mmgame;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.mentormate.mmgame.screens.GameScreen;

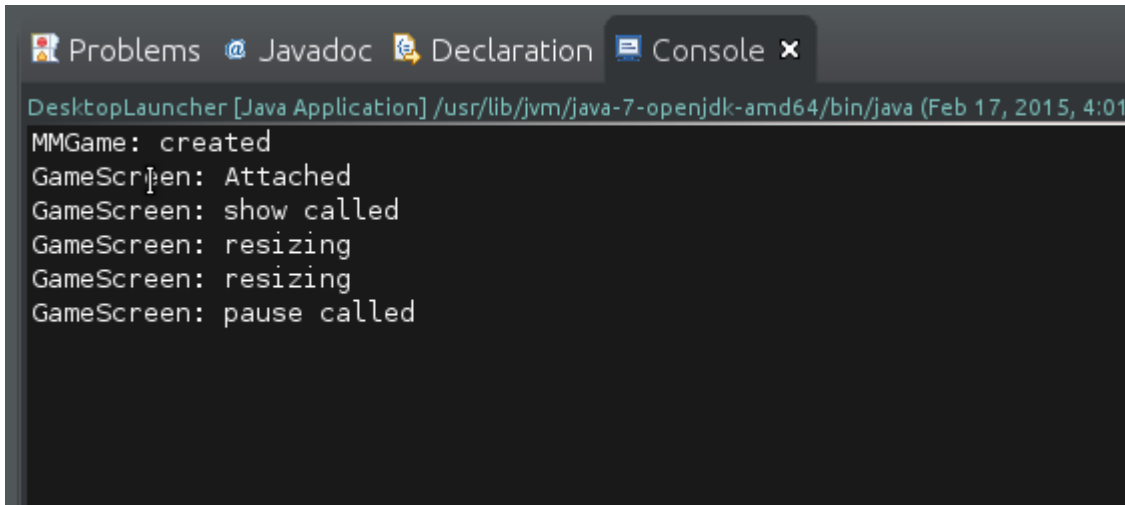
public class MMGame extends Game {

    @Override
    public void create() {
        Gdx.app.log("MMGame", "created");
        setScreen(new GameScreen());
    }

}

```

Now we can run the game. (To do this, as always, we must go to a different Project – the **MMGame – desktop** and run DesktopLauncher. You will see a beautiful blue window.



```
DesktopLauncher [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Feb 17, 2015, 4:01 PM)
MMGame: created
GameScreen: Attached
GameScreen: show called
GameScreen: resizing
GameScreen: resizing
GameScreen: pause called
```

I know, this was not the most exciting lesson, but please spend some time looking at your code again, walking through the order in which each line of code was called.

It is important that you understand this order in which various methods are called, so that we can create objects at the right time and transition smoothly in our game.

If you are ready, let's move on. Join me in Part 4, where we will implement some gameplay!

## Source Code to this Part.

[DOWNLOAD SOURCE CODE](#)

# Part 4 - GameWorld and GameRenderer and the Orthographic Camera

Welcome to Part 4! Long title, huh? In this section, we will create two helper classes for our GameScreen, so that we can start implementing our gameplay. Then we will add an orthographic camera and add some shapes to our game!

## Quick Reminder

We have three Java projects that we have generated using libGDX's setup tool;

1. Whenever I say open a class or create a new package, I will be asking you to modify the MMGame project.
2. If I ask you to run your code, you will open your MMGame-desktop project and run Main.java.
3. When we add assets (images and sounds), we will add to the MMGame-android project. All other projects will receive a copy of the assets that are in this project.

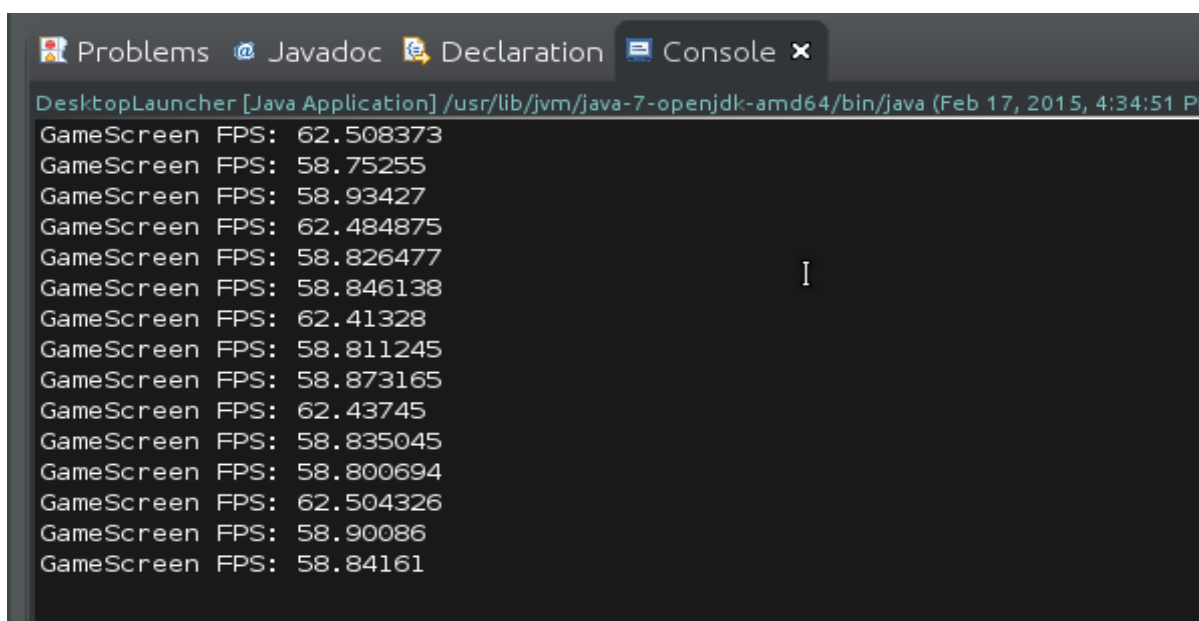
## Examining the GameScreen

In Part 3, we discussed how and when each of its methods are called. I want you to make a small change to your code:

Have a look at your render() method. It has one parameter called delta, a float. To see what this is for, I want you to add the following to your method body: `Gdx.app.log("GameScreen FPS", (1 / delta) + " ");`

```
@Override
    public void render(float delta) {
        // Sets a Color to Fill the Screen with (RGB = 10, 15, 230), Opacity
        // 1 (100%)
        Gdx.gl.glClearColor(10 / 255.0f, 15 / 255.0f, 230 / 255.0f, 1f);
        // Fills the screen with the selected color
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        // Covert Frame rate to String, print it
        Gdx.app.log("GameScreen FPS", (1 / delta) + " ");
    }
```

Try running the game (**DesktopLauncher.java inside your desktop project**). You will see the following:

A screenshot of an IDE's console window. The title bar shows 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console output shows a series of log messages from 'DesktopLauncher [Java Application]' at the path '/usr/lib/jvm/java-7-openjdk-amd64/bin/java' on Feb 17, 2015, at 4:34:51 PM. The messages are 'GameScreen FPS: 62.508373', 'GameScreen FPS: 58.75255', 'GameScreen FPS: 58.93427', 'GameScreen FPS: 62.484875', 'GameScreen FPS: 58.826477', 'GameScreen FPS: 58.846138', 'GameScreen FPS: 62.41328', 'GameScreen FPS: 58.811245', 'GameScreen FPS: 58.873165', 'GameScreen FPS: 62.43745', 'GameScreen FPS: 58.835045', 'GameScreen FPS: 58.800694', 'GameScreen FPS: 62.504326', 'GameScreen FPS: 58.90086', and 'GameScreen FPS: 58.84161'.

```
DesktopLauncher [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Feb 17, 2015, 4:34:51 PM)
GameScreen FPS: 62.508373
GameScreen FPS: 58.75255
GameScreen FPS: 58.93427
GameScreen FPS: 62.484875
GameScreen FPS: 58.826477
GameScreen FPS: 58.846138
GameScreen FPS: 62.41328
GameScreen FPS: 58.811245
GameScreen FPS: 58.873165
GameScreen FPS: 62.43745
GameScreen FPS: 58.835045
GameScreen FPS: 58.800694
GameScreen FPS: 62.504326
GameScreen FPS: 58.90086
GameScreen FPS: 58.84161
```

The float **delta** is the number of seconds (usually a small fraction) that has passed since the last time that the render method was called. When I asked you to print **1/delta**, I was asking you to print out how many times that the render method would be called in one second if that rate was sustained. This is equivalent to our FPS.

Okay, so at this point it should be clear that our **render method** can be treated as our **game loop**. In our game loop, we will do two things:

Firstly, we will update all our game objects. Secondly, we will render those game objects.

To apply good object-oriented programming principles and design patterns, we should follow these guidelines:

1. GameScreen should only do **one thing well**, so...
2. Updating the game objects should be the responsibility of **a helper class**.
3. Rendering these game objects should be the responsibility of **another helper class**.

Alright! We need two helper classes. We will give these helper classes some descriptive names: **GameWorld** and **GameRenderer**.

Create a new package called **com.mentormate.mmgame.gameworld** and create these two classes. They can be empty for now:

## GameWorld.java

```
package com.mentormate.mmgame.gameworld;

public class GameWorld {

}
```

## GameRenderer.java

```
package com.mentormate.mmgame.gameworld;

public class GameRenderer {

}
```

In our GameScreen, we are going to delegate the **updating** and **rendering** tasks to the GameWorld and GameRenderer classes, respectively. To do this, we must do the following:

1. When GameScreen is created, we must create a new GameWorld object and a new GameRenderer object.
2. Inside the **render** method of the GameScreen class, we must ask our GameWorld object and the GameRenderer object to **update and render**.

## 1. Creating the GameWorld and GameRenderer

Open up GameScreen. We are going to create GameWorld and GameRenderer objects in our

constructor. We will call their methods in our render() method. To do this:

- We need two instance variables (variables accessible anywhere inside the class). Declare the following in the class header:

```
private GameWorld world;  
private GameRenderer renderer;
```

- The GameScreen is created in the constructor. Add these lines to our constructor to initialize the above variables:

```
world = new GameWorld(); // initialize world  
renderer = new GameRenderer(); // initialize renderer
```

Add the appropriate imports:

```
import com.mentormate.mmgame.gameworld.GameRenderer;  
import com.mentormate.mmgame.gameworld.GameWorld;
```

## 2. Asking GameWorld to update and GameRenderer to render

The whole point of having the GameWorld and GameRenderer classes is so that our GameScreen doesn't have to do the updating and rendering itself. It can simply ask two other classes to do those tasks.

- Replace all the code in the render method with the following:

```
// We are passing in delta to our update method so that we can perform frame-rate independent movement.
```

```
world.update(delta); // GameWorld updates  
renderer.render(); // GameRenderer renders
```

Your completed GameScreen should look like this:

```
package com.mentormate.mmgame.screens;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.Screen;  
import com.mentormate.mmgame.gameworld.GameRenderer;  
import com.mentormate.mmgame.gameworld.GameWorld;  
  
public class GameScreen implements Screen {  
    private GameWorld world;  
    private GameRenderer renderer;  
  
    public GameScreen() {  
        Gdx.app.log("GameScreen", "Attached");  
        world = new GameWorld();  
    }  
}
```

```

        renderer = new GameRenderer();
    }

    @Override
    public void render(float delta) {
        world.update(delta);
        renderer.render();
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void show() {
        Gdx.app.log("GameScreen", "show called");
    }

    @Override
    public void hide() {
        Gdx.app.log("GameScreen", "hide called");
    }

    @Override
    public void pause() {
        Gdx.app.log("GameScreen", "pause called");
    }

    @Override
    public void resume() {
        Gdx.app.log("GameScreen", "resume called");
    }

    @Override
    public void dispose() {
        // Leave blank
    }
}

```

Eclipse will give you errors because we haven't declared the **update** method in the GameWorld and the **render** method in GameRenderer. Let's do that:

## GameWorld

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;

public class GameWorld {

    public void update(float delta) {
        Gdx.app.log("GameWorld", "update");
    }
}

```

# GameRenderer

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;

public class GameRenderer {

    public void render() {
        Gdx.app.log("GameRenderer", "render");
    }

}
```

Try running your code (Main.java in Desktop project).

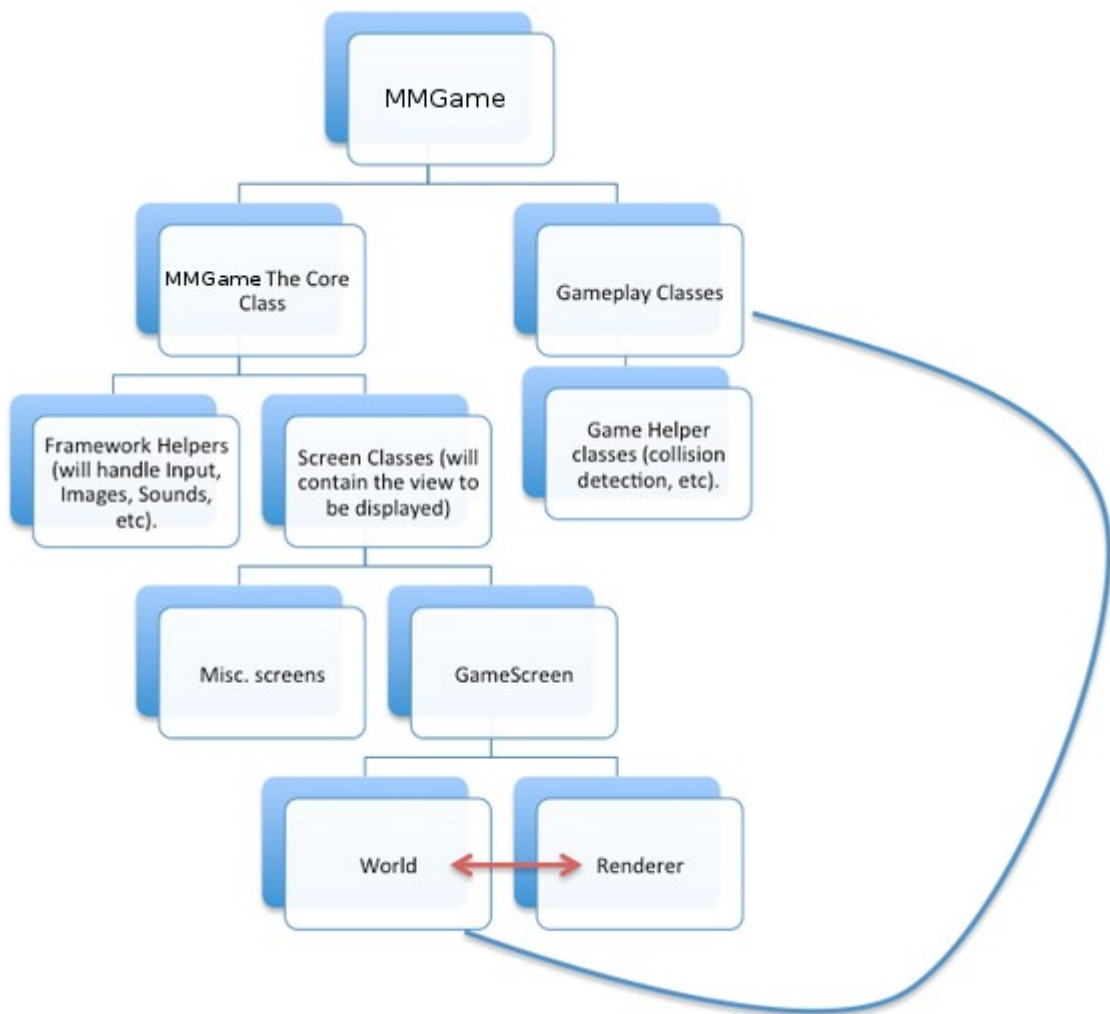
**Warning: your game may flicker (we are not drawing anything).**

You should see the following:

```
GameRenderer: render
GameWorld: update
GameRenderer: render
GameWorld: update
GameRenderer: render
GameWorld: update
GameRenderer: render
GameWorld: update
GameRenderer: render
GameWorld: update
GameRenderer: render
GameWorld: update
```

Awesome. To summarize what we have done so far, we have delegated two complex tasks (updating the game and rendering the game), so that our GameScreen does not have to worry about these two things itself. Have a look at this diagram again (can you see where we are?):





We need to make one change. Our GameRenderer needs to have a reference to the GameWorld that it will be drawing. To do this, we have to ask, "Who has a reference to both the GameRenderer and the GameWorld?" Looking at the diagram, it's pretty obvious. Let's open up our **GameScreen** and make the following changes to the constructor:

```
// This is the constructor, not the class declaration
public GameScreen() {
    Gdx.app.log("GameScreen", "Attached");
    world = new GameWorld();
    renderer = new GameRenderer(world);
}
```

This will give us an error, as the GameRenderer class does not have a constructor that takes in one input of type GameWorld. So we will create this.

### Open up the GameRenderer class.

We need to store **world** as a variable inside our GameRenderer, so that whenever we want to refer to an object inside our GameWorld, we can retrieve it.

- Create an instance variable:

```
private GameWorld myWorld;
```

- Inside the constructor, initialize this variable with the GameWorld object received from GameScreen.

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;

public class GameRenderer {

    private GameWorld myWorld;

    public GameRenderer(GameWorld world) {
        myWorld = world;
    }

    public void render() {
        Gdx.app.log("GameRenderer", "render");
    }
}
```

We will do one more thing in this lesson to illustrate how we would go about creating GameObjects and implementing them in the game. Before we do that, we are going to talk about the **Orthographic Camera**.

## Orthographic Camera

libGDX is a 3D game development framework; however, our game will be in 2D. So what does this mean for us?

Nothing really, because we can make use of something called the orthographic camera.

A lot of "2D" games that you see are actually built in 3D. Many modern platformers (even those that employ pixel art) are rendered by a 3D engine, on which developers create scenes in 3D space, rather than in 2D.



For example, have a look at this fan-made Mario game to the left, in which the world has been reconstructed using 3D models.

Looking at Mario 2.5D above, it is clear that the game is in 3D. The characters have "depth."

To make this game 2D, we might try to rotate the camera so that we are looking at the game directly from the front. Believe it or not, the game will still appear **3D**! Have a look for yourself:



Why does this happen? Because in a 3D environment (look around you), objects that are farther away appear smaller to the viewer. Even though we are looking at Mario from a perpendicular angle, some objects in this 3D world, such as the bricks, will appear smaller than those that are closer to us (the camera).

This where an orthographic camera comes in. When we use an orthographic projection, all objects in your scene, regardless of their distance, are projected onto a single plane. Imagine taking a big canvas and pushing it through the scene of the game in the second image above. When each object hits the canvas, it will be a flat, fixed-sized image. This is what an orthographic camera provides for us, and this is how we can create a 2D game in 3D space.



This is how the game might look if it used an orthographic camera:

I'm going to ask you to make one change in the **DesktopLauncher.java** inside the **Desktop Project** (that we use to run our game). We are going to change the screen resolution to the following:

```
package com.mentormate.mmgame.desktop;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;
import com.mentormate.mmgame.MMGame;

public class DesktopLauncher {
    public static void main(String[] arg) {
        LwjglApplicationConfiguration config = new
LwjglApplicationConfiguration();
        config.title = "MMGame";
        config.width = 272;
        config.height = 408;
        new LwjglApplication(new MMGame(), config);
    }
}
```

## Creating our Camera

Open up GameRenderer. We are going to create a new Orthographic Camera object.

- Declare the instance variable:

```
private OrthographicCamera cam;
```

- Import the following:

```
import com.badlogic.gdx.graphics.OrthographicCamera;
```

- Initialize the instance variable inside the constructor:

```
cam = new OrthographicCamera();
cam.setToOrtho(true, 136, 204);
```

The three arguments are asking: 1. whether you want an Orthographic projection (we do), 2. what the width should be and 3. what the height should be. This is the size of our game world. We will make changes to this at a later time. This is simply for illustration. Remember that we set our screen resolution in DesktopLauncher.java to 272 x 408. This means that everything in our game will scale by a factor of 2x when drawn.

# Creating a ShapeRenderer

To test our camera, we are going to create a ShapeRenderer object, which will draw shapes and lines for us. This is provided by libGDX!

Inside the GameRenderer,

- Declare the instance variable:

```
private ShapeRenderer shapeRenderer;
```

- Import the following:

```
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
```

- Initialize the shapeRenderer, and attach it to our camera inside the constructor:

```
shapeRenderer = new ShapeRenderer();
```

```
shapeRenderer.setProjectionMatrix(cam.combined);
```

This is what you should have so far:

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;

public class GameRenderer {

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    public GameRenderer(GameWorld world) {
        myWorld = world;
        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, 204);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);
    }

    public void render() {
        Gdx.app.log("GameRenderer", "render");
    }
}
```

Now that our ShapeRenderer is ready, we need to create something for it to render! We could create a rectangle object inside our **GameRenderer**, but that violates our design pattern. We need to create all Game Objects inside our **GameWorld** and retrieve it in **GameRenderer** to draw it.

## Open up GameWorld and make these changes:

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.math.Rectangle;

public class GameWorld {

    private Rectangle rect = new Rectangle(0, 0, 17, 12);

    public void update(float delta) {
        Gdx.app.log("GameWorld", "update");
        rect.x++;
        if (rect.x > 137) {
            rect.x = 0;
        }
    }

    public Rectangle getRect() {
        return rect;
    }
}
```

We have created a new Rectangle called rect and provided the import:

**import com.badlogic.gdx.math.Rectangle.** Notice that we do not use the Java Rectangle here as that is not available in some platforms. (the implementation for `gdx.math.Rectangle` uses the correct Rectangle based on the platform).

To make this private Rectangle accessible to anyone who has a reference to a GameWorld object, we create a getter method .

We then added some code that will allow our rectangle to scroll to the right (and reset at the left)!

Now that our Rectangle is ready, let's go back to the GameRenderer.

## Open up GameRenderer and make these changes to the RENDER method:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;

public class GameRenderer {

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    public GameRenderer(GameWorld world) {
        myWorld = world;
        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, 204);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);
    }

    public void render() {
        Gdx.app.log("GameRenderer", "render");

        /*
         * 1. We draw a black background. This prevents flickering.
         */

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        /*
         * 2. We draw the Filled rectangle
         */

        // Tells shapeRenderer to begin drawing filled shapes
        shapeRenderer.begin(ShapeType.Filled);

        // Chooses RGB Color of 87, 109, 120 at full opacity
        shapeRenderer.setColor(87 / 255.0f, 109 / 255.0f, 120 / 255.0f, 1);

        // Draws the rectangle from myWorld (Using ShapeType.Filled)
        shapeRenderer.rect(myWorld.getRect().x, myWorld.getRect().y,
            myWorld.getRect().width, myWorld.getRect().height);

        // Tells the shapeRenderer to finish rendering
        // We MUST do this every time.
        shapeRenderer.end();

        /*
         * 3. We draw the rectangle's outline
         */

        // Tells shapeRenderer to draw an outline of the following shapes
        shapeRenderer.begin(ShapeType.Line);

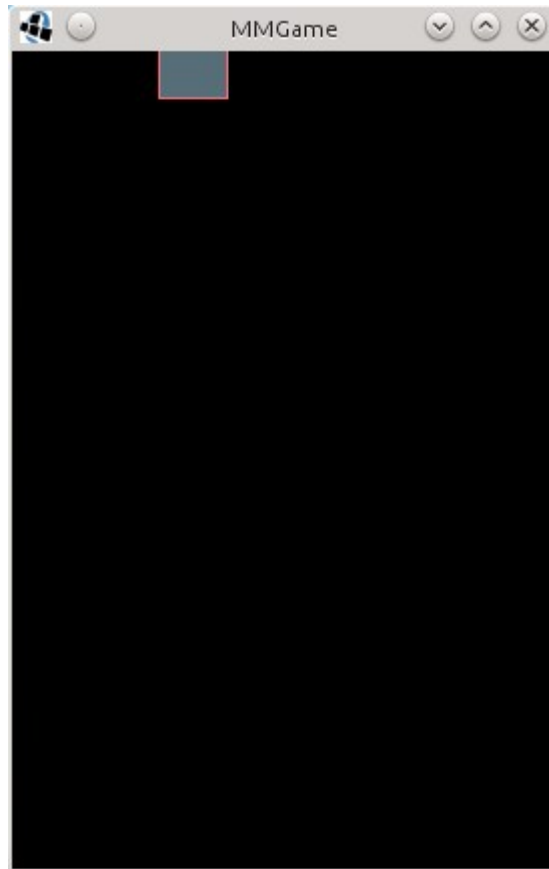
        // Chooses RGB Color of 255, 109, 120 at full opacity
        shapeRenderer.setColor(255 / 255.0f, 109 / 255.0f, 120 / 255.0f, 1);

        // Draws the rectangle from myWorld (Using ShapeType.Line)
        shapeRenderer.rect(myWorld.getRect().x, myWorld.getRect().y,

```

```
        myWorld.getRect().width, myWorld.getRect().height);  
    shapeRenderer.end();  
}  
}
```

No errors? Great! You should see something like this!



**Source Code for this Part.**

**[DWNLOA SOURCE CODE](#)**

## **Part 5 - The Flight of the Logo Adding the Logo**

Today, we are going to implement our flying Logo into the game.



## Coordinate System

We will use a Y Down coordinate system. This means that the upper left corner of the screen has the coordinates (0, 0).

What does this mean? It means that if our Logo has a positive Y velocity, we will be moving downwards.

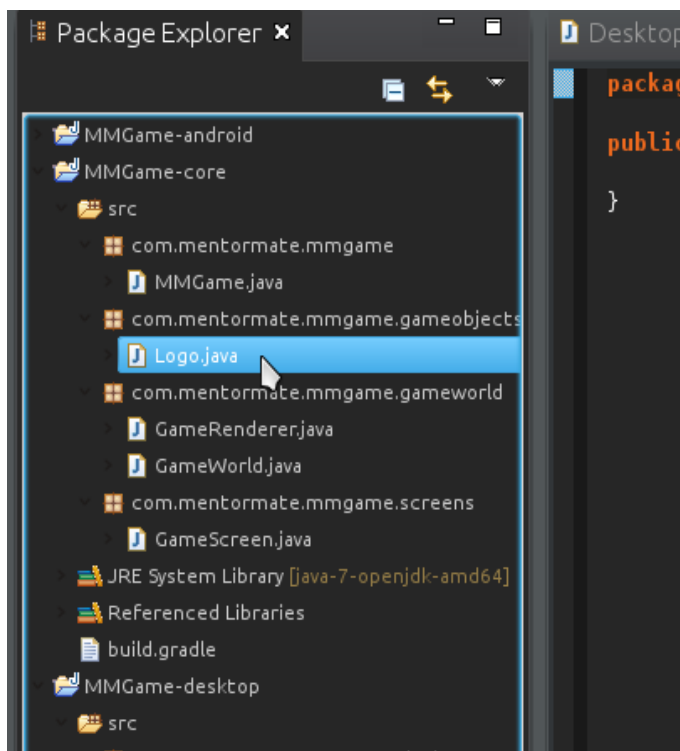
## The Screen Resolution

Our game will run on everything from iPhone to iPad to all the different Android devices out there. We need to handle screen resolution properly.

To do this, we are going to assume that the width of the game is always 136. The height will be dynamically determined! We will calculate our device's screen resolution and determine how tall the game should be.

## Creating the Logo.java class

- Create a new package called **com.mentormate.mmgame.gameobjects** and create a **Logo** class.



### The Instance Variables:

Our Logo needs a position, a velocity and acceleration (more on this below). We also need a rotation angle (for when the logo rotates) and a width and height.

```

private Vector2 position;
private Vector2 velocity;
private Vector2 acceleration;

private float rotation; // For handling logo rotation
private int width;
private int height;

```

Vector2 is a powerful built-in libGDX class. If you are not familiar with mathematical vectors, that's okay! We are just going to treat it as an object that can hold two values: an x component and a y component.

**position.x** then refers to the x coordinate, and **velocity.y** would correspond to the speed in the y direction. **Acceleration** just means change in velocity, just like **velocity** means change in **position**.

This will become much clearer in just a second.

## The Constructor:

What information do we need to create a new Logo? We need to know what its X position should be, what its Y position should be, how wide it should be and how tall it should be.

```

public Logo(float x, float y, int width, int height) {
    this.width = width;
    this.height = height;
    position = new Vector2(x, y);
    velocity = new Vector2(0, 0);
    acceleration = new Vector2(0, 460);
}

```

Our Logo will belong to the GameWorld, and these are the methods we need:

1. The update method (which will be called when GameWorld updates)
2. onClick method (which will be called when the screen is clicked or touched).

We also need to create a bunch of getters to allow access to some of our Bird object's instance variables.

```

package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Vector2;

public class Logo {

    private Vector2 position;
    private Vector2 velocity;
    private Vector2 acceleration;

```

```

private float rotation; // For handling bird rotation
private int width;
private int height;

public Logo(float x, float y, int width, int height) {
    this.width = width;
    this.height = height;
    position = new Vector2(x, y);
    velocity = new Vector2(0, 0);
    acceleration = new Vector2(0, 460);
}

public void update(float delta) {

    velocity.add(acceleration.cpy().scl(delta));

    if (velocity.y > 200) {
        velocity.y = 200;
    }

    position.add(velocity.cpy().scl(delta));
}

public void onClick() {
    velocity.y = -140;
}

public float getX() {
    return position.x;
}

public float getY() {
    return position.y;
}

public float getWidth() {
    return width;
}

public float getHeight() {
    return height;
}

public float getRotation() {
    return rotation;
}
}

```

The logic for this is pretty simple. Every time that the Logo's **update** method is called, we do two things:

1. We add our **scaled** acceleration vector (we will come back to this) to our velocity vector. This gives us our new velocity. For those of you not familiar, this is how the earth's gravity works. Your downward speed increases by 9.8 m/s every second.
2. Remember that Flappy Bird physics has a max velocity cap (there's some sort of terminal velocity). I have experimented with this and set a velocity.y cap at 200.

3. We add the updated **scaled** velocity to the logo's position (this gives us our new position).

What do I mean by scaled in #1 and #3 above? I mean that we multiply the acceleration and velocity vectors by the **delta**, which is the amount of time that has passed since the update method was **previously called**. This has a normalizing effect.

If your game slows down for any reason, your **delta** will go up (your processor will have taken longer time to complete the previous iteration, or repetition, of the loop). By scaling our Vectors with **delta**, we can achieve **frame-rate independent movement**. If the update method took twice as long to execute, then we just move our character by 2x the original velocity, and so on.

We will be applying this scaling to our rotation later on!

Now that our logo is ready, we will unleash it into the GameWorld!

## Caution

Every time you create a new Object, you allocate a little bit of memory in RAM for that object (specifically in the Heap). Once your Heap fills up, a subroutine called a Garbage Collector will come in and clean up your memory to ensure that you do not run out of space. This is great, except when you are building a game. Every time the garbage collector comes into play, your game will stutter for several essential milliseconds. To avoid garbage collection, you need to avoid allocating new objects whenever possible.

I've recently discovered that the method `Vector2.cpy()` creates a new instance of a `Vector2` rather than recycling one instance. This means that at 60 FPS, calling `Vector2.cpy()` would allocate 60 new `Vector2` objects every second, which would cause the Java garbage collector to get involved every frequently.

Just keep this in mind as you go through the remainder . We will solve this issue later in the tutorial.

## Open the GameWorld class

Let's remove the rectangle object we have created earlier. This is what you should have:

```
package com.mentormate.mmgame.gameworld;

public class GameWorld {

    public void update(float delta) {

    }

}
```

Let's first create a constructor for our GameWorld. Add the following constructor above the update method:

```
public GameWorld() {
```

```
}
```

Import Logo, and create a new Logo object as an instance variable (do not initialize it). Create a getter too. And call the logo's update method in the **GameWorld.update(float delta)**. This is what that should look like:

```
package com.mentormate.mmgame.gameworld;

import com.mentormate.mmgame.gameobjects.Logo;

public class GameWorld {
    private Logo logo;

    public GameWorld() {
        // Initialize bird here
    }

    public void update(float delta) {
        logo.update(delta);
    }

    public Logo getBird() {
        return logo;
    }
}
```

Now we must initialize the logo. What information do we need? The **x**, **y**, **width** and **height** (these are the four values we need to call the Logo's constructor that we've created above).

The **x** should be 33 (that's where the bird stays the entire time). The width should be **17**. The height will be **12**.

What about the **y**? Based on my calculations, it should be 5 pixels above the vertical middle of the screen. (Remember that we are scaling everything down to a **137 x ???** screen resolution, where the height is determined by getting the ratio between device screen height and screen width, and multiplying by 137).

Add this line to the constructor:

```
logo = new Logo(33, midPointY - 5, 17, 12);
```

So how do we get midPointY? We will ask our GameScreen to give it to us. Remember that the GameWorld constructor is invoked when the GameScreen creates a GameWorld object. So then we can create an additional parameter to ask GameScreen to give us the midPointY.

Add this to the constructor: `(int midPointY)`

This is what you should have at this point:

```
package com.mentormate.mmgame.gameworld;

import com.mentormate.mmgame.gameobjects.Logo;

public class GameWorld {
    private Logo logo;

    public GameWorld(int midPointY) {
        logo = new Logo(33, midPointY - 5, 17, 12);
    }

    public void update(float delta) {
        logo.update(delta);
    }

    public Logo getLogo() {
        return logo;
    }
}
```

Now we need to go make some changes to our GameScreen.

We have an error, as expected, in the line where we call the GameWorld constructor. We have just said that **"to create a new GameWorld, you must give us an integer"**, so we must do that!

Let's ask GameScreen to calculate the midPointY of the screen and pass it into the constructor.

When I say midPointY, this is what I mean. Remember that our game will be 136 units wide. Our screen may be 1080 pixels wide, so we must scale everything down by about 1/8. To get the game height, we must take the screen height and scale that down by the same factor!

We can retrieve the screen width and height using the following methods: `Gdx.graphics.getWidth()` and `Gdx.graphics.getHeight()`.

Let's use this information to implement our logic inside the constructor:

```
package com.mentormate.mmgame.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.mentormate.mmgame.gameworld.GameRenderer;
import com.mentormate.mmgame.gameworld.GameWorld;

public class GameScreen implements Screen {

    private GameWorld world;
    private GameRenderer renderer;
```

```

// This is the constructor, not the class declaration
public GameScreen() {

    float screenWidth = Gdx.graphics.getWidth();
    float screenHeight = Gdx.graphics.getHeight();
    float gameWidth = 136;
    float gameHeight = screenHeight / (screenWidth / gameWidth);

    int midPointY = (int) (gameHeight / 2);

    world = new GameWorld(midPointY);
    renderer = new GameRenderer(world);

}

@Override
public void render(float delta) {
    world.update(delta);
    renderer.render();
}

@Override
public void resize(int width, int height) {

}

@Override
public void show() {
    Gdx.app.log("GameScreen", "show called");
}

@Override
public void hide() {
    Gdx.app.log("GameScreen", "hide called");
}

@Override
public void pause() {
    Gdx.app.log("GameScreen", "pause called");
}

@Override
public void resume() {
    Gdx.app.log("GameScreen", "resume called");
}

@Override
public void dispose() {
    // Leave blank
}
}

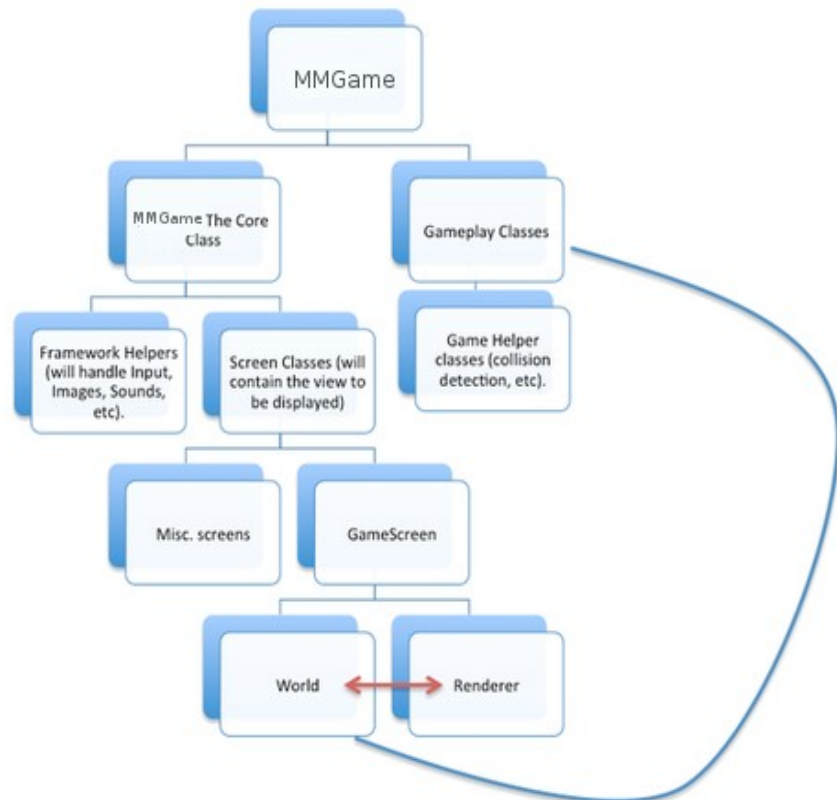
```

Now that we have created our Bird, we need to be able to control it. So let's add an input handler!

The diagram is back!  
We will briefly turn our attention to the **Framework Helpers** on the 3rd level. The MMGame needs help in order to handle input, images, sounds and etc.

We will be creating two classes right now.

The first class will be the **InputHandler** which, as the name suggests, will react to various inputs. In our case, the only input we need to worry about is the touch (clicks are registered as touches on PC/Mac).

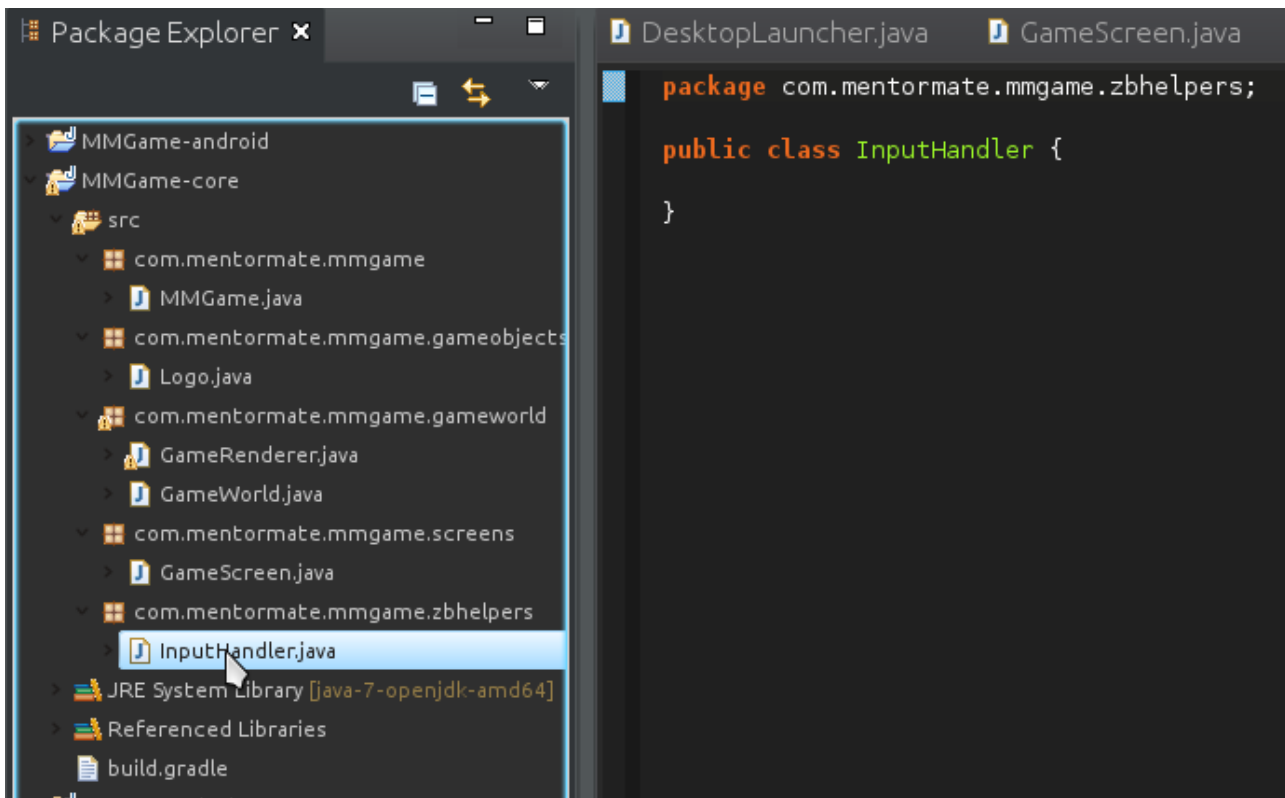


The second class will be the **AssetLoader** which will load various images, animations and sounds for us.

We will get back to the AssetLoader very soon. Let's first implement our InputHandler.

Create the package called **com.mentormate.mmgame.zbhelpers**, and create an **InputHandler** class.





Your code should look like this:

```
package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.InputProcessor;

public class InputHandler implements InputProcessor {

    @Override
    public boolean touchDown(int screenX, int screenY, int pointer, int
button) {
        return false;
    }

    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button)
{
        return false;
    }
}
```

```

@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    return false;
}

@Override
public boolean mouseMoved(int screenX, int screenY) {
    return false;
}

@Override
public boolean scrolled(int amount) {
    return false;
}
}

```

That gives us a bunch of methods that we can work with. For now, we only have to worry about the `touchDown()` method, however.

The `TouchDown` method should call our `Logo`'s **onClick** method, but we have no reference to our `Logo` object. We can't call any of `Logo`'s methods until we get a reference to our `Logo`. Then we ask ourselves, who has a reference to our current **logo**? **GameWorld** does, which belongs to **GameScreen**! So we will ask `GameScreen` to send the `Logo` to us.

Before we go back to `GameScreen`, let's first finish up the `InputHandler` class:

1. Create an instance variable for the `InputHandler` class:

```
private Logo myLogo;
```

2. We must ask for a `Logo` object inside the constructor:

```
public InputHandler(Logo logo) {
    myLogo = logo;
}

```

3. And then we can call the following inside our `touchDown()` method:  
`myLogo.onClick()`

```

package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.InputProcessor;
import com.mentormate.mmgame.gameobjects.Logo;

public class InputHandler implements InputProcessor {

    private Logo myLogo;

    // Ask for a reference to the Bird when InputHandler is created.
    public InputHandler(Logo logo) {
        // myBird now represents the gameWorld's bird.
        myLogo = logo;
    }

    @Override

```

```

    public boolean touchDown(int screenX, int screenY, int pointer, int
button) {
        myLogo.onClick();
        return true; // Return true to say we handled the touch.
    }

    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button)
{
        return false;
    }

    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
        return false;
    }

    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        return false;
    }

    @Override
    public boolean scrolled(int amount) {
        return false;
    }
}

```

Now we just have to go back to the GameScreen and create a new InputHandler, and attach it to our game!  
Open that up:

Update your constructor to be as follows: In the very last line, we are telling libGDX to take our new InputHandler as its processor.

```

public GameScreen() {

    float screenWidth = Gdx.graphics.getWidth();
    float screenHeight = Gdx.graphics.getHeight();
    float gameWidth = 136;
    float gameHeight = screenHeight / (screenWidth / gameWidth);
}

```

```

        int midPointY = (int) (gameHeight / 2);

        world = new GameWorld(midPointY);
        renderer = new GameRenderer(world);

        Gdx.input.setInputProcessor(new InputHandler(world.getLogo()));
    }

```

Gdx.input.setInputProcessor() takes in an InputProcessor object. Since we implemented InputProcessor in our very own InputHandler, we can give our InputHandler to this instead.

Notice that we are calling the constructor, passing in a reference to our Logo object that we retrieve from World. This is just a simplification of the following:

```

Logo logo = world.getLogo();
InputHandler handler = new InputHandler(logo);
Gdx.input.setInputProcessor(handler);

```

Rather than type all this, the one-line solution in the code above should work fine.

## Now where are we?

We have created our Logo class, created a Logo object inside our GameWorld, and created an InputHandler which will call our Logo's onClick method, which will make it fly upwards!

## Source Code.

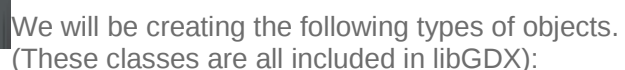
[DOWNLOAD SOURCE CODE](#)

# Part 6 - Adding Graphics

In this part, we will create our AssetLoader object, load an animation and a bunch of textures, and use our Renderer to draw our logo and his background .

## The AssetLoader class

We begin by creating the **AssetLoader** class in the **com.mentormate.mmgame.zbhelpers** package. (You should have lingering errors from GameRenderer)



**Animation** - we can take multiple texture regions and create an Animation object to specify how to animate the Logo.

**Do not download the image below! It has been scaled by a factor of 4, so it will not work with the code. Instead, download the file that I provide a little further down in this lesson.**



## The Full AssetLoader class:

```
package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture;
    public static TextureRegion bg;
    public static TextureRegion grass;

    public static Animation logoAnimation;
    public static TextureRegion logo, logoDown, logoUp;

    public static TextureRegion barTopUp, barTopDown, bar;

    public static void load() {

        texture = new Texture(Gdx.files.internal("data/texture.png"));
        texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

        bg = new TextureRegion(texture, 0, 0, 136, 43);
        bg.flip(false, true);

        grass = new TextureRegion(texture, 0, 43, 143, 11);
        grass.flip(false, true);

        logoDown = new TextureRegion(texture, 136, 0, 17, 12);
        logoDown.flip(false, true);

        logo = new TextureRegion(texture, 153, 0, 17, 12);
        logo.flip(false, true);

        logoUp = new TextureRegion(texture, 170, 0, 17, 12);
        logoUp.flip(false, true);

        TextureRegion[] birds = { logoDown, logo, logoUp };
        logoAnimation = new Animation(0.06f, birds);
        logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

        barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
        // Create by flipping existing skullUp
        barTopDown = new TextureRegion(barTopUp);
        barTopDown.flip(false, true);

        bar = new TextureRegion(texture, 136, 16, 22, 3);
        bar.flip(false, true);

    }

    public static void dispose() {
        // We must dispose of the texture when we are finished.
    }
}
```

```
        texture.dispose();  
    }  
}
```

Let's walk through the code a little bit. It is full of static methods and variables, meaning that we will not be creating instances of the Asset class - there will be only one copy.

It has two methods: the load method and the dispose method.

The load method will be called when the game starts up, and the dispose method will be called when the game is being closed.

## Examining load()

### Texture

The load method begins by creating a new Texture object with the file texture.png, which I will provide below. It sets its minification and magnification filters (used when resizing to a smaller or larger image) to the enum constant: **TextureFilter.Nearest**. This is important because when our small pixel art is stretched to a larger size, each pixel will retain its shape rather than becoming blurry!

### TextureRegion

We can use our texture to create TextureRegion objects, which require five arguments: the pertinent Texture object and the rectangular boundaries of the desired region on that Texture. This is given as x, y, width and height starting from the top left of the image, so our background image would be 0, 0, 136, 43.

TextureRegions must be flipped, because libGDX assumes a Y Up coordinate system by default. We are using a Y Down coordinate system, and must flip each image (except for skullUp, which can remain upside down)!

### Animation

We can create an array of TextureRegion objects and pass it in to the constructor of a new Animation object:

```
TextureRegion[] logos = { logoDown, logo, logoUp }; // creates an array of TextureRegions  
logoAnimation = new Animation(0.06f, logos); // Creates a new Animation in which each frame is 0.06  
seconds long, using the above array.  
logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG); // Sets play mode to be ping  
pong, in which we will see a bounce.
```

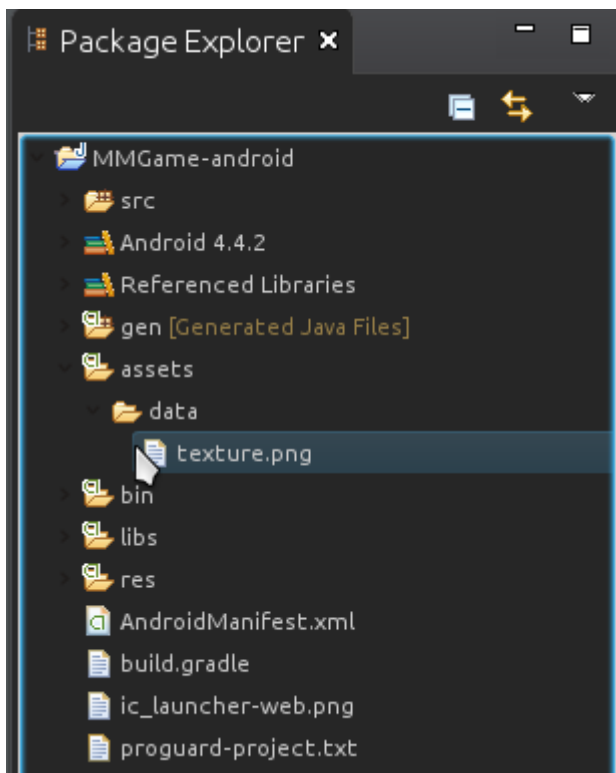
We gave the Animation 3 frames. It will now change frames every 0.06 seconds (down, middle, up, middle, down ...).

## Download the Texture file

Download the provided texture file below, and place it inside the MMGame-android project's assets/data/ folder! This is VERY important.

A note on using images: whenever you update your textures (which you will if you use your own images), you have to clean your projects on Eclipse for it to update immediately.

## DOWNLOAD LINK



Make sure that you have placed your image inside the correct folder as shown to the left (note we are inside the **MMGame-android** project. You can delete the libgdx.png file that is included with the project.

If you are set, make sure you have cleaned your project and move on.

## Calling the Load Method

Now that our AssetLoader is ready (and you have downloaded the texture into the CORRECT FOLDER and cleaned your project), we open up the MMGame class, so that we can load all the images before the GameScreen is created. We add the line in our create method (before the GameScreen is created):

```
AssetLoader.load(); (Import com.mentormate.mmgame.zbhelpers.AssetLoader)
```

We must also call AssetLoader.dispose() when the dispose method of our MMGame is called by our behind-the-scenes platform-dependent code. To do this, we override the existing dispose method from MMGame's superclass, Game.

(the full code follows):

```
package com.mentormate.mmgame;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.mentormate.mmgame.screens.GameScreen;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class MMGame extends Game {
```



```

@Override
public void create() {
    Gdx.app.log("MMGame", "created");
    AssetLoader.load();
    setScreen(new GameScreen());
}

@Override
public void dispose() {
    super.dispose();
    AssetLoader.dispose();
}
}

```

Now that all of our images are loaded, we can start rendering them inside the GameRenderer!

Let's open that up.

To draw a TextureRegion, we need to create a **SpriteBatch** (just like we had to create a ShapeRenderer). The SpriteBatch draws images for us using the indices provided. (x, y, width and height, typically) Let's remove all the non-essential code from **GameRenderer** and create this SpriteBatch, as shown below.

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;

public class GameRenderer {

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    public GameRenderer(GameWorld world) {
        myWorld = world;
        cam = new OrthographicCamera();
        cam.setToOrtho(true, 137, 204);

        batcher = new SpriteBatch();
        // Attach batcher to camera
        batcher.setProjectionMatrix(cam.combined);

        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);
    }

    public void render() {

        Gdx.gl.glClearColor(0, 0, 0, 1);
    }
}

```

```

        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    }
}

```

We must change our camera width to 136 and we must change the height to the game height determined by the GameScreen. To do this, we will change our constructor to ask for the gameHeight and the midPointY.

Add these **TWO NEW** instance variables (keep the other four also) and change the constructor as follows (make sure you change width and height properly to 136 and gameHeight), and store the gameHeight and midPointY variables for future usage:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;

public class GameRenderer {

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);
    }

    public void render() {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    }
}

```

Next, We must also add a parameter to the render method:

```
public void render(float runTime) {  
    ...  
}
```

This value is needed to determine which frame the logo animation should display. The Animation object will use this value (and the previously determined frame duration) to determine which TextureRegion to display.

Now that we have updated the constructor, so we must fix the errors that arise in GameScreen.

Open the GameScreen class and replace the following line:

```
renderer = new GameRenderer(world);  
with this one:  
renderer = new GameRenderer(world, (int) gameHeight, midPointY);
```

We also need to create one extra variable called **runTime**, which will keep track of how long the game has been running for. We will pass this into the render method of the GameRenderer!

- Create an instance variable called runTime and initialize it at 0.  
**private float runTime = 0;**

Inside the render(float delta) method, **increment runTime by delta**, and pass it into the render method (where we will use this value to render the animation properly):

```
@Override  
public void render(float delta) {  
    runTime += delta;  
    world.update(delta);  
    renderer.render(runTime);  
}
```

Your GameScreen should look like this:

```
package com.mentormate.mmgame.screens;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.Screen;  
import com.mentormate.mmgame.gameworld.GameRenderer;  
import com.mentormate.mmgame.gameworld.GameWorld;  
import com.mentormate.mmgame.zbhelpers.InputHandler;  
  
public class GameScreen implements Screen {  
  
    private GameWorld world;  
    private GameRenderer renderer;  
    private float runTime;  
  
    // This is the constructor, not the class declaration  
    public GameScreen() {
```

```

        float screenWidth = Gdx.graphics.getWidth();
        float screenHeight = Gdx.graphics.getHeight();
        float gameWidth = 136;
        float gameHeight = screenHeight / (screenWidth / gameWidth);

        int midPointY = (int) (gameHeight / 2);

        world = new GameWorld(midPointY);
        renderer = new GameRenderer(world, (int) gameHeight, midPointY);

        Gdx.input.setInputProcessor(new InputHandler(world.getLogo()));
    }

    @Override
    public void render(float delta) {
        runTime += delta;
        world.update(delta);
        renderer.render(runTime);
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void show() {
        Gdx.app.log("GameScreen", "show called");
    }

    @Override
    public void hide() {
        Gdx.app.log("GameScreen", "hide called");
    }

    @Override
    public void pause() {
        Gdx.app.log("GameScreen", "pause called");
    }

    @Override
    public void resume() {
        Gdx.app.log("GameScreen", "resume called");
    }

    @Override
    public void dispose() {
        // Leave blank
    }
}

```

Go back to the GameRenderer, and change your render method to the one below:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;

```

```

import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameRenderer {

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);
    }

    public void render(float runTime) {

        // We will move these outside of the loop for performance later.
        Logo logo = myWorld.getLogo();

        // Fill the entire screen with black, to prevent potential
        flickering.
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        // Begin ShapeRenderer
        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(55 / 255.0f, 80 / 255.0f, 100 / 255.0f, 1);
        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        // End ShapeRenderer
    }
}

```

```

        shapeRenderer.end();

        // Begin SpriteBatch
        batcher.begin();
        // Disable transparency
        // This is good for performance when drawing images that do not
require
        // transparency.
        batcher.disableBlending();
        batcher.draw(AssetLoader.bg, 0, midPointY + 23, 136, 43);

        // The bird needs transparency, so we enable that again.
        batcher.enableBlending();

        // Draw logo at its coordinates. Retrieve the Animation object from
        // AssetLoader
        // Pass in the runTime variable to get the current frame.
        batcher.draw(AssetLoader.logoAnimation.getKeyFrame(runTime),
                    logo.getX(), logo.getY(), logo.getWidth(),
logo.getHeight());

        // End SpriteBatch
        batcher.end();
    }
}

```

Try running the code (DesktopLauncher.java) and START CLICKING (otherwise your logo will fall down to oblivion)! It should look like this:



Let's now go back to the render method and see what the logic is there. We always draw the background first, because drawing is done in layers. We begin by drawing some flat colors. We are choosing to use a color filled Shape rather than use a TextureRegion to fill the background.

We draw a temporary green rectangle to show where the grass will be, and a brown one for the dirt.

We then begin the SpriteBatch, again, starting by drawing the background image of the city. On top of that, we retrieve the current TextureRegion of our Animation using our runTime, and draw the logo with blending enabled. Read the comments that I have made inside the render() method! They are important.

## Source Code

[DOWNLOAD SOURCE CODE](#)

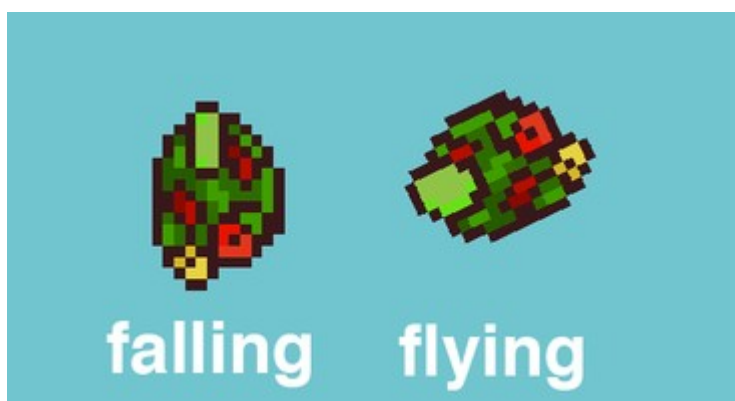
# Part 7 - The Grass, the Logo and the Pipe

We will learn how to rotate our logo, and how to scroll the grass and the pipe. This will involve making some changes to the Logo class, and creating new classes to represent our grass and the pipe.

Let's get started!

## Rotating the Logo

Before we get to the code, let's examine the Bird's rotating behavior first. In **Flappy Bird** the bird has two primary states. The bird is either rising after a touch or the bird is falling. In those two states, the bird rotates as follows:



Rotation is controlled using the Y velocity. In our game, the bird accelerates downwards due to gravity (meaning that the velocity increases). When our velocity is negative (meaning that the logo is moving upwards), the logo will start rotating counterclockwise. When our velocity is higher than some positive value, the logo will start to rotate clockwise (we don't start rotating until the Logo starts speeding up a bit).

## Animations

One thing we have to pay attention to is the animation. The logo should not flap when falling. Instead, it should go in the middle position. Once he starts rising, he will start flapping again.

Let's implement all of these in code:

- We begin by creating two methods. We are going to use experimentally determined hard-coded values to create the following methods anywhere inside the Logo class:

```
public boolean isFalling() {  
    return velocity.y > 110;  
}  
  
public boolean shouldntFlap() {  
    return velocity.y > 70;  
}
```

We will use the isFalling method to decide when the logo should begin rotating downwards. We will use the shouldntFlap method to determine when the logo should stop animating.

- Next, we need to make some changes to the update method. Recall that we have a float called rotation, which will keep track of how much our logo has rotated. We will say that a positive change in rotation is a clockwise rotation and that a negative change in rotation is a counterclockwise rotation.

Add these two blocks of code to the end of the update method. They will handle the counterclockwise and clockwise rotations (rising and falling).

Note that we scale our rotation by delta, so that the logo will rotate at the same rate even if the game slows down (or speeds up, for that matter).

Both of these if statements have some kind of cap on rotation. If we rotate too far, our game will correct that for us.

Here is what your full Logo class should look like:

```
package com.mentormate.mmgame.gameobjects;  
  
import com.badlogic.gdx.math.Vector2;  
  
public class Logo {  
  
    private Vector2 position;  
    private Vector2 velocity;  
    private Vector2 acceleration;  
  
    private float rotation; // For handling bird rotation  
    private int width;  
    private int height;  
  
    public Logo(float x, float y, int width, int height) {  
        this.width = width;  
        this.height = height;  
        position = new Vector2(x, y);  
        velocity = new Vector2(0, 0);  
        acceleration = new Vector2(0, 460);  
    }  
  
    public void update(float delta) {  
  
        velocity.add(acceleration.cpy().scl(delta));
```



```

        if (velocity.y > 200) {
            velocity.y = 200;
        }

        position.add(velocity.cpy().scl(delta));

        // Rotate counterclockwise
        if (velocity.y < 0) {
            rotation -= 600 * delta;

            if (rotation < -20) {
                rotation = -20;
            }
        }

        // Rotate clockwise
        if (isFalling()) {
            rotation += 480 * delta;
            if (rotation > 90) {
                rotation = 90;
            }
        }
    }

    public boolean isFalling() {
        return velocity.y > 110;
    }

    public boolean shouldntFlap() {
        return velocity.y > 70;
    }

    public void onClick() {
        velocity.y = -140;
    }

    public float getX() {
        return position.x;
    }

    public float getY() {
        return position.y;
    }

    public float getWidth() {
        return width;
    }

    public float getHeight() {
        return height;
    }

    public float getRotation() {
        return rotation;
    }
}

```

Great. Now that we've gotten that taken care of, all we need to do is go to the GameRenderer and use

the **shouldntFlap** method to determine whether the bird should animate or not.

## Caution

As mentioned in Part 5 (Reiterated here because of its importance!)

Every time you create a new Object, you allocate a little bit of memory in RAM for that object (specifically in the Heap). Once your Heap fills up, a subroutine called a Garbage Collector will come in and clean up your memory to ensure that you do not run out of space. This is great, except when you are building a game. Every time the garbage collector comes into play, your game will stutter for several essential milliseconds. To avoid garbage collection, you need to avoid allocating new objects whenever possible

I've recently discovered that the method `Vector2.cpy()` creates a new instance of a `Vector2` rather than recycling one instance. This means that at 60 FPS, calling `Vector2.cpy()` would allocate 60 new `Vector2` objects every second, which would cause the Java garbage collector to get involved every frequently.

## Cleaning up GameRenderer

To achieve high performance in games, you have to minimize the work that you do in your game loop. In Part 6, we wrote some code that violates this principle. Have a look at the render method. We had this line of code:

```
Logo logo = myWorld.getLogo();
```

Each time that the render method was called (about 60 times per second), we were asking our game to find `myWorld`, then find its `Logo` object, return it to the `GameRenderer` to put it on the stack as a local variable.

Instead of doing that, we are going to retrieve it once when the `GameRenderer` is first created then store the `Logo` object as an instance variable.

We will be doing the same thing with our `TextureRegions` from `AssetLoader` and for any future objects that we create.

- Begin by adding these instance variables to `GameRenderer` (importing as necessary-- use `com.badlogic.gdx.graphics.g2d.Animation` for animation):

```
// Game Objects  
private Logo logo;
```

```
// Game Assets
```

```
private TextureRegion bg, grass;
private Animation logoAnimation;
private TextureRegion logoMid, logoDown, logoUp;
private TextureRegion barTopUp, barTopDown, bar;
```

Next, we will initialize these in the constructor. But rather than doing all the work in the constructor and making the code messy, we are going to create two helper methods that will initialize them for us:

- Add the following two methods to the GameRenderer:

```
private void initGameObjects() {
    logo = myWorld.getLogo();
}

private void initAssets() {
    bg = AssetLoader.bg;
    grass = AssetLoader.grass;
    logoAnimation = AssetLoader.logoAnimation;
    logoMid = AssetLoader.logo;
    logoDown = AssetLoader.logoDown;
    logoUp = AssetLoader.logoUp;
    barTopUp = AssetLoader.barTopUp;
    barTopDown = AssetLoader.barTopDown;
    bar = AssetLoader.bar;
}
```

```
public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
    myWorld = world;

    // The word "this" refers to this instance.
    // We are setting the instance variables' values to be that of the
    // parameters passed in from GameScreen.
    this.gameHeight = gameHeight;
    this.midPointY = midPointY;

    cam = new OrthographicCamera();
    cam.setToOrtho(true, 136, gameHeight);

    batcher = new SpriteBatch();
    batcher.setProjectionMatrix(cam.combined);
    shapeRenderer = new ShapeRenderer();
    shapeRenderer.setProjectionMatrix(cam.combined);

    // Call helper methods to initialize instance variables
    initGameObjects();
    initAssets();
}
```

Lastly, we must make changes to the **render** method. Specifically, we will remove any references to AssetLoader, and remove the line of code that says:  
Logo logo = myWorld.getLogo().

Secondly, we will modify our logo drawing calls so that we can handle rotations properly. Also change any references to AssetLoader (such as AssetLoader.bg) Here's how you might do that:

```
public void render(float runTime) {

    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    shapeRenderer.begin(ShapeType.Filled);

    // Draw Background color
    shapeRenderer.setColor(55 / 255.0f, 80 / 255.0f, 100 / 255.0f, 1);
    shapeRenderer.rect(0, 0, 136, midPointY + 66);

    // Draw Grass
    shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f,
1);
    shapeRenderer.rect(0, midPointY + 66, 136, 11);

    // Draw Dirt
    shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 77, 136, 52);

    shapeRenderer.end();

    batcher.begin();
    batcher.disableBlending();
    batcher.draw(bg, 0, midPointY + 23, 136, 43);

    batcher.enableBlending();

    if (logo.shouldntFlap()) {
        batcher.draw(logoMid, logo.getX(), logo.getY(),
            logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
            logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
    } else {
        batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
            logo.getY(), logo.getWidth() / 2.0f,
            logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
            1, 1, logo.getRotation());
    }

    batcher.end();

}
```

And your full GameRenderer should look like this:

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
```

```

import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
    private TextureRegion bg, grass;
    private Animation logoAnimation;
    private TextureRegion logoMid, logoDown, logoUp;
    private TextureRegion barTopUp, barTopDown, bar;

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);

        // Call helper methods to initialize instance variables
        initGameObjects();
        initAssets();
    }

    private void initGameObjects() {
        logo = myWorld.getLogo();
    }

    private void initAssets() {
        bg = AssetLoader.bg;
        grass = AssetLoader.grass;
        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        logoDown = AssetLoader.logoDown;
        logoUp = AssetLoader.logoUp;
    }
}

```

```

        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(55 / 255.0f, 80 / 255.0f, 100 / 255.0f, 1);
        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f,
1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        shapeRenderer.end();

        batcher.begin();
        batcher.disableBlending();
        batcher.draw(bg, 0, midPointY + 23, 136, 43);

        batcher.enableBlending();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
        } else {
            batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                logo.getY(), logo.getWidth() / 2.0f,
                logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                1, 1, logo.getRotation());
        }

        batcher.end();
    }
}

```

Now try running the code! Your logo will fly and rotate, exactly as we intended.

# Creating the Scrollable Classes

We will now implement the grass ,androidLogo , the appleLogo and the pipes in our game:

They move at the same speed to the left. We will work with the following parameters:

- We will use 2 pipes (each column will be considered one pipe object).
- We will use 2 bonusLogo Objects
- We will use 2 long strips of grass, attached to each other horizontally.
- When each of the items scrolls to the left (is no longer visible), we will reset it.
- To reset the grass, we will simply attach it to the tail (the end) of the other piece.
- To reset the pipe, we place it at a fixed distance from the pipe that is furthest back, and reset its height.

Notice that those items behave in very similar ways. Because of this, we can take their common traits and create a **Scrollable** Superclass with them, and then use inheritance to create the more specific Pipe BonusLogo and Grass objects.

## The resetting part can be tricky.

For example, if we have three pipes:

- Pipe 1, when resetting, should go behind Pipe 3.
- Pipe 2, when resetting, should go behind Pipe 1.
- Pipe 3, when resetting, should go behind Pipe 2.

Rather than give each of these objects a reference to the other pipes, we will simply create a **ScrollHandler** object that will keep a reference to all of these objects and determine where each should go. This will make things a lot easier.

We begin by creating a Scrollable class (inside com.mentormate.mmgame.gameobjects) which we will use to define all the shared properties of the Pipe , BonusLogo and the Grass objects.

The shared properties include:

Instance variables:

**position**, **velocity**, **width**, **height**, and the boolean **isScrolledLeft** to determine when the Scrollable object is no longer visible and should be reset.

Methods:

**update** and **reset**, and getter methods for various instance variables above.

Here's the full code (it's very straightforward). There's really nothing we haven't seen before, except for the reset method.

```
package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Vector2;

public class Scrollable {

    // Protected is similar to private, but allows inheritance by subclasses.
    protected Vector2 position;
    protected Vector2 velocity;
    protected int width;
    protected int height;
    protected boolean isScrolledLeft;
```

```

    public Scrollable(float x, float y, int width, int height, float
scrollSpeed) {
        position = new Vector2(x, y);
        velocity = new Vector2(scrollSpeed, 0);
        this.width = width;
        this.height = height;
        isScrolledLeft = false;
    }

    public void update(float delta) {
        position.add(velocity.cpy().scl(delta));

        // If the Scrollable object is no longer visible:
        if (position.x + width < 0) {
            isScrolledLeft = true;
        }
    }

    // Reset: Should Override in subclass for more specific behavior.
    public void reset(float newX) {
        position.x = newX;
        isScrolledLeft = false;
    }

    // Getters for instance variables
    public boolean isScrolledLeft() {
        return isScrolledLeft;
    }

    public float getTailX() {
        return position.x + width;
    }

    public float getX() {
        return position.x;
    }

    public float getY() {
        return position.y;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }
}

```

Now that we have a generic Scrollable class, we can create subclasses using inheritance. Create the following three classes also inside `com.mentormate.mmgame.gameobjects`:

## Pipe

```
package com.mentormate.mmgame.gameobjects;
```



```

import java.util.Random;

public class Pipe extends Scrollable {

    private Random r;

    // When Pipe's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public Pipe(float x, float y, int width, int height, float scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        height = r.nextInt(90) + 15;
    }
}

```

## Grass

```

package com.mentormate.mmgame.gameobjects;

public class Grass extends Scrollable {

    // When Grass's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public Grass(float x, float y, int width, int height, float scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
    }
}

```

## BonusLogo

```

package com.mentormate.mmgame.gameobjects;

import java.util.Random;

public class BonusLogo extends Scrollable {

    private Random r;

    // When BonusLogo's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public BonusLogo(float x, float y, int width, int height, float
scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
    }
}

```

```

    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        position.y = r.nextInt(90) + 15;
    }
}

```

The three above classes (when we say **Pipe p = new Pipe(...)** or **Grass g = new Grass(...)** or **BonusLogo b = new BonusLogo(...)**) will create themselves using the Scrollable constructor (by inheritance), and also add some of their own properties (in the case of Pipe and BonusLogo, a new instance variable and a new method).

### What is @Override and super?

In inheritance, the subclasses have access to all the methods in the superclass. This means that Pipe , BonusLogo and Grass are all able to **reset** even if we don't define the method specifically inside the classes. This is because they are extensions of Scrollable which **does** have the **reset** method.

For instance, we could do something like this:

```

Grass g = new Grass(...);
g.reset(); // Reset method from scrollable is called.

```

However, if we want more specific behavior than the one described in the superclass, we can use: **@Override**, which basically says: use this **reset** method inside the subclass rather than the **reset** method in the superclass. We are doing this inside the Pipe class - we are replacing the original **reset** method from the **superclass** with a more specific one in the **subclass**.

So, when we do this:

```

Pipe p = new Pipe(...);
p.reset(); // Reset method from Pipe called.

```

So what is super?

Even when Overriding, the subclass still has a reference to the superclass's original **reset** method. Calling **super.reset(...)** inside the Overridden **reset** will mean that **both** reset methods are called.

### Why do we need a Grass class?

Grass, above, is really a worthless class right now, because it has no properties of its own. But we will later add some properties for collision detection and thats why it gets its own class.

Now that our Scrollable classes are ready, we can implement the ScrollHandler, which will take care of creating these Grass the BonusLogo and Pipe objects, updating them, and handling reset.

Create the new **ScrollHandler** class inside **com.mentormate.mmgame.gameobjects**.

We will start with simple stuff:

- We need a constructor that receives a y position to know where we need to create our **ground** (where the grass and the bottom pipes will begin).
- We need five instance variables: 2 for two Grass objects 2 for two Pipes and 2 for two BonusLogos objects (for now, we will treat each column of pipes as one Pipe object).
- We need getters for all of these.
- We need an update method.

```

package com.mentormate.mmgame.gameobjects;

public class ScrollHandler { // ScrollHandler will create all six objects that
                                // we need.
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bonusLogo1, bonusLogo2;

    // ScrollHandler will use the constants below to determine
    // how fast we need to scroll and also determine
    // the size of the gap between each pair of pipes.

    // Capital letters are used by convention when naming constants.
    public static final int SCROLL_SPEED = -59;
    public static final int PIPE_GAP = 143;
;

    // Constructor receives a float that tells us where we need to create our
    // Grass and Pipe objects.
    public ScrollHandler(float yPos) {
    }

    public void update(float delta) {
    }

    // The getters for our five instance variables
    public Grass getFrontGrass() {
        return frontGrass;
    }

    public Grass getBackGrass() {
        return backGrass;
    }

    public Pipe getPipe1() {
        return pipe1;
    }

    public Pipe getPipe2() {
        return pipe2;
    }

    public BonusLogo getBonusLogo1() {
        return bonusLogo1;
    }

    public BonusLogo getBonusLogo2() {
        return bonusLogo2;
    }
}

```

Now we just have to work on the constructor and the update method. Inside the constructor, we will initialize our Scrollable objects:

```
frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11, SCROLL_SPEED);
```

```
pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED);
bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 17, 12,
    SCROLL_SPEED);
pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60, SCROLL_SPEED);
bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 17, 12,
    SCROLL_SPEED);
```

The logic here is simple. Remember that our Scrollable objects' constructors want : x, y, width, height and scroll speed. We pass each of these in.

The **backGrass** object should be attached to the tail of the **frontGrass**, so we create it at the **frontGrass**' tail.

The **Pipe** and **BonusLogo** objects follow a similar logic, except that we add the PIPE\_GAP to create a gap of 49 pixels (experimentally determined).

We will now complete our update method, in which we will call the update methods for all six of these objects. In addition, we will use a similar logic to the one in the constructor above to reset our objects. The full code follows:

```
package com.mentormate.mmgame.gameobjects;

public class ScrollHandler { // ScrollHandler will create all six objects that
    // we need.
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bonusLogo1, bonusLogo2;

    // ScrollHandler will use the constants below to determine
    // how fast we need to scroll and also determine
    // the size of the gap between each pair of pipes.

    // Capital letters are used by convention when naming constants.
    public static final int SCROLL_SPEED = -59;
    public static final int PIPE_GAP = 143;
;

    // Constructor receives a float that tells us where we need to create our
    // Grass and Pipe objects.
    public ScrollHandler(float yPos) {
        frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
        backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11,
            SCROLL_SPEED);
        pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED);
        bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 17,
12,
            SCROLL_SPEED);
        pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60,
SCROLL_SPEED);
        bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 17,
12,
            SCROLL_SPEED);
    }
}
```

```

public void update(float delta) {
    // Update our objects
    frontGrass.update(delta);
    backGrass.update(delta);
    pipe1.update(delta);
    bonusLogo1.update(delta);
    pipe2.update(delta);
    bonusLogo2.update(delta);

    // Check if any of the pipes are scrolled left,
    // and reset accordingly the bonusLogo
    if (pipe1.isScrolledLeft()) {
        pipe1.reset(pipe2.getTailX() + PIPE_GAP);
        bonusLogo2.reset(pipe2.getTailX() + PIPE_GAP / 2);
    } else if (pipe2.isScrolledLeft()) {
        pipe2.reset(pipe1.getTailX() + PIPE_GAP);
        bonusLogo1.reset(pipe1.getTailX() + PIPE_GAP / 2);
    }

    // Same with grass
    if (frontGrass.isScrolledLeft()) {
        frontGrass.reset(backGrass.getTailX());
    } else if (backGrass.isScrolledLeft()) {
        backGrass.reset(frontGrass.getTailX());
    }
}

// The getters for our five instance variables
public Grass getFrontGrass() {
    return frontGrass;
}

public Grass getBackGrass() {
    return backGrass;
}

public Pipe getPipe1() {
    return pipe1;
}

public Pipe getPipe2() {
    return pipe2;
}

public BonusLogo getBonusLogo1() {
    return bonusLogo1;
}

public BonusLogo getBonusLogo2() {
    return bonusLogo2;
}
}

```

Our Scrollable objects' classes are now setup! Our next two steps will be as follows:

1. Create the ScrollHandler object inside the GameWorld (this will automatically create its six objects)

2. Render the ScrollHandlers' six objects inside the GameRenderer.

## 1. Creating the ScrollHandler object

Open up GameWorld. We are going to make a quick change.

- Add an instance variable (import accordingly):

```
private ScrollHandler scroller
```

- Initialize the scroller in the constructor (we include the Y coordinate of where the grass should begin, which is 66 pixels below the midPointY):

```
scroller = new ScrollHandler(midPointY + 66);
```

- Call the Scroller update method inside the GameWorld update method :

```
scroller.update(delta);
```

- Create a getter for the Scroller (return the ScrollHandler scroller).

The full code is:

```
package com.mentormate.mmgame.gameworld;

import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.ScrollHandler;

public class GameWorld {
    private Logo logo;
    private ScrollHandler scroller;

    public GameWorld(int midPointY) {
        logo = new Logo(33, midPointY - 5, 17, 12);
        // The grass should start 66 pixels below the midPointY
        scroller = new ScrollHandler(midPointY + 66);
    }

    public void update(float delta) {
        logo.update(delta);
        scroller.update(delta);
    }

    public Logo getLogo() {
        return logo;
    }

    public ScrollHandler getScroller() {
        return scroller;
    }
}
```

## 2. Rendering the ScrollHandler's Objects

We begin by creating seven instance variables:

1 for the ScrollHandler

6 for the 2 pipes + 2 grass + 2 bonusLogo objects.

- Add the following in **BOLD** below the declaration of the Logo object.

```
// Game Objects
private Logo bird;
private ScrollHandler scroller;
private Grass frontGrass, backGrass;
private Pipe pipe1, pipe2;
private BonusLogo bLogo1 , bLogo2;
```

- Import the following:

```
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.BonusLogo;
```

- Now we must initialize them in the initGameObjects method, which should now look like this:

```
private void initGameObjects() {
    logo = myWorld.getLogo();
    scroller = myWorld.getScroller();
    frontGrass = scroller.getFrontGrass();
    backGrass = scroller.getBackGrass();
    pipe1 = scroller.getPipe1();
    pipe2 = scroller.getPipe2();
    bLogo1 = scroller.getBonusLogo1();
    bLogo2 = scroller.getBonusLogo2();
}
```

We have to update our AssetsLoader with the bonusLogos for android and apple:

```
package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture;
    public static TextureRegion bg;
```

```

public static TextureRegion grass;

public static Animation logoAnimation;
public static TextureRegion logo, logoDown, logoUp;

public static TextureRegion barTopUp, barTopDown, bar;

public static TextureRegion androidLogo, appleLogo;

public static void load() {

    texture = new Texture(Gdx.files.internal("data/texture.png"));
    texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

    bg = new TextureRegion(texture, 0, 0, 136, 43);
    bg.flip(false, true);

    grass = new TextureRegion(texture, 0, 43, 143, 11);
    grass.flip(false, true);

    logoDown = new TextureRegion(texture, 136, 0, 17, 12);
    logoDown.flip(false, true);

    logo = new TextureRegion(texture, 153, 0, 17, 12);
    logo.flip(false, true);

    logoUp = new TextureRegion(texture, 170, 0, 17, 12);
    logoUp.flip(false, true);

    TextureRegion[] birds = { logoDown, logo, logoUp };
    logoAnimation = new Animation(0.06f, birds);
    logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

    barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
    // Create by flipping existing skullUp
    barTopDown = new TextureRegion(barTopUp);
    barTopDown.flip(false, true);

    androidLogo = new TextureRegion(texture, 136, 22, 11, 14);
    androidLogo.flip(false, true);

    appleLogo = new TextureRegion(texture, 147, 22, 11, 14);
    appleLogo.flip(false, true);

    bar = new TextureRegion(texture, 136, 16, 22, 3);
    bar.flip(false, true);

}

public static void dispose() {
    // We must dispose of the texture when we are finished.
    texture.dispose();
}

}

```

Next, we simply have to render these objects in our **render** method. As the Pipe and the bonusLogo objects are not complete yet, we are going to just add some placeholder code that will change later. Let's create some helper methods so that we can keep the code neat:



The pixel values (widths/heights/etc) you see have been added after careful measurements. I thought you would prefer seeing hard coded values to deriving them yourself!

```
private void drawGrass() {
    // Draw the grass
    batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
        frontGrass.getWidth(), frontGrass.getHeight());
    batcher.draw(grass, backGrass.getX(), backGrass.getY(),
        backGrass.getWidth(), backGrass.getHeight());
}

private void drawBarTops() {
    // Temporary code! Sorry about the mess :)
    // We will fix this when we finish the Pipe class.

    batcher.draw(barTopUp, pipe1.getX() - 1,
        pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
    batcher.draw(barTopDown, pipe1.getX() - 1,
        pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

    batcher.draw(barTopUp, pipe2.getX() - 1,
        pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
    batcher.draw(barTopDown, pipe2.getX() - 1,
        pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
}

private void drawPipes() {
    // Temporary code! Sorry about the mess :)
    // We will fix this when we finish the Pipe class.
    batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
        pipe1.getHeight());
    batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
        pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

    batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
        pipe2.getHeight());
    batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
        pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
}

private void drawLogos() {
    // Draw the first logo
    batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
        bLogo1.getWidth(), bLogo1.getHeight());

    // Draw the second logo
    batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
        bLogo2.getWidth(), bLogo2.getHeight());
}
```

Now we just have to call these methods in the appropriate places in our **render** method. Here's the full code below:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
    private TextureRegion bg, grass;
    private Animation logoAnimation;
    private TextureRegion logoMid, logoDown, logoUp;
    private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
        bonusLogoApple;

    private Logo bird;
    private ScrollHandler scroller;
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bLogo1, bLogo2;

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);

        // Call helper methods to initialize instance variables
    }

```

```

        initGameObjects();
        initAssets();
    }

    private void initGameObjects() {
        logo = myWorld.getLogo();
        scroller = myWorld.getScroller();
        frontGrass = scroller.getFrontGrass();
        backGrass = scroller.getBackGrass();
        pipe1 = scroller.getPipe1();
        pipe2 = scroller.getPipe2();
        bLogo1 = scroller.getBonusLogo1();
        bLogo2 = scroller.getBonusLogo2();
    }

    private void initAssets() {
        bg = AssetLoader.bg;
        grass = AssetLoader.grass;
        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        logoDown = AssetLoader.logoDown;
        logoUp = AssetLoader.logoUp;
        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
            frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
            backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawPipes() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.
        batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
            pipe1.getHeight());
        batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
            pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));
    }

```

```

        batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
                    pipe2.getHeight());
        batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
                    pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
                    bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
                    bLogo2.getWidth(), bLogo2.getHeight());
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);

        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        shapeRenderer.end();

        batcher.begin();
        batcher.disableBlending();
        batcher.draw(bg, 0, midPointY + 23, 136, 43);

        // 1. Draw Grass
        drawGrass();

        // 2. Draw Pipes
        drawPipes();
        batcher.enableBlending();

        // 3. Draw Bar Tops (requires transparency)
        drawBarTops();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                        logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                        logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());

```

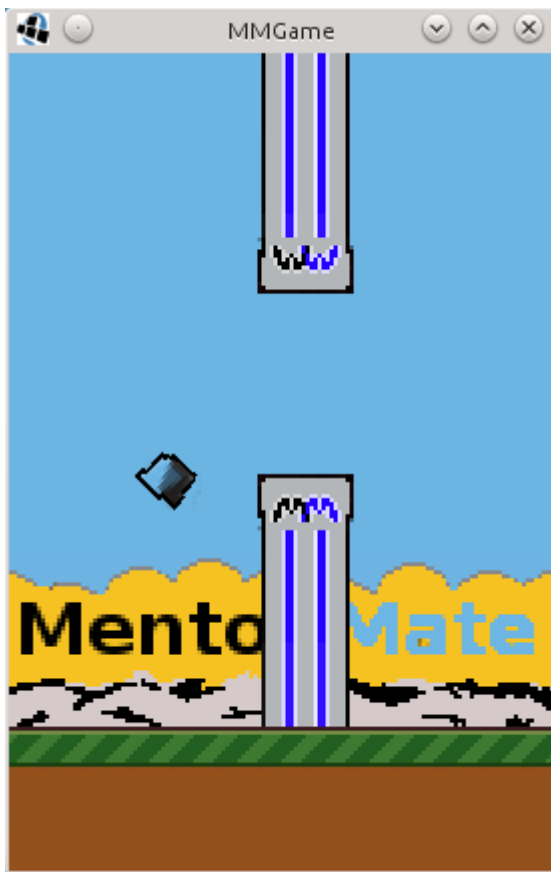
```

    } else {
        batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                    logo.getY(), logo.getWidth() / 2.0f,
                    logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                    1, 1, logo.getRotation());
    }

    batcher.end();
    batcher.begin();
    drawLogos();
    batcher.end();
}
}

```

We've made a lot of progress in Part 7! Let's see what our game looks like now:



## Source Code.

[DOWNLOAD SOURCE](#)

# Part 8 - Collision Detection and Sound Effects

In Flappy Bird, death occurs in one of two ways. You either make contact with the ground or you hit one of the pipes.

We will be implementing the latter type of death in this tutorial (collision with a pipe), and we will be playing a sound effect when this happens!

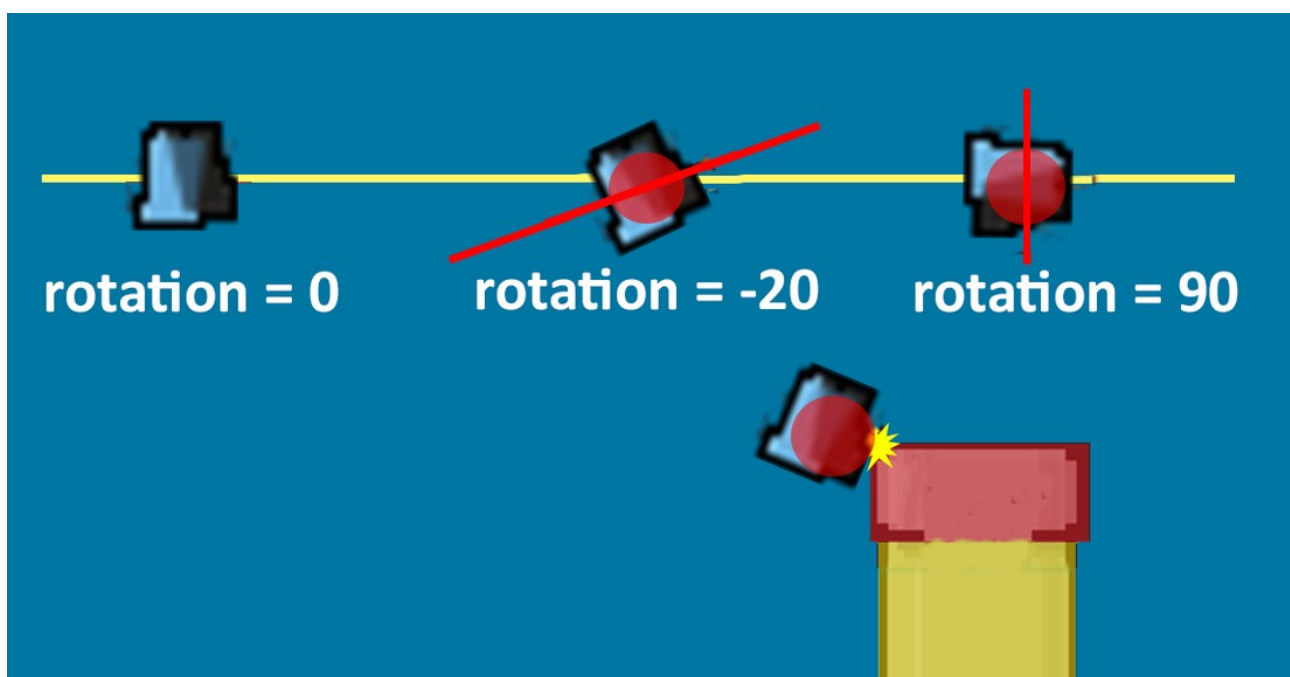
## Discussing Collision

I originally thought about using a rotating polygon to handle collision, but after experimenting, I discovered that a circle is much easier to implement and equally as effective. So, we are going to go with the simpler solution.

The idea is that the circle will always be at centered at the same point. It does not need to rotate. This collision circle will always cover the logo. Because of the way the logo moves, the rear of the logo is extremely unlikely to make contact with the pipes, so we will not prioritize checking that area.

The Pipe object, in our game, comprises both the upper and lower pipes. Each of these pipes will be treated using two rectangles. One rectangle will cover the upper part, the other rectangle will cover the base.

When checking for collision, we simply have to use the built-in Intersector class, which has a method for checking collision between rectangles and circles. Once this collision is detected, we will inform the our game that this happened, and we will tell all of our scrolling objects to stop.



# Logo Class - Bounding Circle

We will begin by quickly creating the bounding **Circle** for our Bird. Open up the **Bird** class.

- Create an instance variable of Circle (import com.badlogic.gdx.math.Circle;) called boundingCircle.  
`private Circle boundingCircle;`

- Initialize it in the constructor with the no values:  
`boundingCircle = new Circle();`

We must set the circle's coordinates each time that the Logo moves. The Logo moves when we add the scaled velocity to the position, so add the following in bold directly below this line ->  
`position.add(velocity.cpy().scl(delta)).`

`boundingCircle.set(position.x + 9, position.y + 6, 6.5f);`

- Add a **getter method** for the **boundingCircle**.

Here is what your Logo class should look like:

```
package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Vector2;

public class Logo {

    private Vector2 position;
    private Vector2 velocity;
    private Vector2 acceleration;

    private float rotation; // For handling bird rotation
    private int width;
    private int height;

    private Circle boundingCircle;

    public Logo(float x, float y, int width, int height) {
        this.width = width;
        this.height = height;
        position = new Vector2(x, y);
        velocity = new Vector2(0, 0);
        acceleration = new Vector2(0, 460);
        boundingCircle = new Circle();
    }

    public void update(float delta) {

        velocity.add(acceleration.cpy().scl(delta));

        if (velocity.y > 200) {
            velocity.y = 200;
        }

        position.add(velocity.cpy().scl(delta));

        // Set the circle's center to be (9, 6) with respect to the logo.
        // Set the circle's radius to be 6.5f;
        boundingCircle.set(position.x + 9, position.y + 6, 6.5f);
    }
}
```

```

        // Rotate counterclockwise
        if (velocity.y < 0) {
            rotation -= 600 * delta;

            if (rotation < -20) {
                rotation = -20;
            }
        }

        // Rotate clockwise
        if (isFalling()) {
            rotation += 480 * delta;
            if (rotation > 90) {
                rotation = 90;
            }
        }
    }

    public boolean isFalling() {
        return velocity.y > 110;
    }

    public boolean shouldntFlap() {
        return velocity.y > 70;
    }

    public void onClick() {
        velocity.y = -140;
    }

    public float getX() {
        return position.x;
    }

    public float getY() {
        return position.y;
    }

    public float getWidth() {
        return width;
    }

    public float getHeight() {
        return height;
    }

    public float getRotation() {
        return rotation;
    }

    public Circle getBouncingCircle() {
        return boundingCircle;
    }
}

```



Next, we are going to make sure that our Circle is positioned correctly. Open up the **GameRenderer** and add these four lines of code at the very bottom of the **render** method. We are going to draw the **boundingCircle**.

```
shapeRenderer.begin(ShapeType.Filled);
shapeRenderer.setColor(Color.RED);
shapeRenderer.circle(logo.getBoundingBox().x, logo.getBoundingBox().y,
logo.getBoundingBox().radius);
shapeRenderer.end();
```

With that, our game will now look like this:



## Pipe Class - Bounding Rectangles

Now that we have the Bird's bounding circle, we must create the four Rectangle objects to represent each of our Pipe objects as shown below. Open the Pipe class.

- Create the following instance variables (import com.badlogic.gdx.math.Rectangle).

```
private Rectangle barTopUp, barTopDown, barUp, barDown
```

- We need a constant to represent the 45 pixel vertical gap between the upper and lower pipe, and some other constants:

```
public static final int VERTICAL_GAP = 45;
public static final int PIPE_TOP_WIDTH = 24;
public static final int PIPE_TOP_HEIGHT = 11;
```

- We also need to know where the ground is, so that we can provide a height for our Lower Bar (as shown in the diagram above). Create one more instance variable:

**private float groundY;**

- Make the below changes **IN BOLD** to the constructor so that we can initialize **barTopUp**, **barTopDown**, **barUp**, **barDown** and **groundY**:

```
public Pipe(float x, float y, int width, int height, float scrollSpeed, float groundY) {  
  
    super(x, y, width, height, scrollSpeed);  
    // Initialize a Random object for Random number generation  
    r = new Random();  
    barTopUp = new Rectangle();  
    barTopDown = new Rectangle();  
    barUp = new Rectangle();  
    barDown = new Rectangle();  
    this.groundY = groundY;  
  
}
```

- Now add the **getter methods** for the four rectangle objects, so that we can access them when we check for collision:

```
public Rectangle getBarTopUp() {  
    return barTopUp;  
}  
  
public Rectangle getBarTopDown() {  
    return barTopDown;  
}  
  
public Rectangle getBarUp() {  
    return barUp;  
}  
  
public Rectangle getBarDown() {  
    return barDown;  
}  
  
public float getGroundY() {  
    return groundY;  
}
```

As with the Logo's **boundingCircle**, we must update each of the four rectangles whenever our Pipe object's position is updated. At the moment, our **Pipe** does not have an update method. It is inheriting the **update** method from the **Scrollable** class. We could try to update our four rectangles in our **Scrollable** class, but the easier way to do this is to **@Override** the **update** method (review what this means in Part 7 if you have forgotten).

By calling **super**, we are calling the original **update** method that belongs to the **Scrollable** class. Any code that follows the **super** call is additional functionality. In this case, we simply update our four rectangles.

Here is how we would update our upper bar and the lower bar rectangles (this is what your **Pipe** class should now look like):

```
package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Rectangle;

public class Pipe extends Scrollable {

    private Random r;

    private Rectangle barTopUp, barTopDown, barUp, barDown;

    public static final int VERTICAL_GAP = 45;
    public static final int PIPE_TOP_WIDTH = 24;
    public static final int PIPE_TOP_HEIGHT = 11;
    private float groundY;

    // When Pipe's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public Pipe(float x, float y, int width, int height, float scrollSpeed,
               float groundY) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
        barTopUp = new Rectangle();
        barTopDown = new Rectangle();
        barUp = new Rectangle();
        barDown = new Rectangle();
        this.groundY = groundY;
    }

    @Override
    public void update(float delta) {
        // Call the update method in the superclass (Scrollable)
        super.update(delta);

        // The set() method allows you to set the top left corner's x, y
        // coordinates,
        // along with the width and height of the rectangle

        barUp.set(position.x, position.y, width, height);
        barDown.set(position.x, position.y + height + VERTICAL_GAP, width,
                    groundY - (position.y + height + VERTICAL_GAP));

        // Our barTop width is 24. The bar is only 22 pixels wide. So the
        // must be shifted by 1 pixel to the left (so that the MM is
        // centered
        // with respect to its bar).

        // This shift is equivalent to: (PIPE_TOP_WIDTH - width) / 2
        barTopUp.set(position.x - (PIPE_TOP_WIDTH - width) / 2, position.y
                    + height - PIPE_TOP_HEIGHT, PIPE_TOP_WIDTH,
PIPE_TOP_HEIGHT);
        barTopDown.set(position.x - (PIPE_TOP_WIDTH - width) / 2, barDown.y,
                    PIPE_TOP_WIDTH, PIPE_TOP_HEIGHT);
    }
}
```

```

    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        height = r.nextInt(90) + 15;
    }

    public Rectangle getBarTopUp() {
        return barTopUp;
    }

    public Rectangle getBarTopDown() {
        return barTopDown;
    }

    public Rectangle getBarUp() {
        return barUp;
    }

    public Rectangle getBarDown() {
        return barDown;
    }

    public float getGroundY() {
        return groundY;
    }
}

```

## Updating the ScrollHandler

We have changed the constructor for our Pipe objects, so we must include the additional argument whenever we create a Pipe object in ScrollHandler. Change the following lines:

```

pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED);
pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60, SCROLL_SPEED);

```

To this:

```

pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED, yPos);
pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60, SCROLL_SPEED, yPos);

```

Now let's go back to the GameRenderer and add some lines of code at the end of the Render method.

This is just temporary code for testing. Just copy and paste as shown below.

```

package com.mentormate.mmgame.gameworld;

```

```

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
    private TextureRegion bg, grass;
    private Animation logoAnimation;
    private TextureRegion logoMid, logoDown, logoUp;
    private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
        bonusLogoApple;

    private Logo bird;
    private ScrollHandler scroller;
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bLogo1, bLogo2;

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);

        // Call helper methods to initialize instance variables

```

```

        initGameObjects();
        initAssets();
    }

    private void initGameObjects() {
        logo = myWorld.getLogo();
        scroller = myWorld.getScroller();
        frontGrass = scroller.getFrontGrass();
        backGrass = scroller.getBackGrass();
        pipe1 = scroller.getPipe1();
        pipe2 = scroller.getPipe2();
        bLogo1 = scroller.getBonusLogo1();
        bLogo2 = scroller.getBonusLogo2();
    }

    private void initAssets() {
        bg = AssetLoader.bg;
        grass = AssetLoader.grass;
        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        logoDown = AssetLoader.logoDown;
        logoUp = AssetLoader.logoUp;
        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
            frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
            backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawPipes() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.
        batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
            pipe1.getHeight());
        batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
            pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));
    }

```

```

        batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
                    pipe2.getHeight());
        batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
                    pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
                    bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
                    bLogo2.getWidth(), bLogo2.getHeight());
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        shapeRenderer.end();

        batcher.begin();
        batcher.disableBlending();
        batcher.draw(bg, 0, midPointY + 23, 136, 43);

        // 1. Draw Grass
        drawGrass();

        // 2. Draw Pipes
        drawPipes();
        batcher.enableBlending();

        // 3. Draw Bar Tops (requires transparency)
        drawBarTops();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                        logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                        logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
        } else {

```

```

        batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                    logo.getY(), logo.getWidth() / 2.0f,
                    logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                    1, 1, logo.getRotation());
    }

    batcher.end();
    batcher.begin();
    drawLogos();
    batcher.end();

    shapeRenderer.begin(ShapeType.Filled);
    shapeRenderer.setColor(Color.RED);
    shapeRenderer.circle(logo.getBoundingCircle().x,
                        logo.getBoundingCircle().y,
logo.getBoundingCircle().radius);

    /*
     * Excuse the mess below. Temporary code for testing bounding
     * rectangles.
     */
    // Bar up for pipes 1 and 2
    shapeRenderer.rect(pipe1.getBarUp().x, pipe1.getBarUp().y,
                    pipe1.getBarUp().width, pipe1.getBarUp().height);
    shapeRenderer.rect(pipe2.getBarUp().x, pipe2.getBarUp().y,
                    pipe2.getBarUp().width, pipe2.getBarUp().height);

    // Bar down for pipes 1 and 2
    shapeRenderer.rect(pipe1.getBarDown().x, pipe1.getBarDown().y,
                    pipe1.getBarDown().width, pipe1.getBarDown().height);
    shapeRenderer.rect(pipe2.getBarDown().x, pipe2.getBarDown().y,
                    pipe2.getBarDown().width, pipe2.getBarDown().height);

    // BarTop up for Pipes 1 and 2
    shapeRenderer.rect(pipe1.getBarTopUp().x, pipe1.getBarTopUp().y,
                    pipe1.getBarTopUp().width, pipe1.getBarTopUp().height);
    shapeRenderer.rect(pipe2.getBarTopUp().x, pipe2.getBarTopUp().y,
                    pipe2.getBarTopUp().width, pipe2.getBarTopUp().height);

    // BarTop down for Pipes 1 and 2
    shapeRenderer.rect(pipe1.getBarTopDown().x, pipe1.getBarTopDown().y,
                    pipe1.getBarTopDown().width,
pipe1.getBarTopDown().height);
    shapeRenderer.rect(pipe2.getBarTopDown().x, pipe2.getBarTopDown().y,
                    pipe2.getBarTopDown().width,
pipe2.getBarTopDown().height);

    shapeRenderer.end();
}
}

```

Next, we run our game! This is what it should look like:





As you can see the only thing that left is the boundary circle for the logo

We will start with the BonusLogo class and add the boundryCircle implementation;

```
package com.mentormate.mmgame.gameobjects;

import java.util.Random;
import com.badlogic.gdx.math.Circle;

public class BonusLogo extends Scrollable {

    private Random r;
    private Circle boundingCircle;

    // When BonusLogo's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public BonusLogo(float x, float y, int width, int height, float
scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
        boundingCircle = new Circle();
    }
}
```

```

@Override
public void update(float delta) {
    // Call the update method in the superclass (Scrollable)
    super.update(delta);
    boundingCircle.set(position.x + 9, position.y + 6, 6.5f);
}

@Override
public void reset(float newX) {
    // Call the reset method in the superclass (Scrollable)
    super.reset(newX);
    // Change the height to a random number
    position.y = r.nextInt(90) + 15;
}

public Circle getBouncingCircle() {
    return boundingCircle;
}
}

```

And next let's paint those images in red in our GameRenderer class

```

public void render(float runTime) {

    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    shapeRenderer.begin(ShapeType.Filled);

    // Draw Background color
    shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
    shapeRenderer.rect(0, 0, 136, midPointY + 66);

    // Draw Grass
    shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 66, 136, 11);

    // Draw Dirt
    shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 77, 136, 52);

    shapeRenderer.end();

    batcher.begin();
    batcher.disableBlending();
    batcher.draw(bg, 0, midPointY + 23, 136, 43);

    // 1. Draw Grass
    drawGrass();

    // 2. Draw Pipes
    drawPipes();
    batcher.enableBlending();

    // 3. Draw Bar Tops (requires transparency)
}

```

```

        drawBarTops();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());

        } else {
            batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                logo.getY(), logo.getWidth() / 2.0f,
                logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                1, 1, logo.getRotation());

        }

        batcher.end();
        batcher.begin();
        drawLogos();
        batcher.end();

        shapeRenderer.begin(ShapeType.Filled);
        shapeRenderer.setColor(Color.RED);
        shapeRenderer.circle(logo.getBoundingBox().x,
            logo.getBoundingBox().y,
logo.getBoundingBox().radius);

        /*
        * Excuse the mess below. Temporary code for testing bounding
        * rectangles.
        */
        // Bar up for pipes 1 and 2
        shapeRenderer.rect(pipe1.getBarUp().x, pipe1.getBarUp().y,
            pipe1.getBarUp().width, pipe1.getBarUp().height);
        shapeRenderer.rect(pipe2.getBarUp().x, pipe2.getBarUp().y,
            pipe2.getBarUp().width, pipe2.getBarUp().height);

        // Bar down for pipes 1 and 2
        shapeRenderer.rect(pipe1.getBarDown().x, pipe1.getBarDown().y,
            pipe1.getBarDown().width, pipe1.getBarDown().height);
        shapeRenderer.rect(pipe2.getBarDown().x, pipe2.getBarDown().y,
            pipe2.getBarDown().width, pipe2.getBarDown().height);

        // BarTop up for Pipes 1 and 2
        shapeRenderer.rect(pipe1.getBarTopUp().x, pipe1.getBarTopUp().y,
            pipe1.getBarTopUp().width, pipe1.getBarTopUp().height);
        shapeRenderer.rect(pipe2.getBarTopUp().x, pipe2.getBarTopUp().y,
            pipe2.getBarTopUp().width, pipe2.getBarTopUp().height);

        // BarTop down for Pipes 1 and 2
        shapeRenderer.rect(pipe1.getBarTopDown().x, pipe1.getBarTopDown().y,
            pipe1.getBarTopDown().width,
pipe1.getBarTopDown().height);
        shapeRenderer.rect(pipe2.getBarTopDown().x, pipe2.getBarTopDown().y,
            pipe2.getBarTopDown().width,
pipe2.getBarTopDown().height);

        //Draw the circle for the bonus Logo
        shapeRenderer
            .circle(bLogo1.getBoundingBox().x,
                bLogo1.getBoundingBox().y,
                bLogo1.getBoundingBox().radius);
    
```

```

        shapeRenderer
            .circle(bLogo2.getBoundingCircle().x,
                    bLogo2.getBoundingCircle().y,
                    bLogo2.getBoundingCircle().radius);

        shapeRenderer.end();
    }

```

And finally we should see:



We have the necessary bounding boxes to test for collision. Now we simply have to add the logic.

## Checking for Collision

Checking for collision requires multiple classes working together.

- The ScrollHandler has access to all the Pipes and their bounding rectangles, so it should be the one doing the actual collision checking.
- The GameWorld needs to know when the collision occurs, so that it can handle the collision (get current score, make the logo stop, play sound, etc).

- The GameRenderer, once the Logo dies, needs to be able to react (show the score, display a flash).

We will start in the GameWorld class. We will add the following in bold to our update method.

```
public void update(float delta) {  
    logo.update(delta);  
    scroller.update(delta);  
  
    if (scroller.collides(logo)) {  
        // Clean up on game over  
        scroller.stop();  
    }  
  
}
```

We now need to create two methods in our ScrollHandler: the stop and the collides. Let's go to the ScrollHandlerclass and add the following methods:

```
public void stop() {  
    frontGrass.stop();  
    backGrass.stop();  
    pipe1.stop();  
    pipe2.stop();  
    bonusLogo1.stop();  
    bonusLogo2.stop();  
}  
  
// Return true if ANY pipe hits the logo.  
public boolean collides(Logo logo) {  
    return (pipe1.collides(logo) || pipe2.collides(logo) );  
}
```

This is the full class:

```

package com.mentormate.mmgame.gameobjects;

public class ScrollHandler { // ScrollHandler will create all six objects that
                                // we need.
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bonusLogo1, bonusLogo2;

    // ScrollHandler will use the constants below to determine
    // how fast we need to scroll and also determine
    // the size of the gap between each pair of pipes.

    // Capital letters are used by convention when naming constants.
    public static final int SCROLL_SPEED = -59;
    public static final int PIPE_GAP = 143;

    // Constructor receives a float that tells us where we need to create our
    // Grass and Pipe objects.
    public ScrollHandler(float yPos) {

        frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
        backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11,
                               SCROLL_SPEED);
        pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED, yPos);
        bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 16,
11,
                               SCROLL_SPEED);
        pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60,
SCROLL_SPEED,
                               yPos);
        bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 16,
11,
                               SCROLL_SPEED);
    }

    public void update(float delta) {
        // Update our objects
        frontGrass.update(delta);
        backGrass.update(delta);
        pipe1.update(delta);
        bonusLogo1.update(delta);
        pipe2.update(delta);
        bonusLogo2.update(delta);

        // Check if any of the pipes are scrolled left,
        // and reset accordingly the bonusLogo
        if (pipe1.isScrolledLeft()) {
            pipe1.reset(pipe2.getTailX() + PIPE_GAP);
            bonusLogo2.reset(pipe2.getTailX() + PIPE_GAP / 2);
        } else if (pipe2.isScrolledLeft()) {
            pipe2.reset(pipe1.getTailX() + PIPE_GAP);
            bonusLogo1.reset(pipe1.getTailX() + PIPE_GAP / 2);
        }

        // Same with grass
        if (frontGrass.isScrolledLeft()) {
            frontGrass.reset(backGrass.getTailX());
        } else if (backGrass.isScrolledLeft()) {
            backGrass.reset(frontGrass.getTailX());
        }
    }
}

```

```

    }

}

public void stop() {
    frontGrass.stop();
    backGrass.stop();
    pipe1.stop();
    pipe2.stop();
    bonusLogo1.stop();
    bonusLogo2.stop();
}

// Return true if ANY pipe hits the logo.
public boolean collides(Logo logo) {
    return (pipe1.collides(logo) || pipe2.collides(logo));
}

// The getters for our five instance variables
public Grass getFrontGrass() {
    return frontGrass;
}

public Grass getBackGrass() {
    return backGrass;
}

public Pipe getPipe1() {
    return pipe1;
}

public Pipe getPipe2() {
    return pipe2;
}

public BonusLogo getBonusLogo1() {
    return bonusLogo1;
}

public BonusLogo getBonusLogo2() {
    return bonusLogo2;
}
}

```

Now, to fix errors, we need to make two changes. Our **Scrollable** objects (Pipe BonusLogo and Grass) need to be able to **stop**, so we will create a **stop** method inside the **Scrollable Class**.

- Open up Scrollable and add the following method:

```

public void stop() {
    velocity.x = 0;
}

```

- Next, our Pipe objects need to individually check if they collided with the Logo, so we need to add that method. Open up the Pipe class, and add the following method:

```

public boolean collides(Logo logo) {
    if (position.x < logo.getX() + logo.getWidth()) {
        return (Intersector.overlaps(logo.getBoundingCircle(), barUp)
            || Intersector.overlaps(logo.getBoundingCircle(), barDown)
            || Intersector.overlaps(logo.getBoundingCircle(), barTopUp)
            || Intersector.overlaps(logo.getBoundingCircle(), barTopDown));
    }
    return false;
}

```

In this method, we begin by checking if the `position.x` is less than `logo.getX() + logo.getWidth()`, because otherwise, collision is impossible. This is a very cheap check (it does not take much toll on performance). Most of the time, this condition will fail, and we will not have to perform the more expensive checks.

If this **if statement** *does evaluate* to **true**, we do the much more expensive **Intersector.overlaps()** calls (which returns true if the circle argument object collides with a rectangle argument object). We will return **true** if any of the four rectangle collides with the bird's circle.

We have to make the same for the `bonusLogo`;

```

public boolean collides(Logo logo) {
    if (position.x < logo.getX() + logo.getWidth()) {
        return (Intersector.overlaps(logo.getBoundingCircle(),
            boundingCircle));
    }
    return false;
}

```

Here are the updated **Pipe BonusLog** and **Scrollable** classes:

## Updated Pipe Class

```

package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.math.Rectangle;

public class Pipe extends Scrollable {

    private Random r;

    private Rectangle barTopUp, barTopDown, barUp, barDown;
}

```



```

public static final int VERTICAL_GAP = 45;
public static final int PIPE_TOP_WIDTH = 24;
public static final int PIPE_TOP_HEIGHT = 11;
private float groundY;

// When Pipe's constructor is invoked, invoke the super (Scrollable)
// constructor
public Pipe(float x, float y, int width, int height, float scrollSpeed,
           float groundY) {
    super(x, y, width, height, scrollSpeed);
    // Initialize a Random object for Random number generation
    r = new Random();
    barTopUp = new Rectangle();
    barTopDown = new Rectangle();
    barUp = new Rectangle();
    barDown = new Rectangle();
    this.groundY = groundY;
}

@Override
public void update(float delta) {
    // Call the update method in the superclass (Scrollable)
    super.update(delta);

    // The set() method allows you to set the top left corner's x, y
    // coordinates,
    // along with the width and height of the rectangle

    barUp.set(position.x, position.y, width, height);
    barDown.set(position.x, position.y + height + VERTICAL_GAP, width,
                groundY - (position.y + height + VERTICAL_GAP));

    // Our barTop width is 24. The bar is only 22 pixels wide. So the
    // must be shifted by 1 pixel to the left (so that the MM is
    // centered
    // with respect to its bar).

    // This shift is equivalent to: (PIPE_TOP_WIDTH - width) / 2
    barTopUp.set(position.x - (PIPE_TOP_WIDTH - width) / 2, position.y
                + height - PIPE_TOP_HEIGHT, PIPE_TOP_WIDTH,
PIPE_TOP_HEIGHT);
    barTopDown.set(position.x - (PIPE_TOP_WIDTH - width) / 2, barDown.y,
                PIPE_TOP_WIDTH, PIPE_TOP_HEIGHT);
}

@Override
public void reset(float newX) {
    // Call the reset method in the superclass (Scrollable)
    super.reset(newX);
    // Change the height to a random number
    height = r.nextInt(90) + 15;
}

public boolean collides(Logo logo) {
    if (position.x < logo.getX() + logo.getWidth()) {
        return (Intersector.overlaps(logo.getBouncingCircle(), barUp)
                || Intersector.overlaps(logo.getBouncingCircle(),
barDown)
                || Intersector.overlaps(logo.getBouncingCircle(),

```

```

barTopUp) || Intersector
                                .overlaps(logo.getBoundingCircle(),
barTopDown));
    }
    return false;
}

public Rectangle getBarTopUp() {
    return barTopUp;
}

public Rectangle getBarTopDown() {
    return barTopDown;
}

public Rectangle getBarUp() {
    return barUp;
}

public Rectangle getBarDown() {
    return barDown;
}

public float getGroundY() {
    return groundY;
}

```

## Updated Scrollable class

```

package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Vector2;

public class Scrollable {

    // Protected is similar to private, but allows inheritance by subclasses.
    protected Vector2 position;
    protected Vector2 velocity;
    protected int width;
    protected int height;
    protected boolean isScrolledLeft;

    public Scrollable(float x, float y, int width, int height, float
scrollSpeed) {
        position = new Vector2(x, y);
        velocity = new Vector2(scrollSpeed, 0);
        this.width = width;
        this.height = height;
        isScrolledLeft = false;
    }

    public void update(float delta) {
        position.add(velocity.cpy().scl(delta));

        // If the Scrollable object is no longer visible:
        if (position.x + width < 0) {

```

```

        isScrolledLeft = true;
    }
}

// Reset: Should Override in subclass for more specific behavior.
public void reset(float newX) {
    position.x = newX;
    isScrolledLeft = false;
}

public void stop() {
    velocity.x = 0;
}

// Getters for instance variables
public boolean isScrolledLeft() {
    return isScrolledLeft;
}

public float getTailX() {
    return position.x + width;
}

public float getX() {
    return position.x;
}

public float getY() {
    return position.y;
}

public int getWidth() {
    return width;
}

public int getHeight() {
    return height;
}
}

```

## Updated BonusLogo class

```

package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Intersector;

public class BonusLogo extends Scrollable {

    private Random r;
    private Circle boundingCircle;
}

```

```

// When BonusLogo's constructor is invoked, invoke the super (Scrollable)
// constructor
public BonusLogo(float x, float y, int width, int height, float
scrollSpeed) {
    super(x, y, width, height, scrollSpeed);
    // Initialize a Random object for Random number generation
    r = new Random();
    boundingCircle = new Circle();
}

@Override
public void update(float delta) {
    // Call the update method in the superclass (Scrollable)
    super.update(delta);
    boundingCircle.set(position.x + 9, position.y + 6, 6.5f);
}

public boolean collides(Logo logo) {
    if (position.x < logo.getX() + logo.getWidth()) {
        return (Intersector.overlaps(logo.getBoundingCircle(),
            boundingCircle));
    }
    return false;
}

@Override
public void reset(float newX) {
    // Call the reset method in the superclass (Scrollable)
    super.reset(newX);
    // Change the height to a random number
    position.y = r.nextInt(90) + 15;
}

public Circle getBoundingCircle() {
    return boundingCircle;
}
}

```

## Testing our code!

Now that we have completed all the collision checking code, run your game to make sure that it's working! One the Logo collides with one of the pipes, all the scrolling should stop. We will do some more sophisticated death handling in Day 9. For now, let's try playing a sound file when the bird dies.

## Download the Sound File

### [DOWNLOAD LINK](#)

Download this file into your **MMGGame -android** project's **assets/data** folder.

# Updating the AssetLoader

Now that we have a sound file, we must create a Sound object using this file in our AssetLoader. Sound objects are stored in memory and can be loaded once and used quickly (this is appropriate for short sound files, which have a very small file size).

- Create this instance variable in AssetLoader:

```
public static Sound dead;
```

- Initialize it as below in the load method:

```
dead = Gdx.audio.newSound(Gdx.files.internal("data/dead.wav"));
```

Here's the full class:

```
package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture;
    public static TextureRegion bg;
    public static TextureRegion grass;

    public static Animation logoAnimation;
    public static TextureRegion logo, logoDown, logoUp;

    public static TextureRegion barTopUp, barTopDown, bar;

    public static TextureRegion androidLogo, appleLogo;

    public static Sound dead;

    public static void load() {

        texture = new Texture(Gdx.files.internal("data/texture.png"));
        texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

        bg = new TextureRegion(texture, 0, 0, 136, 43);
        bg.flip(false, true);

        grass = new TextureRegion(texture, 0, 43, 143, 11);
        grass.flip(false, true);

        logoDown = new TextureRegion(texture, 136, 0, 17, 12);
        logoDown.flip(false, true);

        logo = new TextureRegion(texture, 153, 0, 17, 12);
        logo.flip(false, true);

        logoUp = new TextureRegion(texture, 170, 0, 17, 12);
```

```

        logoUp.flip(false, true);

        TextureRegion[] birds = { logoDown, logo, logoUp };
        logoAnimation = new Animation(0.06f, birds);
        logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

        barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
        // Create by flipping existing skullUp
        barTopDown = new TextureRegion(barTopUp);
        barTopDown.flip(false, true);

        androidLogo = new TextureRegion(texture, 136, 22, 11, 14);
        androidLogo.flip(false, true);

        appleLogo = new TextureRegion(texture, 147, 22, 11, 14);
        appleLogo.flip(false, true);

        bar = new TextureRegion(texture, 136, 16, 22, 3);
        bar.flip(false, true);

        dead = Gdx.audio.newSound(Gdx.files.internal("data/dead.wav"));
    }

    public static void dispose() {
        // We must dispose of the texture when we are finished.
        texture.dispose();
    }
}

```

## Playing the Sound File

Now that we have our Sound object, we can play it in our game. Open up the GameWorld, and create the following instance variable:

```
private boolean isAlive = true;
```

Then make the following changes to the update method:

```

public void update(float delta) {

    logo.update(delta);
    scroller.update(delta);

    if (isAlive && scroller.collides(logo)) {
        scroller.stop();
        AssetLoader.dead.play();
        isAlive = false;
    }
}

```

```
}
```

Now, when your Logo dies, it will play a sound file (just once)! Here's the updated GameWorld. Try running the game!

```
package com.mentormate.mmgame.gameworld;

import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameWorld {
    private Logo logo;
    private ScrollHandler scroller;
    private boolean isAlive = true;

    public GameWorld(int midPointY) {
        logo = new Logo(33, midPointY - 5, 17, 12);
        // The grass should start 66 pixels below the midPointY
        scroller = new ScrollHandler(midPointY + 66);
    }

    public void update(float delta) {
        logo.update(delta);
        scroller.update(delta);

        if (scroller.collides(logo)) {
            // Clean up on game over
            scroller.stop();
        }

        if (isAlive && scroller.collides(logo)) {
            scroller.stop();
            AssetLoader.dead.play();
            isAlive = false;
        }
    }

    public Logo getLogo() {
        return logo;
    }

    public ScrollHandler getScroller() {
        return scroller;
    }
}
```

Finally, you can remove all the **red rectangles and circles** from the GameRenderer, now that we know collision is working properly. Here's what your **GameRenderer** should look like:

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
```

```

import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.zbhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
    private TextureRegion bg, grass;
    private Animation logoAnimation;
    private TextureRegion logoMid, logoDown, logoUp;
    private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
        bonusLogoApple;

    private Logo bird;
    private ScrollHandler scroller;
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bLogo1, bLogo2;

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);

        // Call helper methods to initialize instance variables
        initGameObjects();
        initAssets();
    }

```



```

private void initGameObjects() {
    logo = myWorld.getLogo();
    scroller = myWorld.getScroller();
    frontGrass = scroller.getFrontGrass();
    backGrass = scroller.getBackGrass();
    pipe1 = scroller.getPipe1();
    pipe2 = scroller.getPipe2();
    bLogo1 = scroller.getBonusLogo1();
    bLogo2 = scroller.getBonusLogo2();
}

private void initAssets() {
    bg = AssetLoader.bg;
    grass = AssetLoader.grass;
    logoAnimation = AssetLoader.logoAnimation;
    logoMid = AssetLoader.logo;
    logoDown = AssetLoader.logoDown;
    logoUp = AssetLoader.logoUp;
    barTopUp = AssetLoader.barTopUp;
    barTopDown = AssetLoader.barTopDown;
    bar = AssetLoader.bar;
    bonusLogoAndroid = AssetLoader.androidLogo;
    bonusLogoApple = AssetLoader.appleLogo;
}

private void drawGrass() {
    // Draw the grass
    batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
        frontGrass.getWidth(), frontGrass.getHeight());
    batcher.draw(grass, backGrass.getX(), backGrass.getY(),
        backGrass.getWidth(), backGrass.getHeight());
}

private void drawBarTops() {
    // Temporary code! Sorry about the mess :)
    // We will fix this when we finish the Pipe class.

    batcher.draw(barTopUp, pipe1.getX() - 1,
        pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
    batcher.draw(barTopDown, pipe1.getX() - 1,
        pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

    batcher.draw(barTopUp, pipe2.getX() - 1,
        pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
    batcher.draw(barTopDown, pipe2.getX() - 1,
        pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
}

private void drawPipes() {
    // Temporary code! Sorry about the mess :)
    // We will fix this when we finish the Pipe class.
    batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
        pipe1.getHeight());
    batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
        pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

    batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
        pipe2.getHeight());
    batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +

```

```

45,
        pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
            bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
            bLogo2.getWidth(), bLogo2.getHeight());
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        shapeRenderer.end();

        batcher.begin();
        batcher.disableBlending();
        batcher.draw(bg, 0, midPointY + 23, 136, 43);

        // 1. Draw Grass
        drawGrass();

        // 2. Draw Pipes
        drawPipes();
        batcher.enableBlending();

        // 3. Draw Bar Tops (requires transparency)
        drawBarTops();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
        } else {
            batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                logo.getY(), logo.getWidth() / 2.0f,
                logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),

```

```

        1, 1, logo.getRotation());
    }

    batcher.end();
    batcher.begin();
    drawLogos();
    batcher.end();
}
}

```

## Source Code

[DOWNLOAD SOURCE](#)

# Part 9 - Finishing Gameplay and Basic UI

In this part, we are going to finish up the gameplay by adding collision to the ground and handling death. Then, we are going to implement the score keeping and setup a BitmapFont to display the score!

## A Couple More Sound Effects

We need to add a sound effect for the flapping of wings and another sound effect for scoring a point.

- Create two new Sound instance variables called flap and coin, and initialize them inside the load method using:

```

flap = Gdx.audio.newSound(Gdx.files.internal("data/flap.wav"));
coin = Gdx.audio.newSound(Gdx.files.internal("data/coin.wav"));

```

When we are done with our Sound objects, we must dispose of them. So down in the dispose method, add the following code:

```

dead.dispose();
flap.dispose();
coin.dispose();

```

Download the following files into the assets/data folder in your ZombieGame-android project:

[DOWNLOAD coin.wav](#)

[DOWNLOAD flap.wav](#)

The Best Practice will be to download all the files from [HERE](#) and add them in the assets/data/ folder

Just to be proactive, we are going to add a .font file generated using **Hiero**. Hiero converts a text file into a .png Texture image similar to the one for our game. It also generates a .fnt configuration file which **libGDX** can read and figure out where each letter (or TextureRegion) exists.

I have generated these files for you. I will show you how to implement them into our game.

I will show you the full AssetsLoader implementation with all the fields that we are going to use in the future just don't forget to download all the assets :

```
package com.mentormate.mmgame.zbhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Preferences;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture, logoTexture;
    public static TextureRegion bg;
    public static TextureRegion grass;

    public static Animation logoAnimation;

    public static TextureRegion logo, logoDown, logoUp, playButtonUp,
        playButtonDown, ready, gameOver, highScore, scoreboard, star,
        gameLogo, noStar, retry, firstCandy, secondCandy, mmLogo;

    public static TextureRegion barTopUp, barTopDown, bar;

    public static TextureRegion androidLogo, appleLogo;

    public static Sound dead, flap, coin, fall;

    public static BitmapFont font, shadow, whiteFont;
    private static Preferences prefs;

    public static void load() {

        logoTexture = new Texture(Gdx.files.internal("data/logo.png"));
        logoTexture.setFilter(TextureFilter.Linear, TextureFilter.Linear);

        mmLogo = new TextureRegion(logoTexture, 0, 0, 512, 114);

        texture = new Texture(Gdx.files.internal("data/texture.png"));
        texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

        playButtonUp = new TextureRegion(texture, 0, 83, 29, 16);
        playButtonDown = new TextureRegion(texture, 29, 83, 29, 16);
        playButtonUp.flip(false, true);
```

```
playButtonDown.flip(false, true);

ready = new TextureRegion(texture, 59, 83, 34, 7);
ready.flip(false, true);

retry = new TextureRegion(texture, 59, 110, 33, 7);
retry.flip(false, true);

gameOver = new TextureRegion(texture, 59, 92, 46, 7);
gameOver.flip(false, true);

scoreboard = new TextureRegion(texture, 111, 83, 97, 37);
scoreboard.flip(false, true);

star = new TextureRegion(texture, 152, 70, 10, 10);
noStar = new TextureRegion(texture, 165, 70, 10, 10);

star.flip(false, true);
noStar.flip(false, true);

highScore = new TextureRegion(texture, 59, 101, 48, 7);
highScore.flip(false, true);

gameLogo = new TextureRegion(texture, 0, 55, 135, 24);
gameLogo.flip(false, true);

bg = new TextureRegion(texture, 0, 0, 136, 43);
bg.flip(false, true);

grass = new TextureRegion(texture, 0, 43, 143, 11);
grass.flip(false, true);

logoDown = new TextureRegion(texture, 136, 0, 17, 12);
logoDown.flip(false, true);

logo = new TextureRegion(texture, 153, 0, 17, 12);
logo.flip(false, true);

logoUp = new TextureRegion(texture, 170, 0, 17, 12);
logoUp.flip(false, true);

TextureRegion[] birds = { logoDown, logo, logoUp };
logoAnimation = new Animation(0.06f, birds);
logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
// Create by flipping existing skullUp
barTopDown = new TextureRegion(barTopUp);
barTopDown.flip(false, true);

androidLogo = new TextureRegion(texture, 136, 22, 11, 14);
androidLogo.flip(false, true);

appleLogo = new TextureRegion(texture, 147, 22, 11, 14);
appleLogo.flip(false, true);

bar = new TextureRegion(texture, 136, 16, 22, 3);
bar.flip(false, true);

dead = Gdx.audio.newSound(Gdx.files.internal("data/dead.wav"));
flap = Gdx.audio.newSound(Gdx.files.internal("data/flap.wav"));
coin = Gdx.audio.newSound(Gdx.files.internal("data/coin.wav"));
```

```

        fall = Gdx.audio.newSound(Gdx.files.internal("data/fall.wav"));

        font = new BitmapFont(Gdx.files.internal("data/text.fnt"));
        font.setScale(.25f, -.25f);

        whiteFont = new
BitmapFont(Gdx.files.internal("data/whitetext.fnt"));
        whiteFont.setScale(.1f, -.1f);

        shadow = new BitmapFont(Gdx.files.internal("data/shadow.fnt"));
        shadow.setScale(.25f, -.25f);

        // Create (or retrieve existing) preferences file
        prefs = Gdx.app.getPreferences("MMGame");

        if (!prefs.contains("highScore")) {
            prefs.putInteger("highScore", 0);
        }
    }

    public static void setHighScore(int val) {
        prefs.putInteger("highScore", val);
        prefs.flush();
    }

    public static int getHighScore() {
        return prefs.getInteger("highScore");
    }

    public static void dispose() {
        // We must dispose of the texture when we are finished.
        texture.dispose();

        // Dispose sounds
        dead.dispose();
        flap.dispose();
        coin.dispose();

        font.dispose();
        shadow.dispose();
    }
}

```

## The Ground Hurts

We next move on to the GameWorld class.

1. Begin by deleting the instance variable `isAlive`. We will be revamping how we handle death.
2. We want our character to die when he hits the ground. We will be implementing this now.

Rather than creating a rectangle object representing the collision box for our grass objects and

updating them, we are just going to treat the ground as a static box for simplicity.

- Start by creating a new instance variable:

```
(import com.badlogic.gdx.math)
```

```
private Rectangle ground;
```

-Initialize it in the constructor as:

```
ground = new Rectangle(0, midPointY + 66, 136, 11);
```

- Next, we are going to change our update method to the following:

```
public void update(float delta) {
    // Add a delta cap so that if our game takes too long
    // to update, we will not break our collision detection.

    if (delta > .15f) {
        delta = .15f;
    }

    logo.update(delta);
    scroller.update(delta);

    if (scroller.collides(logo) && logo.isAlive()) {
        scroller.stop();
        logo.die();
        AssetLoader.dead.play();
    }

    if (Intersector.overlaps(logo.getBoundingBox(), ground)) {
        scroller.stop();
        logo.die();
        logo.decelerate();
    }
}
```

## Fixing our Logo

Now we must fix our errors in the Logo class

- We need to create an instance variable called:

```
private boolean isAlive
```

- We initialize this in the constructor:

```
isAlive = true;
```

- Next, generate a getter for this:

```
public boolean isAlive() {
```

```
    return isAlive;
}
```

- Now that we have defined isAlive, we make a small change to our shouldntFlap method. We don't want our logo to animate when it is dead:

```
public boolean shouldntFlap() {
    return velocity.y > 70 || !isAlive;
}
```

This method will return true if either of the two conditions above are true (meaning that if the logo has a high y velocity or is dead, we will not be flapping).

- We make another small change in the onClick method, which should only work if our bird is alive:

```
public void onClick() {
    if (isAlive) {
        AssetLoader.flap.play();
        velocity.y = -140;
    }
}
```

You may need to import: `com.mentormate.mmgame.zbhelpers.AssetLoader;`

- We are going to add two new methods: die and decelerate. These are pretty straightforward:

```
public void die() {
    isAlive = false;
    velocity.y = 0;
}
```

```
public void decelerate() {
    // We want the logo to stop accelerating downwards once it is dead.
    acceleration.y = 0;
}
```

- Lastly, add a new condition to the final if statement inside the update method:

```
if (isFalling() || !isAlive) {
```



```
...  
}
```

This is so that if our logo dies while facing upwards, it will point its nose to the ground just like the bird in the game Flappy Bird.

The completed Logo class is as follows:

```
package com.mentormate.mmgame.gameobjects;  
  
import com.badlogic.gdx.math.Circle;  
import com.badlogic.gdx.math.Vector2;  
import com.mentormate.mmgame.zbhelpers.AssetLoader;  
  
public class Logo {  
  
    private Vector2 position;  
    private Vector2 velocity;  
    private Vector2 acceleration;  
  
    private float rotation; // For handling bird rotation  
    private int width;  
    private int height;  
  
    private boolean isAlive;  
  
    private Circle boundingCircle;  
  
    public Logo(float x, float y, int width, int height) {  
        this.width = width;  
        this.height = height;  
        position = new Vector2(x, y);  
        velocity = new Vector2(0, 0);  
        acceleration = new Vector2(0, 460);  
        boundingCircle = new Circle();  
        isAlive = true;  
    }  
  
    public void update(float delta) {  
  
        velocity.add(acceleration.cpy().scl(delta));  
  
        if (velocity.y > 200) {  
            velocity.y = 200;  
        }  
  
        position.add(velocity.cpy().scl(delta));  
  
        // Set the circle's center to be (9, 6) with respect to the logo.  
        // Set the circle's radius to be 6.5f;  
        boundingCircle.set(position.x + 9, position.y + 6, 6.5f);  
  
        // Rotate counterclockwise  
        if (velocity.y < 0) {  
            rotation -= 600 * delta;  
  
            if (rotation < -20) {  
                rotation = -20;  
            }  
        }  
    }  
}
```

```

    }

    // Rotate clockwise
    if (isFalling() || !isAlive) {
        rotation += 480 * delta;
        if (rotation > 90) {
            rotation = 90;
        }
    }
}

public boolean isFalling() {
    return velocity.y > 110;
}

public boolean shouldntFlap() {
    return velocity.y > 70 || !isAlive;
}

public void onClick() {
    if (isAlive) {
        AssetLoader.flap.play();
        velocity.y = -140;
    }
}

public void die() {
    isAlive = false;
    velocity.y = 0;
}

public void decelerate() {
    acceleration.y = 0;
}

public float getX() {
    return position.x;
}

public float getY() {
    return position.y;
}

public float getWidth() {
    return width;
}

public float getHeight() {
    return height;
}

public float getRotation() {
    return rotation;
}

public Circle getBouncingCircle() {
    return boundingCircle;
}

public boolean isAlive() {
    return isAlive;
}

```

```
}  
  
}
```

In **Flappy Bird**, you score a point when your Bird passes about half way thorough each vertical pair of Pipes. We will add score when passing trough the android and apple logos . We need to create an integer that keeps track of our current score. We will do this in the GameWorld.

## Open up the GameWorld class

- Create a new instance variable:

```
private int score = 0;
```

- Next, create these these **getter** and **increment** methods:

```
public int getScore() {  
    return score;  
}
```

```
public void addScore(int increment) {  
    score += increment;  
}
```

Your GameWorld should look like this for now:

```
package com.mentormate.mmgame.gameworld;  
  
import com.badlogic.gdx.math.Intersector;  
import com.badlogic.gdx.math.Rectangle;  
import com.mentormate.mmgame.gameobjects.Logo;  
import com.mentormate.mmgame.gameobjects.ScrollHandler;  
import com.mentormate.mmgame.zbhelpers.AssetLoader;  
  
public class GameWorld {  
    private Logo logo;  
    private ScrollHandler scroller;  
    private boolean isAlive = true;  
    private Rectangle ground;  
  
    private int score = 0;  
  
    public GameWorld(int midPointY) {  
        logo = new Logo(33, midPointY - 5, 17, 12);  
        // The grass should start 66 pixels below the midPointY  
        scroller = new ScrollHandler(midPointY + 66);  
        ground = new Rectangle(0, midPointY + 66, 136, 11);  
    }  
  
    public void update(float delta) {  
        // Add a delta cap so that if our game takes too long  
        // to update, we will not break our collision detection.  
  
        if (delta > .15f) {  
            delta = .15f;  
        }  
  
        logo.update(delta);  
        scroller.update(delta);  
  
        if (scroller.collides(logo) && logo.isAlive()) {
```

```

        scroller.stop();
        logo.die();
        AssetLoader.dead.play();
    }

    if (Intersector.overlaps(logo.getBoundingBox(), ground)) {
        scroller.stop();
        logo.die();
        logo.decelerate();
    }
}

public Logo getLogo() {
    return logo;
}

public ScrollHandler getScroller() {
    return scroller;
}

public int getScore() {
    return score;
}

public void addScore(int increment) {
    score += increment;
}
}

```

## Increment the Score

The logic for incrementing our score will be in our ScrollHandler class. Let's open it.

We need a reference to our GameWorld so that we can change its score. So, we are going to ask for a GameWorld object inside our constructor, and store the passed in parameter as an instance variable called gameWorld. Make sure you import GameWorld (com.mentormate.mmgame.gameworld.GameWorld).

This is the new constructor:

```

public ScrollHandler(GameWorld gameWorld, float yPos) {
    this.gameWorld = gameWorld;
    frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
    backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11,
        SCROLL_SPEED);
    pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED, yPos);
    bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 16,
11,
        SCROLL_SPEED);
    pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60,
SCROLL_SPEED,
        yPos);
    bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 16,

```

```
11,
    SCROLL_SPEED);
}
```

Quickly go back to the **GameWorld** and update our **ScrollHandler** constructor call to the following:

```
scroller = new ScrollHandler(this, midPointY + 66);
```

The general logic will be this:

- If a logo's midpoint with respect to x is lesser than bird's beak, we are going to add 1 to our score.
- Since we do not want this to repeat, we need to keep a **boolean** variable for EACH bonusLogo called **isScored**. Only if **isScored** is false, will we add one to the score (Setting **isScored** to true in the process. If we just need to change **isScored** back to false when the BonusLogo resets, then this will work the next time the BonusLogo returns).

This will be our new **scored method** that implements the above logic:

```
public boolean scored(Logo bird) {
    if (bonusLogo1.collides(bird)) {
        if (!bonusLogo1.isScored()) {
            bonusLogo1.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            return true;
        } else {
            return true;
        }
    } else if (bonusLogo2.collides(bird)) {
        if (!bonusLogo2.isScored()) {
            bonusLogo2.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            return true;
        } else {
            return true;
        }
    } else {
        return false;
    }
}
```

To solve all the errors that we have just introduced, we must do the following:

1. Import AssetLoader (**com.mentormate.mmgame.mmhelpers.AssetLoader**).
2. Create the following **addScore** method:

```
private void addScore(int increment) {
    gameWorld.addScore(increment);
}
```

3. We must now move on to our BonusLogo class and implement our boolean: **isScored**.

This is the completed **ScrollHandler**:

```

package com.mentormate.mmgame.gameobjects;

import com.mentormate.mmgame.gameworld.GameWorld;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class ScrollHandler { // ScrollHandler will create all six objects that
                                // we need.
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bonusLogo1, bonusLogo2;
    private GameWorld gameWorld;

    // ScrollHandler will use the constants below to determine
    // how fast we need to scroll and also determine
    // the size of the gap between each pair of pipes.

    // Capital letters are used by convention when naming constants.
    public static final int SCROLL_SPEED = -59;
    public static final int PIPE_GAP = 143;

    // Constructor receives a float that tells us where we need to create our
    // Grass and Pipe objects.
    public ScrollHandler(GameWorld gameWorld, float yPos) {
        this.gameWorld = gameWorld;
        frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
        backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11,
                               SCROLL_SPEED);
        pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED, yPos);
        bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 16,
11,
                               SCROLL_SPEED);
        pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60,
SCROLL_SPEED,
                               yPos);
        bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 16,
11,
                               SCROLL_SPEED);
    }

    public void update(float delta) {
        // Update our objects
        frontGrass.update(delta);
        backGrass.update(delta);
        pipe1.update(delta);
        bonusLogo1.update(delta);
        pipe2.update(delta);
        bonusLogo2.update(delta);

        // Check if any of the pipes are scrolled left,
        // and reset accordingly the bonusLogo
        if (pipe1.isScrolledLeft()) {
            pipe1.reset(pipe2.getTailX() + PIPE_GAP);
            bonusLogo2.reset(pipe2.getTailX() + PIPE_GAP / 2);
        } else if (pipe2.isScrolledLeft()) {
            pipe2.reset(pipe1.getTailX() + PIPE_GAP);
            bonusLogo1.reset(pipe1.getTailX() + PIPE_GAP / 2);
        }

        // Same with grass
        if (frontGrass.isScrolledLeft()) {
            frontGrass.reset(backGrass.getTailX());

```

```

        } else if (backGrass.isScrolledLeft()) {
            backGrass.reset(frontGrass.getTailX());
        }
    }

    public void stop() {
        frontGrass.stop();
        backGrass.stop();
        pipe1.stop();
        pipe2.stop();
        bonusLogo1.stop();
        bonusLogo2.stop();
    }

    // Return true if ANY pipe hits the logo.
    public boolean collides(Logo logo) {
        return (pipe1.collides(logo) || pipe2.collides(logo));
    }

    public boolean scored(Logo bird) {
        if (bonusLogo1.collides(bird)) {
            if (!bonusLogo1.isScored()) {
                bonusLogo1.setScored(true);
                AssetLoader.coin.play();
                addScore(1);
                return true;
            } else {
                return true;
            }
        } else if (bonusLogo2.collides(bird)) {
            if (!bonusLogo2.isScored()) {
                bonusLogo2.setScored(true);
                AssetLoader.coin.play();
                addScore(1);
                return true;
            } else {
                return true;
            }
        } else {
            return false;
        }
    }

    // The getters for our five instance variables
    public Grass getFrontGrass() {
        return frontGrass;
    }

    public Grass getBackGrass() {
        return backGrass;
    }

    public Pipe getPipe1() {
        return pipe1;
    }

    public Pipe getPipe2() {

```

```

        return pipe2;
    }

    public BonusLogo getBonusLogo1() {
        return bonusLogo1;
    }

    public BonusLogo getBonusLogo2() {
        return bonusLogo2;
    }
}

```

## BonusLogo Class - isScored boolean

- Create a new instance variable:

```
private boolean isScored = false;
```

This boolean is set to true in the ScrollHandler when a bonusLogo has been awarded to the player for scoring object.

- Update the reset method so that isScored is set to false when the BonusLogo resets:

```

@Override
public void reset(float newX) {
    super.reset(newX);
    height = r.nextInt(90) + 15;
    isScored = false;
}

```

Next, we just have to create the getter and setter methods:

```

public boolean isScored() {
    return isScored;
}

```

```

public void setScored(boolean b) {
    isScored = b;
}

```



Your BonusLogo class should now look like this:

```
package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Intersector;

public class BonusLogo extends Scrollable {

    private Random r;
    private Circle boundingCircle;

    private boolean isScored = false;

    // When BonusLogo's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public BonusLogo(float x, float y, int width, int height, float
scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
        boundingCircle = new Circle();
    }

    @Override
    public void update(float delta) {
        // Call the update method in the superclass (Scrollable)
        super.update(delta);
        boundingCircle.set(position.x + 9, position.y + 6, 6.5f);
    }

    public boolean collides(Logo logo) {
        if (position.x < logo.getX() + logo.getWidth()) {
            return (Intersector.overlaps(logo.getBouncingCircle(),
                boundingCircle));
        }
        return false;
    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        position.y = r.nextInt(90) + 15;
        isScored = false;
    }

    public Circle getBouncingCircle() {
        return boundingCircle;
    }

    public boolean isScored() {
        return isScored;
    }
}
```

```

    public void setScored(boolean b) {
        isScored = b;
    }
}

```

Last just update the GameWord class update method to:

```

public void update(float delta) {
    // Add a delta cap so that if our game takes too long
    // to update, we will not break our collision detection.

    if (delta > .15f) {
        delta = .15f;
    }

    logo.update(delta);
    scroller.update(delta);

    if (scroller.collides(logo) && logo.isAlive()) {
        scroller.stop();
        logo.die();
        AssetLoader.dead.play();
    } else if (scroller.scored(logo) && logo.isAlive()) {
        // TODO add logic to hide bonus Logo
    }

    if (Intersector.overlaps(logo.getBoundingBox(), ground)) {
        scroller.stop();
        logo.die();
        logo.decelerate();
    }
}

```

## Run the code!

You should hear the Coin.wav file play each time that you score . But we don't want to just hear our score go up. We want to see our score change!

So we are going to render some text to the screen for this.

## Displaying the Score in GameRenderer

Displaying text is very simple. Between our batcher.begin() and batcher.end() calls, we must include a line like this:

```
AssetLoader.shadow.draw(batch, "hello world", x, y);
```

A BitmapFont object, as shown above, has a draw method that takes in the current SpriteBatch, a

String, and the x and y coordinates to draw the text at.

- Add the following lines of code right before the `batcher.end()` call in the render method:

... rest of render method

```
// Convert integer into String
String score = myWorld.getScore() + "";

// Draw shadow first
AssetLoader.shadow.draw(batcher, "" + myWorld.getScore(), (136 / 2) - (3 * score.length()), 12);
// Draw text
AssetLoader.font.draw(batcher, "" + myWorld.getScore(), (136 / 2) - (3 * score.length() - 1), 11);
batcher.end();

}
```

We are first converting the score integer to a String, so that we can draw it using our `BitmapFont`. We calculate the appropriate X location by checking the length of the score, so that we can center the score as best we can.

Here's the full `GameRenderer` class:

```
package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
```

```

private TextureRegion bg, grass;
private Animation logoAnimation;
private TextureRegion logoMid, logoDown, logoUp;
private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
    bonusLogoApple;

private Logo bird;
private ScrollHandler scroller;
private Grass frontGrass, backGrass;
private Pipe pipe1, pipe2;
private BonusLogo bLogo1, bLogo2;

private GameWorld myWorld;
private OrthographicCamera cam;
private ShapeRenderer shapeRenderer;

private SpriteBatch batcher;

private int midPointY;
private int gameHeight;

public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
    myWorld = world;

    // The word "this" refers to this instance.
    // We are setting the instance variables' values to be that of the
    // parameters passed in from GameScreen.
    this.gameHeight = gameHeight;
    this.midPointY = midPointY;

    cam = new OrthographicCamera();
    cam.setToOrtho(true, 136, gameHeight);

    batcher = new SpriteBatch();
    batcher.setProjectionMatrix(cam.combined);
    shapeRenderer = new ShapeRenderer();
    shapeRenderer.setProjectionMatrix(cam.combined);

    // Call helper methods to initialize instance variables
    initGameObjects();
    initAssets();
}

private void initGameObjects() {
    logo = myWorld.getLogo();
    scroller = myWorld.getScroller();
    frontGrass = scroller.getFrontGrass();
    backGrass = scroller.getBackGrass();
    pipe1 = scroller.getPipe1();
    pipe2 = scroller.getPipe2();
    bLogo1 = scroller.getBonusLogo1();
    bLogo2 = scroller.getBonusLogo2();
}

private void initAssets() {
    bg = AssetLoader.bg;
    grass = AssetLoader.grass;
    logoAnimation = AssetLoader.logoAnimation;
    logoMid = AssetLoader.logo;
    logoDown = AssetLoader.logoDown;
    logoUp = AssetLoader.logoUp;
    barTopUp = AssetLoader.barTopUp;

```

```

        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
                     frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
                     backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
                     pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
                     pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
                     pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
                     pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawPipes() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.
        batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
                     pipe1.getHeight());
        batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
                     pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

        batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
                     pipe2.getHeight());
        batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
                     pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
                     bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
                     bLogo2.getWidth(), bLogo2.getHeight());
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
    }

```

```

Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

shapeRenderer.begin(ShapeType.Filled);

// Draw Background color
shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
shapeRenderer.rect(0, 0, 136, midPointY + 66);

// Draw Grass
shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
shapeRenderer.rect(0, midPointY + 66, 136, 11);

// Draw Dirt
shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
shapeRenderer.rect(0, midPointY + 77, 136, 52);

shapeRenderer.end();

batcher.begin();
batcher.disableBlending();
batcher.draw(bg, 0, midPointY + 23, 136, 43);

// 1. Draw Grass
drawGrass();

// 2. Draw Pipes
drawPipes();
batcher.enableBlending();

// 3. Draw Bar Tops (requires transparency)
drawBarTops();

if (logo.shouldntFlap()) {
    batcher.draw(logoMid, logo.getX(), logo.getY(),
        logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
        logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
} else {
    batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
        logo.getY(), logo.getWidth() / 2.0f,
        logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
        1, 1, logo.getRotation());
}

batcher.end();
batcher.begin();
drawLogos();

// Convert integer into String
String score = myWorld.getScore() + "";

// Draw shadow first
AssetLoader.shadow.draw(batcher, "" + myWorld.getScore(), (136 / 2)
    - (3 * score.length()), 12);

// Draw text
AssetLoader.font.draw(batcher, "" + myWorld.getScore(), (136 / 2)
    - (3 * score.length() - 1), 11);
batcher.end();
}

```

```
}
```

## Source Code

[DOWNLOAD SOURCE](#)

# Part 10 - GameStates and High Score

In this tutorial, we are going to discuss GameStates, which we will use to implement a 'start on touch,' pausing, and restarting. Then we will make use of the libGDX preferences to keep track of the high score. At the end of Part 10, you will have a fully featured Flappy Bird clone.

In Part 11, we will add our final touches, adding splash screens and etc.

If you are ready, lets get started.

## Quick-fix - Ceiling collisions

Forgot to add ceiling collision for our bird. Update the update method inside the Logo class!

```
public void update(float delta) {  
  
    velocity.add(acceleration.cpy().scl(delta));  
  
    if (velocity.y > 200) {  
        velocity.y = 200;  
    }  
  
    // CEILING CHECK  
    if (position.y < -13) {  
        position.y = -13;  
        velocity.y = 0;  
    }  
  
    position.add(velocity.cpy().scl(delta));  
  
    // Set the circle's center to be (9, 6) with respect to the logo.  
    // Set the circle's radius to be 6.5f;  
    boundingCircle.set(position.x + 9, position.y + 6, 6.5f);  
  
    // Rotate counterclockwise  
    if (velocity.y < 0) {  
        rotation -= 600 * delta;  
  
        if (rotation < -20) {  
            rotation = -20;  
        }  
    }  
}
```

# Adding GameStates

The idea behind GameStates is to divide our game into multiple 'states' such as "RUNNING" or "GAMEOVER". Then we will use if statements or switches to control the flow of our game depending on what the state is.

An easy way to implement GameStates is to create an Enum, which is just a variable that can only take certain values that have been defined for it.,

We add our enum to our GameWorld. Add the following code somewhere inside the GameWorld class:

```
public enum GameState {  
  
    READY, RUNNING, GAMEOVER  
  
}
```

- Now we can create GameState variables just like any other variables. Create an instance variable:

```
private GameState currentState;
```

- Initialize it in the constructor:

```
currentState = GameState.READY;
```

```
package com.mentormate.mmgame.gameworld;  
  
import com.badlogic.gdx.math.Intersector;  
import com.badlogic.gdx.math.Rectangle;  
import com.mentormate.mmgame.gameobjects.Logo;  
import com.mentormate.mmgame.gameobjects.ScrollHandler;  
import com.mentormate.mmgame.mmhelpers.AssetLoader;  
  
public class GameWorld {  
    private Logo logo;  
    private ScrollHandler scroller;  
    private boolean isAlive = true;  
    private Rectangle ground;  
    private int score = 0;  
    private GameState currentState;  
  
    public enum GameState {  
  
        READY, RUNNING, GAMEOVER  
  
    }  
  
    public GameWorld(int midPointY) {  
        logo = new Logo(33, midPointY - 5, 17, 12);
```



```

        // The grass should start 66 pixels below the midPointY
        scroller = new ScrollHandler(this, midPointY + 66);
        ground = new Rectangle(0, midPointY + 66, 136, 11);
        currentState = GameState.READY;
    }

```

We next need to make a couple of changes to our update method. In fact, we are going to change its name to updateRunning.

Once you have made this change, create a new **update** method and an **updateReady** method as shown below:

```

    public void update(float delta) {

        switch (currentState) {
            case READY:
                updateReady(delta);
                break;

            case RUNNING:
            default:
                updateRunning(delta);
                break;
        }
    }

    private void updateReady(float delta) {
        // Do nothing for now
    }

```

The **update** method now checks the current state of the game before determining which more specific update method to call.

Next, we must change our GameState when our Logo dies. We will consider the Logo fully dead when he hits the ground. Inside the **updateRunning** method (changed from the original name of **update**, like I asked you to above), add this inside the final if statement:  
**currentState = GameState.GAMEOVER;**

And finally, add these various methods, which will handle GameState changes, restarting, and so on.

**IsReady** simply returns true if the currentState is GameState.READY. **Start** changes the currentState to GameState.RUNNING.

**Restart** is a more interesting - it resets all the instance variables that have changed since the game started! This is how we will handle restarting the game.

When the **restart** method is called, we will **cascade down** to the other objects and call their **onRestart** methods. The result should be that every instance variable that has changed throughout the flow of the game will be reset - like dominos.

What are the **arguments** that we are passing into the onRestart methods? These are all the initial values of variables that may have changed when our game was running. For example, our logo's y position may have changed, so we pass in the *original* y position.

```

    public void restart() {

```

```

        currentState = GameState.READY;
        score = 0;
        logo.onRestart(midPointY - 5);
        scroller.onRestart();
        currentState = GameState.READY;
    }

```

We also need access to the midPointY, which is currently not being stored as an instance variable in the constructor. Let's change that.

Add this instance variable:

```
public int midPointY;
```

Initialize it in the constructor:

```
this.midPointY = midPointY;
```

In case you got lost somewhere, here's the full GameWorld class:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.math.Rectangle;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class GameWorld {
    private Logo logo;
    private ScrollHandler scroller;
    private boolean isAlive = true;
    private Rectangle ground;
    private int score = 0;
    private GameState currentState;
    public int midPointY;

    public enum GameState {

        READY, RUNNING, GAMEOVER

    }

    public GameWorld(int midPointY) {
        logo = new Logo(33, midPointY - 5, 17, 12);
        // The grass should start 66 pixels below the midPointY
        scroller = new ScrollHandler(this, midPointY + 66);
        ground = new Rectangle(0, midPointY + 66, 136, 11);
        currentState = GameState.READY;
        this.midPointY = midPointY;
    }

    public void update(float delta) {

        switch (currentState) {
            case READY:
                updateReady(delta);
                break;

            case RUNNING:
            default:

```

```

        updateRunning(delta);
        break;
    }

}

private void updateReady(float delta) {
    // Do nothing for now
}

public void updateRunning(float delta) {
    // Add a delta cap so that if our game takes too long
    // to update, we will not break our collision detection.

    if (delta > .15f) {
        delta = .15f;
    }

    logo.update(delta);
    scroller.update(delta);

    if (scroller.collides(logo) && logo.isAlive()) {
        scroller.stop();
        logo.die();
        AssetLoader.dead.play();
    } else if (scroller.scored(logo) && logo.isAlive()) {
        // TODO add logic to hide bonus Logo
    }

    if (Intersector.overlaps(logo.getBoundingBox(), ground)) {
        scroller.stop();
        logo.die();
        logo.decelerate();
        currentState = GameState.GAMEOVER;
    }
}

public Logo getLogo() {
    return logo;
}

public ScrollHandler getScroller() {
    return scroller;
}

public int getScore() {
    return score;
}

public void addScore(int increment) {
    score += increment;
}

public void restart() {
    currentState = GameState.READY;
    score = 0;
    logo.onRestart(midPointY - 5);
    scroller.onRestart();
    currentState = GameState.READY;
}

```

```

    public boolean isReady() {
        return currentState == GameState.READY;
    }

    public void start() {
        currentState = GameState.RUNNING;
    }

    public boolean isGameOver() {
        return currentState == GameState.GAMEOVER;
    }
}

```

We will now add the **onRestart** methods to our Logo and Scroller objects. Let's start with the easier one - the Logo object.

## Resetting the Logo

Create the onRestart method for our logo. Inside, we must undo all the changes that were done during the course of the gameplay:

```

    public void onRestart(int y) {
        rotation = 0;
        position.y = y;
        velocity.x = 0;
        velocity.y = 0;
        acceleration.x = 0;
        acceleration.y = 460;
        isAlive = true;
    }

```

That's it! Here's the full class:

```

package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Vector2;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class Logo {

    private Vector2 position;
    private Vector2 velocity;
    private Vector2 acceleration;

    private float rotation; // For handling bird rotation
    private int width;
    private int height;

    private boolean isAlive;

    private Circle boundingCircle;

```

```

public Logo(float x, float y, int width, int height) {
    this.width = width;
    this.height = height;
    position = new Vector2(x, y);
    velocity = new Vector2(0, 0);
    acceleration = new Vector2(0, 460);
    boundingCircle = new Circle();
    isAlive = true;
}

public void update(float delta) {

    velocity.add(acceleration.cpy().scl(delta));

    if (velocity.y > 200) {
        velocity.y = 200;
    }

    // CEILING CHECK
    if (position.y < -13) {
        position.y = -13;
        velocity.y = 0;
    }

    position.add(velocity.cpy().scl(delta));

    // Set the circle's center to be (9, 6) with respect to the logo.
    // Set the circle's radius to be 6.5f;
    boundingCircle.set(position.x + 9, position.y + 6, 6.5f);

    // Rotate counterclockwise
    if (velocity.y < 0) {
        rotation -= 600 * delta;

        if (rotation < -20) {
            rotation = -20;
        }
    }

    // Rotate clockwise
    if (isFalling() || !isAlive) {
        rotation += 480 * delta;
        if (rotation > 90) {
            rotation = 90;
        }
    }
}

public void onRestart(int y) {
    rotation = 0;
    position.y = y;
    velocity.x = 0;
    velocity.y = 0;
    acceleration.x = 0;
    acceleration.y = 460;
    isAlive = true;
}

public boolean isFalling() {
    return velocity.y > 110;
}

```

```

    }

    public boolean shouldntFlap() {
        return velocity.y > 70 || !isAlive;
    }

    public void onClick() {
        if (isAlive) {
            AssetLoader.flap.play();
            velocity.y = -140;
        }
    }

    public void die() {
        isAlive = false;
        velocity.y = 0;
    }

    public void decelerate() {
        acceleration.y = 0;
    }

    public float getX() {
        return position.x;
    }

    public float getY() {
        return position.y;
    }

    public float getWidth() {
        return width;
    }

    public float getHeight() {
        return height;
    }

    public float getRotation() {
        return rotation;
    }

    public Circle getBouncingCircle() {
        return bouncingCircle;
    }

    public boolean isAlive() {
        return isAlive;
    }
}

```

## OnRestart - ScrollHandler

We must now go to the ScrollHandler class and create a similar method, resetting all the instance variables of the ScrollHandler class! Notice that we call some more non-existent onRestart methods. We will go deeper and complete them all.

```

public void onRestart() {

```

```

        frontGrass.onRestart(0, SCROLL_SPEED);
        backGrass.onRestart(frontGrass.getTailX(), SCROLL_SPEED);
        pipe1.onRestart(210, SCROLL_SPEED);
        bonusLogo1.onRestart(pipe1.getTailX() + PIPE_GAP / 2, SCROLL_SPEED);
        pipe2.onRestart(pipe1.getTailX() + PIPE_GAP, SCROLL_SPEED);
        bonusLogo2.onRestart(pipe2.getTailX() + PIPE_GAP / 2, SCROLL_SPEED);
    }

```

## Grass Class onRestart

This is really simple. We just need to move the grass back to their original positions and reset their speed back to SCROLL\_SPEED:

```

package com.mentormate.mmgame.gameobjects;

public class Grass extends Scrollable {

    // When Grass's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public Grass(float x, float y, int width, int height, float scrollSpeed) {
        super(x, y, width, height, scrollSpeed);
    }

    public void onRestart(float x, float scrollSpeed) {
        position.x = x;
        velocity.x = scrollSpeed;
    }
}

```

## Pipe onRestart

Pipe is slightly more complicated, but still very simple. Add this method:

```

public void onRestart(float x, float scrollSpeed) {
    velocity.x = scrollSpeed;
    reset(x);
}

```

```

package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.math.Rectangle;

public class Pipe extends Scrollable {

```

```

    private Random r;

    private Rectangle barTopUp, barTopDown, barUp, barDown;

    public static final int VERTICAL_GAP = 45;
    public static final int PIPE_TOP_WIDTH = 24;
    public static final int PIPE_TOP_HEIGHT = 11;
    private float groundY;

    // When Pipe's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public Pipe(float x, float y, int width, int height, float scrollSpeed,
               float groundY) {
        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
        barTopUp = new Rectangle();
        barTopDown = new Rectangle();
        barUp = new Rectangle();
        barDown = new Rectangle();
        this.groundY = groundY;
    }

    @Override
    public void update(float delta) {
        // Call the update method in the superclass (Scrollable)
        super.update(delta);

        // The set() method allows you to set the top left corner's x, y
        // coordinates,
        // along with the width and height of the rectangle

        barUp.set(position.x, position.y, width, height);
        barDown.set(position.x, position.y + height + VERTICAL_GAP, width,
                    groundY - (position.y + height + VERTICAL_GAP));

        // Our barTop width is 24. The bar is only 22 pixels wide. So the
        // must be shifted by 1 pixel to the left (so that the MM is
        // centered
        // with respect to its bar).

        // This shift is equivalent to: (PIPE_TOP_WIDTH - width) / 2
        barTopUp.set(position.x - (PIPE_TOP_WIDTH - width) / 2, position.y
                    + height - PIPE_TOP_HEIGHT, PIPE_TOP_WIDTH,
PIPE_TOP_HEIGHT);
        barTopDown.set(position.x - (PIPE_TOP_WIDTH - width) / 2, barDown.y,
                    PIPE_TOP_WIDTH, PIPE_TOP_HEIGHT);
    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        height = r.nextInt(90) + 15;
    }

    public boolean collides(Logo logo) {
        if (position.x < logo.getX() + logo.getWidth()) {

```



```

        return (Intersector.overlaps(logo.getBoundingCircle(), barUp)
            || Intersector.overlaps(logo.getBoundingCircle(),
barDown)
            || Intersector.overlaps(logo.getBoundingCircle(),
barTopUp) || Intersector
                .overlaps(logo.getBoundingCircle(),
barTopDown));
    }
    return false;
}

public Rectangle getBarTopUp() {
    return barTopUp;
}

public Rectangle getBarTopDown() {
    return barTopDown;
}

public Rectangle getBarUp() {
    return barUp;
}

public Rectangle getBarDown() {
    return barDown;
}

public float getGroundY() {
    return groundY;
}

public void onRestart(float x, float scrollSpeed) {
    velocity.x = scrollSpeed;
    reset(x);
}
}

```

The Same for BonusLogo:

```

package com.mentormate.mmgame.gameobjects;

import java.util.Random;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Intersector;

public class BonusLogo extends Scrollable {

    private Random r;
    private Circle boundingCircle;

    private boolean isScored = false;

    // When BonusLogo's constructor is invoked, invoke the super (Scrollable)
    // constructor
    public BonusLogo(float x, float y, int width, int height, float
scrollSpeed) {

```

```

        super(x, y, width, height, scrollSpeed);
        // Initialize a Random object for Random number generation
        r = new Random();
        boundingCircle = new Circle();
    }

    @Override
    public void update(float delta) {
        // Call the update method in the superclass (Scrollable)
        super.update(delta);
        boundingCircle.set(position.x + 9, position.y + 6, 6.5f);
    }

    public boolean collides(Logo logo) {
        if (position.x < logo.getX() + logo.getWidth()) {
            return (Intersector.overlaps(logo.getBoundingCircle(),
                boundingCircle));
        }
        return false;
    }

    @Override
    public void reset(float newX) {
        // Call the reset method in the superclass (Scrollable)
        super.reset(newX);
        // Change the height to a random number
        position.y = r.nextInt(90) + 15;
        isScored = false;
    }

    public Circle getBoundingCircle() {
        return boundingCircle;
    }

    public boolean isScored() {
        return isScored;
    }

    public void setScored(boolean b) {
        isScored = b;
    }

    public void onRestart(float x, float scrollSpeed) {
        velocity.x = scrollSpeed;
        reset(x);
    }
}

```

And we are done! Let's look back at what we've accomplished. We started by adding GameState in the GameWorld class. We then added a restart method, which asked Logo to restart and ScrollHandler to restart, which in turn asked its two Pipe two bonusLogo objects and two Grass objects to restart. Now all we need to do is let the game know when to restart.

## Go to the InputHandler

Our InputHandler needs a reference to our GameWorld object, so that we can determine what the current GameState is and handle touches correctly. Rather than adding another argument to the

constructor, I'm going to change the existing constructor to be this:

```
public InputHandler(GameWorld myWorld) {  
    // myLogo now represents the gameWorld's logo.  
    this.myWorld = myWorld;  
    myLogo = myWorld.getLogo();  
}
```

- Of course create an instance variable for myWorld (above the constructor): private GameWorld myWorld;
- Don't forget to import!

Next, we are going to update our touchDown method:

```
@Override  
public boolean touchDown(int screenX, int screenY, int pointer, int  
button) {  
    if (myWorld.isReady()) {  
        myWorld.start();  
    }  
  
    myLogo.onClick();  
  
    if (myWorld.isGameOver()) {  
        // Reset all variables, go to GameState.READ  
        myWorld.restart();  
    }  
    return true; // Return true to say we handled the touch.  
}
```

## Here's the full class!

```
package com.mentormate.mmgame.mmhelpers;  
  
import com.badlogic.gdx.InputProcessor;  
import com.mentormate.mmgame.gameobjects.Logo;  
import com.mentormate.mmgame.gameworld.GameWorld;  
  
public class InputHandler implements InputProcessor {  
  
    private Logo myLogo;  
    private GameWorld myWorld;  
  
    // Ask for a reference to the Bird when InputHandler is created.  
    public InputHandler(GameWorld myWorld) {  
        // myLogo now represents the gameWorld's logo.  
        this.myWorld = myWorld;  
        myLogo = myWorld.getLogo();  
    }  
  
    @Override  
    public boolean touchDown(int screenX, int screenY, int pointer, int  
button) {
```

```

        if (myWorld.isReady()) {
            myWorld.start();
        }

        myLogo.onClick();

        if (myWorld.isGameOver()) {
            // Reset all variables, go to GameState.READ
            myWorld.restart();
        }
        return true; // Return true to say we handled the touch.
    }

    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button)
    {
        return false;
    }

    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
        return false;
    }

    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        return false;
    }

    @Override
    public boolean scrolled(int amount) {
        return false;
    }
}

```

## Fixing GameScreen

Of course, since we changed the constructor for our InputHandler, we need to update our GameScreen, where we initialized the InputHandler using its constructor.

Change this line:

```
Gdx.input.setInputProcessor(new InputHandler(world.getLogo()));
```

To this line:

```
Gdx.input.setInputProcessor(new InputHandler(world));
```

The full class should be:

```
package com.mentormate.mmgame.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.mentormate.mmgame.gameworld.GameRenderer;
import com.mentormate.mmgame.gameworld.GameWorld;
import com.mentormate.mmgame.mmhelpers.InputHandler;

public class GameScreen implements Screen {

    private GameWorld world;
    private GameRenderer renderer;
    private float runTime;

    // This is the constructor, not the class declaration
    public GameScreen() {

        float screenWidth = Gdx.graphics.getWidth();
        float screenHeight = Gdx.graphics.getHeight();
        float gameWidth = 136;
        float gameHeight = screenHeight / (screenWidth / gameWidth);

        int midPointY = (int) (gameHeight / 2);

        world = new GameWorld(midPointY);
        renderer = new GameRenderer(world, (int) gameHeight, midPointY);

        Gdx.input.setInputProcessor(new InputHandler(world));

    }

    @Override
    public void render(float delta) {
        runTime += delta;
        world.update(delta);
        renderer.render(runTime);
    }

    @Override
    public void resize(int width, int height) {

    }

    @Override
    public void show() {
        Gdx.app.log("GameScreen", "show called");
    }

    @Override
    public void hide() {
        Gdx.app.log("GameScreen", "hide called");
    }

    @Override
```

```

    public void pause() {
        Gdx.app.log("GameScreen", "pause called");
    }

    @Override
    public void resume() {
        Gdx.app.log("GameScreen", "resume called");
    }

    @Override
    public void dispose() {
        // Leave blank
    }
}

```

## Changing GameRenderer

Our restarting code is setup! Now when the game starts, it will start in the ready state, in which nothing will happen. We will tap the screen to start the game. When our logo dies, we will enter the game over state, at which time we can tap the screen again to restart.

No buttons yet, but it's a start!

Now to make this more intuitive, we are going to make changes to the Game Renderer, displaying some useful information.

Change your GameRenderer render method to the one shown below:

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class GameRenderer {

```

```

// Game Objects
private Logo logo;

// Game Assets
private TextureRegion bg, grass;
private Animation logoAnimation;
private TextureRegion logoMid, logoDown, logoUp;
private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
    bonusLogoApple;

private Logo bird;
private ScrollHandler scroller;
private Grass frontGrass, backGrass;
private Pipe pipe1, pipe2;
private BonusLogo bLogo1, bLogo2;

private GameWorld myWorld;
private OrthographicCamera cam;
private ShapeRenderer shapeRenderer;

private SpriteBatch batcher;

private int midPointY;
private int gameHeight;

public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
    myWorld = world;

    // The word "this" refers to this instance.
    // We are setting the instance variables' values to be that of the
    // parameters passed in from GameScreen.
    this.gameHeight = gameHeight;
    this.midPointY = midPointY;

    cam = new OrthographicCamera();
    cam.setToOrtho(true, 136, gameHeight);

    batcher = new SpriteBatch();
    batcher.setProjectionMatrix(cam.combined);
    shapeRenderer = new ShapeRenderer();
    shapeRenderer.setProjectionMatrix(cam.combined);

    // Call helper methods to initialize instance variables
    initGameObjects();
    initAssets();
}

private void initGameObjects() {
    logo = myWorld.getLogo();
    scroller = myWorld.getScroller();
    frontGrass = scroller.getFrontGrass();
    backGrass = scroller.getBackGrass();
    pipe1 = scroller.getPipe1();
    pipe2 = scroller.getPipe2();
    bLogo1 = scroller.getBonusLogo1();
    bLogo2 = scroller.getBonusLogo2();
}

private void initAssets() {
    bg = AssetLoader.bg;
    grass = AssetLoader.grass;

```

```

        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        logoDown = AssetLoader.logoDown;
        logoUp = AssetLoader.logoUp;
        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
            frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
            backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawPipes() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.
        batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
            pipe1.getHeight());
        batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
            pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

        batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
            pipe2.getHeight());
        batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
            pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
            bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
            bLogo2.getWidth(), bLogo2.getHeight());
    }

```



```

}

public void render(float runTime) {

    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    shapeRenderer.begin(ShapeType.Filled);

    // Draw Background color
    shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
    shapeRenderer.rect(0, 0, 136, midPointY + 66);

    // Draw Grass
    shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 66, 136, 11);

    // Draw Dirt
    shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 77, 136, 52);

    shapeRenderer.end();

    batcher.begin();
    batcher.disableBlending();
    batcher.draw(bg, 0, midPointY + 23, 136, 43);

    // 1. Draw Grass
    drawGrass();

    // 2. Draw Pipes
    drawPipes();
    batcher.enableBlending();

    // 3. Draw Bar Tops (requires transparency)
    drawBarTops();

    if (logo.shouldntFlap()) {
        batcher.draw(logoMid, logo.getX(), logo.getY(),
            logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
            logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
    } else {
        batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
            logo.getY(), logo.getWidth() / 2.0f,
            logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
            1, 1, logo.getRotation());
    }

    // TEMPORARY CODE! We will fix this section later:

    if (myWorld.isReady()) {
        // Draw shadow first
        AssetLoader.shadow.draw(batcher, "Touch me", (136 / 2) - (42),
76);

        // Draw text
        AssetLoader.font
            .draw(batcher, "Touch me", (136 / 2) - (42 - 1),
75);
    } else {

```

```

        if (myWorld.isGameOver()) {
            AssetLoader.shadow.draw(batcher, "Game Over", 25, 56);
            AssetLoader.font.draw(batcher, "Game Over", 24, 55);

            AssetLoader.shadow.draw(batcher, "Try again?", 23, 76);
            AssetLoader.font.draw(batcher, "Try again?", 24, 75);

        }

        // Convert integer into String
        String score = myWorld.getScore() + "";

        // Draw shadow first
        AssetLoader.shadow.draw(batcher, "" + myWorld.getScore(), (136
/ 2)
                                - (3 * score.length()), 12);
        // Draw text
        AssetLoader.font.draw(batcher, "" + myWorld.getScore(), (136 /
2)
                                - (3 * score.length() - 1), 11);

    }

    batcher.end();
    batcher.end();

    batcher.begin();
    drawLogos();
    batcher.end();

}

}

```



Our gameplay is now finished. Let's wrap things up by implementing high score!

## Implementing High Score

The easiest way to store small data for a libGDX game is to use a Preferences object. A Preferences object maps key-value pairs. This means that you can store some key and a corresponding value, and you can pull the values out!

Let's see an example:

```
// We specify the name of the Preferences File
Preferences prefs = Gdx.app.getPreferences("PreferenceName");// We store the value 10 with the key
of "highScore"
prefs.putInteger("highScore", 10);
prefs.flush(); // This saves the preferences file.
```

A week later, you try running:

```
System.out.println(prefs.getInteger("highScore"));
```

The resulting value will be 10!

So, the three important methods you need to know are:

1. put...
2. get...
3. and flush (for saving).

You could store other things like:

```
putBoolean("soundEnabled", true); // getBoolean("soundEnabled") retrieves a boolean.
putString("playerName", "James");
```

Let's implement this for high score.

We already did it in our AssetLoader class now we are just going to discuss it .

## Open up the AssetLoader class.

-We've Create an static variable in the header:  
**public static Preferences prefs;**

Inside the load method, we have add the following lines of code:

```
// Create (or retrieve existing) preferences file
```

```

prefs = Gdx.app.getPreferences("MMGame");

// Provide default high score of 0
if (!prefs.contains("highScore")) {
    prefs.putInteger("highScore", 0);
}

```

Now we can access the file **prefs** from anywhere in the game! We have created some helper methods so that we don't have to deal with preferences outside of AssetLoader.

```

// Receives an integer and maps it to the String highScore in prefs
public static void setHighScore(int val) {
    prefs.putInteger("highScore", val);

    prefs.flush();
}

// Retrieves the current high score
public static int getHighScore() {
    return prefs.getInteger("highScore");
}

```

Here's the full class again:

```

package com.mentormate.mmgame.mmhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Preferences;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture, logoTexture;
    public static TextureRegion bg;
    public static TextureRegion grass;

    public static Animation logoAnimation;

    public static TextureRegion logo, logoDown, logoUp, playButtonUp,
        playButtonDown, ready, gameOver, highScore, scoreboard, star,
        gameLogo, noStar, retry, firstCandy, secondCandy, mmLogo;

    public static TextureRegion barTopUp, barTopDown, bar;

    public static TextureRegion androidLogo, appleLogo;

    public static Sound dead, flap, coin, fall;

    public static BitmapFont font, shadow, whiteFont;
    private static Preferences prefs;
}

```

```

public static void load() {

    logoTexture = new Texture(Gdx.files.internal("data/logo.png"));
    logoTexture.setFilter(TextureFilter.Linear, TextureFilter.Linear);

    mmLogo = new TextureRegion(logoTexture, 0, 0, 512, 114);

    texture = new Texture(Gdx.files.internal("data/texture.png"));
    texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

    playButtonUp = new TextureRegion(texture, 0, 83, 29, 16);
    playButtonDown = new TextureRegion(texture, 29, 83, 29, 16);
    playButtonUp.flip(false, true);
    playButtonDown.flip(false, true);

    ready = new TextureRegion(texture, 59, 83, 34, 7);
    ready.flip(false, true);

    retry = new TextureRegion(texture, 59, 110, 33, 7);
    retry.flip(false, true);

    gameOver = new TextureRegion(texture, 59, 92, 46, 7);
    gameOver.flip(false, true);

    scoreboard = new TextureRegion(texture, 111, 83, 97, 37);
    scoreboard.flip(false, true);

    star = new TextureRegion(texture, 152, 70, 10, 10);
    noStar = new TextureRegion(texture, 165, 70, 10, 10);

    star.flip(false, true);
    noStar.flip(false, true);

    highScore = new TextureRegion(texture, 59, 101, 48, 7);
    highScore.flip(false, true);

    gameLogo = new TextureRegion(texture, 0, 55, 135, 24);
    gameLogo.flip(false, true);

    bg = new TextureRegion(texture, 0, 0, 136, 43);
    bg.flip(false, true);

    grass = new TextureRegion(texture, 0, 43, 143, 11);
    grass.flip(false, true);

    logoDown = new TextureRegion(texture, 136, 0, 17, 12);
    logoDown.flip(false, true);

    logo = new TextureRegion(texture, 153, 0, 17, 12);
    logo.flip(false, true);

    logoUp = new TextureRegion(texture, 170, 0, 17, 12);
    logoUp.flip(false, true);

    TextureRegion[] birds = { logoDown, logo, logoUp };
    logoAnimation = new Animation(0.06f, birds);
    logoAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

    barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
    // Create by flipping existing skullUp
    barTopDown = new TextureRegion(barTopUp);
    barTopDown.flip(false, true);
}

```

```

        androidLogo = new TextureRegion(texture, 136, 22, 11, 14);
        androidLogo.flip(false, true);

        appleLogo = new TextureRegion(texture, 147, 22, 11, 14);
        appleLogo.flip(false, true);

        bar = new TextureRegion(texture, 136, 16, 22, 3);
        bar.flip(false, true);

        dead = Gdx.audio.newSound(Gdx.files.internal("data/dead.wav"));
        flap = Gdx.audio.newSound(Gdx.files.internal("data/flap.wav"));
        coin = Gdx.audio.newSound(Gdx.files.internal("data/coin.wav"));
        fall = Gdx.audio.newSound(Gdx.files.internal("data/fall.wav"));

        font = new BitmapFont(Gdx.files.internal("data/text.fnt"));
        font.setScale(.25f, -.25f);

        whiteFont = new
BitmapFont(Gdx.files.internal("data/whitetext.fnt"));
        whiteFont.setScale(.1f, -.1f);

        shadow = new BitmapFont(Gdx.files.internal("data/shadow.fnt"));
        shadow.setScale(.25f, -.25f);

        // Create (or retrieve existing) preferences file
        prefs = Gdx.app.getPreferences("MMGame");

        if (!prefs.contains("highScore")) {
            prefs.putInteger("highScore", 0);
        }
    }

    public static void setHighScore(int val) {
        prefs.putInteger("highScore", val);
        prefs.flush();
    }

    public static int getHighScore() {
        return prefs.getInteger("highScore");
    }

    public static void dispose() {
        // We must dispose of the texture when we are finished.
        texture.dispose();

        // Dispose sounds
        dead.dispose();
        flap.dispose();
        coin.dispose();

        font.dispose();
        shadow.dispose();
    }
}

```

Now we can go back to GameWorld and start saving those high scores!

Let's begin by adding a fourth enum constant:

```
public enum GameState {  
    READY, RUNNING, GAMEOVER, HIGHSCORE  
}
```

Now, we just need to add one if statement to where we handle our death (**update method**, when we check for collision between Logo and Ground). We simply check whether our current score is better than the previous high score, and if it is, we update the high score:

```
if (Intersector.overlaps(logo.getBoundingBoxCircle(), ground)) {
    scroller.stop();
    logo.die();
    logo.decelerate();
    currentState = GameState.GAMEOVER;

    if (score > AssetLoader.getHighScore()) {
        AssetLoader.setHighScore(score);
        currentState = GameState.HIGHSCORE;
    }
}
```

Create one more method that we can use to check if we are currently in the HIGHSCORE state:

```
public boolean isHighScore() {
    return currentState == GameState.HIGHSCORE;
}
```

Now we can head to our GameRenderer and handle our new HIGHSCORE gameState, and display the high score at the end of the game:

## Updated GameRenderer:

NOTE: This is temporary code used just for illustration:

## package

```
import
import
import
import
import
import
import
import
import
import
```

```

import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets
    private TextureRegion bg, grass;
    private Animation logoAnimation;
    private TextureRegion logoMid, logoDown, logoUp;
    private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
        bonusLogoApple;

    private Logo bird;
    private ScrollHandler scroller;
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bLogo1, bLogo2;

    private GameWorld myWorld;
    private OrthographicCamera cam;
    private ShapeRenderer shapeRenderer;

    private SpriteBatch batcher;

    private int midPointY;
    private int gameHeight;

    public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
        myWorld = world;

        // The word "this" refers to this instance.
        // We are setting the instance variables' values to be that of the
        // parameters passed in from GameScreen.
        this.gameHeight = gameHeight;
        this.midPointY = midPointY;

        cam = new OrthographicCamera();
        cam.setToOrtho(true, 136, gameHeight);

        batcher = new SpriteBatch();
        batcher.setProjectionMatrix(cam.combined);
        shapeRenderer = new ShapeRenderer();
        shapeRenderer.setProjectionMatrix(cam.combined);

        // Call helper methods to initialize instance variables
        initGameObjects();
        initAssets();
    }

    private void initGameObjects() {
        logo = myWorld.getLogo();
        scroller = myWorld.getScroller();
        frontGrass = scroller.getFrontGrass();
        backGrass = scroller.getBackGrass();
        pipe1 = scroller.getPipe1();
        pipe2 = scroller.getPipe2();
        bLogo1 = scroller.getBonusLogo1();
    }

```



```

        bLogo2 = scroller.getBonusLogo2();
    }

    private void initAssets() {
        bg = AssetLoader.bg;
        grass = AssetLoader.grass;
        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        logoDown = AssetLoader.logoDown;
        logoUp = AssetLoader.logoUp;
        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
            frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
            backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawPipes() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.
        batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
            pipe1.getHeight());
        batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
            pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

        batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
            pipe2.getHeight());
        batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
            pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
    }

    private void drawLogos() {
        // Draw the first logo

```

```

        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
                    bLogo1.getWidth(), bLogo1.getHeight());

        // Draw the second logo
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
                    bLogo2.getWidth(), bLogo2.getHeight());
    }

    public void render(float runTime) {

        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        shapeRenderer.begin(ShapeType.Filled);

        // Draw Background color
        shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
        shapeRenderer.rect(0, 0, 136, midPointY + 66);

        // Draw Grass
        shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 66, 136, 11);

        // Draw Dirt
        shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
        shapeRenderer.rect(0, midPointY + 77, 136, 52);

        shapeRenderer.end();

        batcher.begin();
        batcher.disableBlending();
        batcher.draw(bg, 0, midPointY + 23, 136, 43);

        // 1. Draw Grass
        drawGrass();

        // 2. Draw Pipes
        drawPipes();
        batcher.enableBlending();

        // 3. Draw Bar Tops (requires transparency)
        drawBarTops();

        if (logo.shouldntFlap()) {
            batcher.draw(logoMid, logo.getX(), logo.getY(),
                        logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                        logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
        } else {
            batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                        logo.getY(), logo.getWidth() / 2.0f,
                        logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                        1, 1, logo.getRotation());
        }

        // TEMPORARY CODE! We will fix this section later:

        if (myWorld.isReady()) {
            // Draw shadow first
            AssetLoader.shadow.draw(batcher, "Touch me", (136 / 2) - (42), 76);

```

```

        // Draw text
        AssetLoader.font
            .draw(batcher, "Touch me", (136 / 2) - (42 - 1), 75);
    } else {

        if (myWorld.isGameOver() || myWorld.isHighScore()) {

            if (myWorld.isGameOver()) {
                AssetLoader.shadow.draw(batcher, "Game Over", 25, 56);
                AssetLoader.font.draw(batcher, "Game Over", 24, 55);

                AssetLoader.shadow.draw(batcher, "High Score:", 23, 106);
                AssetLoader.font.draw(batcher, "High Score:", 22, 105);

                String highScore = AssetLoader.getHighScore() + "";

                // Draw shadow first
                AssetLoader.shadow.draw(batcher, highScore, (136 / 2)
                    - (3 * highScore.length()), 128);
                // Draw text
                AssetLoader.font.draw(batcher, highScore, (136 / 2)
                    - (3 * highScore.length() - 1), 127);
            } else {
                AssetLoader.shadow.draw(batcher, "High Score!", 19, 56);
                AssetLoader.font.draw(batcher, "High Score!", 18, 55);
            }

            AssetLoader.shadow.draw(batcher, "Try again?", 23, 76);
            AssetLoader.font.draw(batcher, "Try again?", 24, 75);

            // Convert integer into String
            String score = myWorld.getScore() + "";

            // Draw shadow first
            AssetLoader.shadow.draw(batcher, score,
                (136 / 2) - (3 * score.length()), 12);
            // Draw text
            AssetLoader.font.draw(batcher, score,
                (136 / 2) - (3 * score.length() - 1), 11);

        }

        // Convert integer into String
        String score = myWorld.getScore() + "";

        // Draw shadow first
        AssetLoader.shadow.draw(batcher, "" + myWorld.getScore(), (136 / 2)
            - (3 * score.length()), 12);
        // Draw text
        AssetLoader.font.draw(batcher, "" + myWorld.getScore(), (136 / 2)
            - (3 * score.length() - 1), 11);

    }

    batcher.end();

    batcher.begin();
    drawLogos();
    batcher.end();

}
}

```

The very last thing we need to do is change one line in the **InputHandler** class to this:

```
if (myWorld.isGameOver() || myWorld.isHighScore()) {  
    // Reset all variables, go to GameState.READ  
    myWorld.restart();  
}
```

## Source Code

[DOWNLOAD SOURCE](#)

# Part 11 - Hide the Bonus Logo and add Menus and Tweening

Hidding the bonusLogo is very simple.

In the BonusLogo add private field  
**private boolean visible = true;**

**with;**

```
public boolean isVisible() {  
    return visible;  
}  
  
public void setVisible(boolean visible) {  
    this.visible = visible;  
}
```

In the reset set the boolean to true so every time the logo is reset to be visible again;

```
@Override  
public void reset(float newX) {  
    // Call the reset method in the superclass (Scrollable)  
    super.reset(newX);  
    // Change the height to a random number  
    setVisible(true);  
    position.set(position.x, r.nextInt(90) + 15);  
    setScored(false);  
}
```

Now we have to add the logic for hiding when the bonus is scored.  
In the **ScrollHandler** class we have to change the **scored** method.

```

public boolean scored(Logo logo) {
    if (bonusLogo1.collides(logo)) {
        if (!bonusLogo1.isScored()) {
            bonusLogo1.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            bonusLogo1.setVisible(false);
            return true;
        } else {
            return true;
        }
    } else if (bonusLogo2.collides(logo)) {
        if (!bonusLogo2.isScored()) {
            bonusLogo2.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            bonusLogo2.setVisible(false);
            return true;
        } else {
            return true;
        }
    } else {
        return false;
    }
}

```

And the final touch is a small change in the **GameRenderer** class to do not draw the Texture if it's collected

```

private void drawLogos() {
    // Draw the first logo
    if (bLogo1.isVisible() && logo.isAlive()) {
        batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
            bLogo1.getWidth(), bLogo1.getHeight());
    }

    // Draw the second logo
    if (bLogo2.isVisible() && logo.isAlive()) {
        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
            bLogo2.getWidth(), bLogo2.getHeight());
    }
}

```

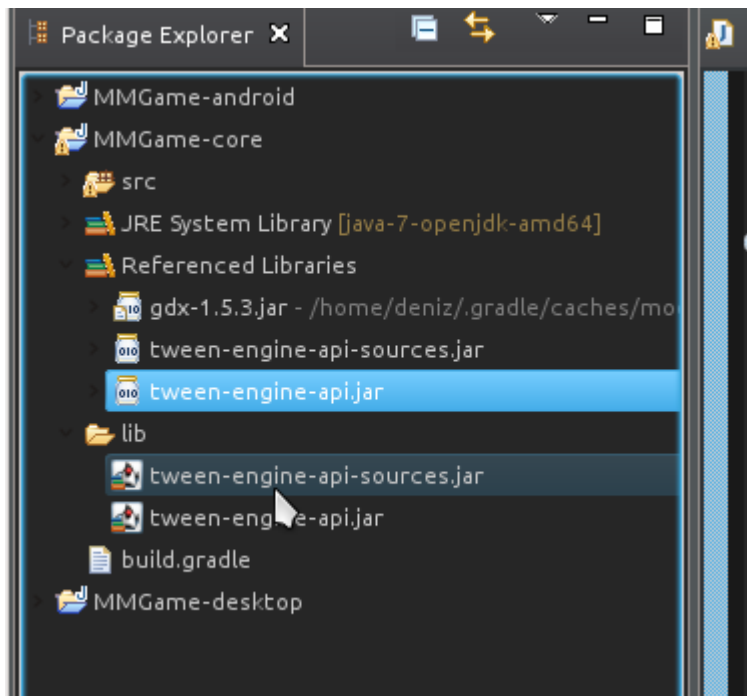
Now you can try the fully playable MMGame  
The only stuff that left is some final UI touches.

## Adding Tween Engine Library to our libGDX projects

Download [those libs](#) and add them into your core

## project lib directory.

Add the libs to project build path from eclipse . If you've setuped the Tween Engine, check your core project in Eclipse. You should see something like this:



## Setting up Android Project

We will now setup our Android Project, so that you can start testing your game on Android devices. Open the **MMGame-android** project.

### 1. Changing the icon:

To change the icon, open the **res > drawable-hdpi** folder. Replace the `ic_launcher.png` with your own image, such as this one: [HERE](#)

### 2. Changing Screen Orientation

The default libGDX screen orientation is landscape. We will be using portrait mode. Open the **AndroidManifest.xml** and change this line:

```
android:screenOrientation="landscape"
to:
android:screenOrientation="portrait"
```

and for the image

```
change:
android:icon="@drawable/ic_launcher"
```

to:

```
android:icon="@drawable/mmgamelogo" //mmgamelogo is the name of the image that
```

you have added in the drawable-hdpi folder.

If there is a problem with running the android project there is a link to the source blow:

[DOWNLOAD LINK](#)

## SimpleButton Object

Create a new package called **com.mentormate.mmgame.ui**. Inside, create a **SimpleButton** class. We will use this for basic UI. Look through the code. It should be mostly self-explanatory.

```
package com.mentormate.mmgame.ui;

import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;

public class SimpleButton {
    private float x, y, width, height;

    private TextureRegion buttonUp;
    private TextureRegion buttonDown;

    private Rectangle bounds;

    private boolean isPressed = false;

    public SimpleButton(float x, float y, float width, float height,
        TextureRegion buttonUp, TextureRegion buttonDown) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.buttonUp = buttonUp;
        this.buttonDown = buttonDown;

        bounds = new Rectangle(x, y, width, height);
    }

    public boolean isClicked(int screenX, int screenY) {
        return bounds.contains(screenX, screenY);
    }

    public void draw(SpriteBatch batcher) {
        if (isPressed) {
            batcher.draw(buttonDown, x, y, width, height);
        } else {
            batcher.draw(buttonUp, x, y, width, height);
        }
    }

    public boolean isTouchDown(int screenX, int screenY) {
```

```

        if (bounds.contains(screenX, screenY)) {
            isPressed = true;
            return true;
        }

        return false;
    }

    public boolean isTouchUp(int screenX, int screenY) {
        // It only counts as a touchUp if the button is in a pressed state.
        if (bounds.contains(screenX, screenY) && isPressed) {
            isPressed = false;
            return true;
        }

        // Whenever a finger is released, we will cancel any presses.
        isPressed = false;
        return false;
    }
}

```

## We have Updated the AssetLoader early

But see it again;

```

package com.mentormate.mmgame.mmhelpers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Preferences;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.Texture.TextureFilter;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.TextureRegion;

public class AssetLoader {

    public static Texture texture, logoTexture;
    public static TextureRegion bg;
    public static TextureRegion grass;

    public static Animation logoAnimation;

    public static TextureRegion logo, logoDown, logoUp, playButtonUp,
        playButtonDown, ready, gameOver, highScore, scoreboard, star,
        gameLogo, noStar, retry, firstCandy, secondCandy, mmLogo;

    public static TextureRegion barTopUp, barTopDown, bar;

    public static TextureRegion androidLogo, appleLogo;

    public static Sound dead, flap, coin, fall;

    public static BitmapFont font, shadow, whiteFont;
}

```



```

private static Preferences prefs;

public static void load() {

    logoTexture = new Texture(Gdx.files.internal("data/logo.png"));
    logoTexture.setFilter(TextureFilter.Linear, TextureFilter.Linear);

    mmLogo = new TextureRegion(logoTexture, 0, 0, 512, 114);

    texture = new Texture(Gdx.files.internal("data/texture.png"));
    texture.setFilter(TextureFilter.Nearest, TextureFilter.Nearest);

    playButtonUp = new TextureRegion(texture, 0, 83, 29, 16);
    playButtonDown = new TextureRegion(texture, 29, 83, 29, 16);
    playButtonUp.flip(false, true);
    playButtonDown.flip(false, true);

    ready = new TextureRegion(texture, 59, 83, 34, 7);
    ready.flip(false, true);

    retry = new TextureRegion(texture, 59, 110, 33, 7);
    retry.flip(false, true);

    gameOver = new TextureRegion(texture, 59, 92, 46, 7);
    gameOver.flip(false, true);

    scoreboard = new TextureRegion(texture, 111, 83, 97, 37);
    scoreboard.flip(false, true);

    star = new TextureRegion(texture, 152, 70, 10, 10);
    noStar = new TextureRegion(texture, 165, 70, 10, 10);

    star.flip(false, true);
    noStar.flip(false, true);

    highScore = new TextureRegion(texture, 59, 101, 48, 7);
    highScore.flip(false, true);

    gameLogo = new TextureRegion(texture, 0, 55, 135, 24);
    gameLogo.flip(false, true);

    bg = new TextureRegion(texture, 0, 0, 136, 43);
    bg.flip(false, true);

    grass = new TextureRegion(texture, 0, 43, 143, 11);
    grass.flip(false, true);

    logoDown = new TextureRegion(texture, 136, 0, 17, 12);
    logoDown.flip(false, true);

    logo = new TextureRegion(texture, 153, 0, 17, 12);
    logo.flip(false, true);

    logoUp = new TextureRegion(texture, 170, 0, 17, 12);
    logoUp.flip(false, true);

    TextureRegion[] birds = { logoDown, logo, logoUp };
    logoAnimation = new Animation(0.06f, birds);
    logoAnimation.setPlayMode(Animation.LOOP_PINGPONG);

    barTopUp = new TextureRegion(texture, 192, 0, 24, 14);
    // Create by flipping existing skullUp
}

```

```

        barTopDown = new TextureRegion(barTopUp);
        barTopDown.flip(false, true);

        androidLogo = new TextureRegion(texture, 136, 22, 11, 14);
        androidLogo.flip(false, true);

        appleLogo = new TextureRegion(texture, 147, 22, 11, 14);
        appleLogo.flip(false, true);

        bar = new TextureRegion(texture, 136, 16, 22, 3);
        bar.flip(false, true);

        dead = Gdx.audio.newSound(Gdx.files.internal("data/dead.wav"));
        flap = Gdx.audio.newSound(Gdx.files.internal("data/flap.wav"));
        coin = Gdx.audio.newSound(Gdx.files.internal("data/coin.wav"));
        fall = Gdx.audio.newSound(Gdx.files.internal("data/fall.wav"));

        font = new BitmapFont(Gdx.files.internal("data/text.fnt"));
        font.setScale(.25f, -.25f);

        whiteFont = new
BitmapFont(Gdx.files.internal("data/whitetext.fnt"));
        whiteFont.setScale(.1f, -.1f);

        shadow = new BitmapFont(Gdx.files.internal("data/shadow.fnt"));
        shadow.setScale(.25f, -.25f);

        // Create (or retrieve existing) preferences file
        prefs = Gdx.app.getPreferences("MMGame");

        if (!prefs.contains("highScore")) {
            prefs.putInteger("highScore", 0);
        }
    }

    public static void setHighScore(int val) {
        prefs.putInteger("highScore", val);
        prefs.flush();
    }

    public static int getHighScore() {
        return prefs.getInteger("highScore");
    }

    public static void dispose() {
        // We must dispose of the texture when we are finished.
        texture.dispose();

        // Dispose sounds
        dead.dispose();
        flap.dispose();
        coin.dispose();

        font.dispose();
        shadow.dispose();
    }
}

```

# The TweenEngine

We have added the Tween Engine library to our project. Let's talk about what it does.

The Tween Engine allows you to mathematically interpolate between a starting value and a final value.

For example, let's say that I have some float called x, with a value of zero. I want to exponentially change this value to a final value of 1 (gradually increase from 0 to 1 at increasing speed). Let's add that I want all of this done in exactly 2.8 seconds.

This is what the Tween Engine enables us to do.

The general implementation of a Tween Engine works like this:

You have a class called Point, for example, which has the following instance variables:

```
float x = 0;
```

```
float y = 0;
```

To use the Tween Engine, to mathematically interpolate these values to x = 1 and y = 5, you need to create a TweenAccessor called PointAccessor.

This class will have two of your own methods. The first method is a getter. It retrieves the values of all the variables that you want to modify from the Point object, and stores it inside an array.

Behind the scenes, the Tween Engine will receive these values and modify them. It will then return the modified values to you in the second method, where you can then send these new values to your Point object.

Let's see an example:

Create a new package called: com.mentormate.mmgame.tweenaccessors and create a SpriteAccessor class:

```
package com.mentormate.mmgame.tweenaccessors;

import aurelienribon.tweenengine.TweenAccessor;
import com.badlogic.gdx.graphics.g2d.Sprite;

public class SpriteAccessor implements TweenAccessor<Sprite> {

    public static final int ALPHA = 1;
```

```

@Override
public int getValues(Sprite target, int tweenType, float[] returnValues) {
    switch (tweenType) {
        case ALPHA:
            returnValues[0] = target.getColor().a;
            return 1;
        default:
            return 0;
    }
}

@Override
public void setValues(Sprite target, int tweenType, float[] newValues) {
    switch (tweenType) {
        case ALPHA:
            target.setColor(1, 1, 1, newValues[0]);
            break;
    }
}
}

```

The class above is an implementation of TweenAccessor, specifically for the Sprite class. As I've previously mentioned, all classes that you want to modify using the TweenEngine need their own Accessor.

The above TweenAccessor only modifies one variable (the alpha, or opacity). If we wanted to modify more variables in different ways, we would create more constants to denote other types of tweening that our Accessor is capable of (such as rotation).

All TweenAccessors need to have two methods: **getValues** and **setValues**, which each target the specific class that you are tweening, in this case the **Sprite** class.

Your role when creating a TweenAccessor is simple. 1. Retrieve the values that you want modified from the Object you are changing, and place them into an array. Let the Tween Engine do its work. 2. Then, you receive the modified values and send them back to the Object you are changing.

**Let's see how the methods above fulfill these roles:**

1. In the **getValues** method, you must retrieve all the values that you want to change from the Sprite object, and store them into the array of floats called returnValues. In this case we are only changing one value, so we can store it into the first index of returnValues: returnValues[0].

Behind the scenes, this value is modified (brought closer to the final value that you have specified elsewhere).

2. After that, these updated values are available to you in the **setValues** method (in the same indices, or positions). Whatever you placed into returnValues[0] will now be available as newValues[0]. You next simply send this value back to the Sprite object.

These methods are called automatically. You only have to provide the initial and final values. These will hopefully make more sense when you see this Accessor in action.

Create a **SplashScreen** class inside **com.mentormate.mmgame.screens**, as shown below.

```

package com.mentormate.mmgame.screens;

import aurelienribon.tweenengine.BaseTween;
import aurelienribon.tweenengine.Tween;

```

```

import aurelienribon.tweenengine.TweenCallback;
import aurelienribon.tweenengine.TweenEquations;
import aurelienribon.tweenengine.TweenManager;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.GL10;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.mentormate.mmgame.MMGame;
import com.mentormate.mmgame.mmhelpers.AssetLoader;
import com.mentormate.mmgame.tweenaccessors.SpriteAccessor;

public class SplashScreen implements Screen {

    private TweenManager manager;
    private SpriteBatch batcher;
    private Sprite sprite;
    private MMGame game;

    public SplashScreen(MMGame game) {
        this.game = game;
    }

    @Override
    public void show() {
        sprite = new Sprite(AssetLoader.mmLogo);
        sprite.setColor(1, 1, 1, 0);

        float width = Gdx.graphics.getWidth();
        float height = Gdx.graphics.getHeight();
        float desiredWidth = width * .7f;
        float scale = desiredWidth / sprite.getWidth();

        sprite.setSize(sprite.getWidth() * scale, sprite.getHeight() *
scale);
        sprite.setPosition((width / 2) - (sprite.getWidth() / 2), (height /
2)
            - (sprite.getHeight() / 2));
        setupTween();
        batcher = new SpriteBatch();
    }

    private void setupTween() {
        Tween.registerAccessor(Sprite.class, new SpriteAccessor());
        manager = new TweenManager();

        TweenCallback cb = new TweenCallback() {
            @Override
            public void onEvent(int type, BaseTween<?> source) {
                game.setScreen(new GameScreen());
            }
        };

        Tween.to(sprite, SpriteAccessor.ALPHA, .8f).target(1)
            .ease(TweenEquations.easeInOutQuad).repeatYoyo(1, .4f)
            .setCallback(cb).setCallbackTriggers(TweenCallback.COMPL
ETE)
            .start(manager);
    }

    @Override

```

```

    public void render(float delta) {
        manager.update(delta);
        Gdx.gl.glClearColor(1, 1, 1, 1);
        Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
        batcher.begin();
        sprite.draw(batcher);
        batcher.end();
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void hide() {
        // TODO Auto-generated method stub
    }

    @Override
    public void pause() {
        // TODO Auto-generated method stub
    }

    @Override
    public void resume() {
        // TODO Auto-generated method stub
    }

    @Override
    public void dispose() {
        // TODO Auto-generated method stub
    }
}

```

## Modifying MMGame

Before we discuss what SplashScreen does, we want SplashScreen, not GameScreen to open when we run the game. So let's open our MMGame class, and make this change:

```

package com.mentormate.mmgame;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.mentormate.mmgame.mmhelpers.AssetLoader;
import com.mentormate.mmgame.screens.SplashScreen;

public class MMGame extends Game {

    @Override
    public void create() {
        Gdx.app.log("MMGame", "created");
        AssetLoader.load();
    }
}

```

```

        setScreen(new SplashScreen(this));
    }

    @Override
    public void dispose() {
        super.dispose();
        AssetLoader.dispose();
    }
}

```

Now go back to the SplashScreen class.

Let's focus specifically on the setupTween method, as the other code is mostly self-explanatory at this point.

```
private Tween.registerAccessor(Sprite.class, new SpriteAccessor());
```

1. This line registers a new Accessor. We are basically saying, "I want to be able to modify Sprite objects using the Tween Engine. Here's the Accessor I made for it that follows your specifications (that we must have a getValues method and a setValues method)."

```
manager = new TweenManager();
```

2. The Tween Engine requires a TweenManager to work, which is updated in our render method with the delta value. This manager will then handle the interpolation for us using our SpriteAccessor.

```

TweenCallback cb = new TweenCallback() {
    @Override
    public void onEvent(int type, BaseTween<?> source) {
        game.setScreen(new GameScreen());
    }
};

```

3. We can create something called a TweenCallback, which is an object whose methods are called when Tweening is complete. In this case, we create a new TweenCallback called cb, whose onEvent method (which will be called when our Tweening is finished), will send us to the GameScreen.

Before we move on to the most important stuff, let's see what we are trying to accomplish. We are trying to take our logo sprite, start it at 0 opacity, increase it to 1 (100%) and bring it back down to zero.

Let's look at this big chunk of code in parts:

```

Tween.to(sprite, SpriteAccessor.ALPHA, .8f).target(1).ease(TweenEquations.easeInOutQuad).repeatYoyo(1, .4f)
.setCallback(cb).setCallbackTriggers(TweenCallback.COMPLETE).start(manager);

```

```
Tween.to(sprite, SpriteAccessor.ALPHA, .8f).target(1)
```

- This says: We are going to tween the sprite object using the SpriteAccessor's ALPHA tweenType. We want this to take .8 seconds. We want you to modify the starting alpha value (this is specified in the SpriteAccessor class) to the desired target value of 1.

```
.ease(TweenEquations.easeInOutQuad).repeatYoyo(1, .4f)
```

- We want this to use a quadratic interpolation (you will see what this does), and repeat once as a Yoyo (with .4 seconds between the repetition). This accomplishes the desired effect of bringing our opacity

back down to zero once it hits one.

```
.setCallback(cb).setCallbackTriggers(TweenCallback.COMPLETE)
```

This then says, use the callback that we have created above called cb, and notify it when tweening is complete.

```
.start(manager);
```

Finally, this specifies which manager will be doing the work.

Now have a look at the render method and see what the manager is doing, and then run the code.

This is probably very confusing! It's hard to understand until you have experimented with it, so before moving on, I encourage you to do some of your own experimentation.

## More TweenAccessors

Now that you know how TweenAccessors work, create two new classes for your com.mentormate.mmgame.tweenaccessors package.

The first class is a simple Value class, which will just be a wrapper class for one float. We create this because only Objects can be tweened (primitives cannot). So, to tween a float, we must create a class for it.

```
package com.mentormate.mmgame.tweenaccessors;

public class Value {

    private float val = 1;

    public float getValue() {
        return val;
    }

    public void setValue(float newVal) {
        val = newVal;
    }
}
```

The **ValueAccessor** class will help us tween the **val** variable in the Value class above:

The ValueAccessor will be used whenever we need to tween a float. For example if we want to flash a screen using a white rectangle with changing opacity, we will create a new Value object and tween it using our ValueAccessor. In fact, we are going to do this to smoothly transition into our GameScreen from the SplashScreen.

We are going to make several changes to our GameScreen, which consists of the GameScreen class and its helpers: InputHandler, GameWorld and GameRenderer classes.

I will only discuss the major changes made to the code. Most of the changes will be easy to grasp, and after a little bit of experimentation, you will understand their purposes.

Start by making changes to the **InputHandler** class:



```

package com.mentormate.mmgame.mmhelpers;

import java.util.ArrayList;
import java.util.List;

import com.badlogic.gdx.Input.Keys;
import com.badlogic.gdx.InputProcessor;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameworld.GameWorld;
import com.mentormate.mmgame.ui.SimpleButton;

public class InputHandler implements InputProcessor {

    private Logo myLogo;
    private GameWorld myWorld;

    private List<SimpleButton> menuButtons;

    private SimpleButton playButton;

    private float scaleFactorX;
    private float scaleFactorY;

    public InputHandler(GameWorld myWorld, float scaleFactorX,
                        float scaleFactorY) {
        this.myWorld = myWorld;
        myLogo = myWorld.getLogo();

        int midPointY = myWorld.getMidPointY();

        this.scaleFactorX = scaleFactorX;
        this.scaleFactorY = scaleFactorY;

        menuButtons = new ArrayList<SimpleButton>();
        playButton = new SimpleButton(
            136 / 2 - (AssetLoader.playButtonUp.getRegionWidth() /
2),
            midPointY + 50, 29, 16, AssetLoader.playButtonUp,
            AssetLoader.playButtonDown);
        menuButtons.add(playButton);
    }

    @Override
    public boolean touchDown(int screenX, int screenY, int pointer, int
button) {
        screenX = scaleX(screenX);
        screenY = scaleY(screenY);
        System.out.println(screenX + " " + screenY);
        if (myWorld.isMenu()) {
            playButton.isTouchDown(screenX, screenY);
        } else if (myWorld.isReady()) {
            myWorld.start();
        }

        myLogo.onClick();

        if (myWorld.isGameOver() || myWorld.isHighScore()) {
            myWorld.restart();
        }

        return true;
    }

```

```

    }

    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button)
    {
        screenX = scaleX(screenX);
        screenY = scaleY(screenY);

        if (myWorld.isMenu()) {
            if (playButton.isTouchUp(screenX, screenY)) {
                myWorld.ready();
                return true;
            }
        }

        return false;
    }

    @Override
    public boolean keyDown(int keycode) {

        // Can now use Space Bar to play the game
        if (keycode == Keys.SPACE) {

            if (myWorld.isMenu()) {
                myWorld.ready();
            } else if (myWorld.isReady()) {
                myWorld.start();
            }

            myLogo.onClick();

            if (myWorld.isGameOver() || myWorld.isHighScore()) {
                myWorld.restart();
            }

        }

        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
        return false;
    }

    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        return false;
    }

    @Override

```

```

    public boolean scrolled(int amount) {
        return false;
    }

    private int scaleX(int screenX) {
        return (int) (screenX / scaleFactorX);
    }

    private int scaleY(int screenY) {
        return (int) (screenY / scaleFactorY);
    }

    public List<SimpleButton> getMenuButtons() {
        return menuButtons;
    }
}

```

The big change to the InputHandler is that we will be generating buttons here. This is not the best way to go about doing this, but as buttons are heavily reliant on input, I have created them here.

The role of the InputHandler is now to create the Buttons and to handle interactions with them, as shown above.

I also created some methods so that we can scale the touches (which are currently dependent on screen size) to our screen size independent game world width and height. Now, touches at x and y will be reported using our GameWorld's coordinates.

I also enabled the use of the Spacebar for those who want to use the keyboard.

To handle the changes to the constructor, we must now go update our **GameScreen** class. The changes here are minor and I will let you review them. :) (Notice the change to the renderer.render() call).

```

package com.mentormate.mmgame.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.mentormate.mmgame.gameworld.GameRenderer;
import com.mentormate.mmgame.gameworld.GameWorld;
import com.mentormate.mmgame.mmhelpers.InputHandler;

public class GameScreen implements Screen {

    private GameWorld world;
    private GameRenderer renderer;
    private float runTime;

    // This is the constructor, not the class declaration
    public GameScreen() {

        float screenWidth = Gdx.graphics.getWidth();
        float screenHeight = Gdx.graphics.getHeight();
        float gameWidth = 136;
        float gameHeight = screenHeight / (screenWidth / gameWidth);

        int midPointY = (int) (gameHeight / 2);

        world = new GameWorld(midPointY);
        Gdx.input.setInputProcessor(new InputHandler(world, screenWidth

```

```

        / gameWidth, screenHeight / gameHeight));
        renderer = new GameRenderer(world, (int) gameHeight, midPointY);
    }

    @Override
    public void render(float delta) {
        runTime += delta;
        world.update(delta);
        renderer.render(delta, runTime);
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void show() {
        Gdx.app.log("GameScreen", "show called");
    }

    @Override
    public void hide() {
        Gdx.app.log("GameScreen", "hide called");
    }

    @Override
    public void pause() {
        Gdx.app.log("GameScreen", "pause called");
    }

    @Override
    public void resume() {
        Gdx.app.log("GameScreen", "resume called");
    }

    @Override
    public void dispose() {
        // Leave blank
    }
}

```

The GameWorld went through some minor changes (dealing with GameState and a few new methods):

```

package com.mentormate.mmgame.gameworld;

import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.math.Rectangle;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class GameWorld {
    private Logo logo;
    private ScrollHandler scroller;
    private boolean isAlive = true;
    private Rectangle ground;
    private int score = 0;
    private float runTime = 0;
}

```

```

private GameState currentState;
public int midPointY;
private GameRenderer renderer;

public enum GameState {

    MENU, READY, RUNNING, GAMEOVER, HIGHSCORE

}

public GameWorld(int midPointY) {
    logo = new Logo(33, midPointY - 5, 17, 12);
    // The grass should start 66 pixels below the midPointY
    scroller = new ScrollHandler(this, midPointY + 66);
    ground = new Rectangle(0, midPointY + 66, 137, 11);
    currentState = GameState.MENU;
    this.midPointY = midPointY;
}

public void update(float delta) {
    runTime += delta;
    switch (currentState) {
        case READY:
            updateReady(delta);
            break;

        case RUNNING:
            updateRunning(delta);
            break;
        default:
            break;
    }
}

private void updateReady(float delta) {
    logo.updateReady(runTime);
    scroller.updateReady(delta);
}

public void updateRunning(float delta) {
    if (delta > .15f) {
        delta = .15f;
    }

    logo.update(delta);
    scroller.update(delta);

    if (scroller.collides(logo) && logo.isAlive()) {
        scroller.stop();
        logo.die();
        AssetLoader.dead.play();
        renderer.prepareTransition(255, 255, 255, .3f);

        AssetLoader.fall.play();
    } else if (scroller.scored(logo) && logo.isAlive()) {
        // TODO add logic to hide candy
        // AssetLoader.coin.play();
    }

    if (Intersector.overlaps(logo.getBoundingBox(), ground)) {

```

```

        if (logo.isAlive()) {
            AssetLoader.dead.play();
            renderer.prepareTransition(255, 255, 255, .3f);

            logo.die();
        }

        scroller.stop();
        logo.decelerate();
        currentState = GameState.GAMEOVER;

        if (score > AssetLoader.getHighScore()) {
            AssetLoader.setHighScore(score);
            currentState = GameState.HIGHSCORE;
        }
    }

}

public Logo getLogo() {
    return logo;
}

public int getMidPointY() {
    return midPointY;
}

public ScrollHandler getScroller() {
    return scroller;
}

public int getScore() {
    return score;
}

public void addScore(int increment) {
    score += increment;
}

public void ready() {
    currentState = GameState.READY;
    renderer.prepareTransition(0, 0, 0, 1f);
}

public void restart() {
    score = 0;
    logo.onRestart(midPointY - 5);
    scroller.onRestart();
    ready();
}

public boolean isReady() {
    return currentState == GameState.READY;
}

public void start() {
    currentState = GameState.RUNNING;
}

public boolean isGameOver() {
    return currentState == GameState.GAMEOVER;
}

```

```

    public boolean isHighScore() {
        return currentState == GameState.HIGHSCORE;
    }

    public boolean isMenu() {
        return currentState == GameState.MENU;
    }

    public boolean isRunning() {
        return currentState == GameState.RUNNING;
    }

    public void setRenderer(GameRenderer renderer) {
        this.renderer = renderer;
    }
}

```

This was accompanied by minor changes in the **Logo** class and the **ScrollHandler** class.

### Changes to Logo class

```

package com.mentormate.mmgame.gameobjects;

import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Vector2;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class Logo {

    private Vector2 position;
    private Vector2 velocity;
    private Vector2 acceleration;

    private float rotation; // For handling bird rotation
    private int width;
    private int height;

    private float originalY;
    private boolean isAlive;

    private Circle boundingCircle;

    public Logo(float x, float y, int width, int height) {
        this.width = width;
        this.height = height;
        this.originalY = y;
        position = new Vector2(x, y);
        velocity = new Vector2(0, 0);
        acceleration = new Vector2(0, 460);
        boundingCircle = new Circle();
        isAlive = true;
    }

    public void update(float delta) {

        velocity.add(acceleration.cpy().scl(delta));

        if (velocity.y > 200) {
            velocity.y = 200;

```

```

    }

    // CEILING CHECK
    if (position.y < -13) {
        position.y = -13;
        velocity.y = 0;
    }

    position.add(velocity.cpy().scl(delta));

    // Set the circle's center to be (9, 6) with respect to the logo.
    // Set the circle's radius to be 6.5f;
    boundingCircle.set(position.x + 9, position.y + 6, 6.5f);

    // Rotate counterclockwise
    if (velocity.y < 0) {
        rotation -= 600 * delta;

        if (rotation < -20) {
            rotation = -20;
        }
    }

    // Rotate clockwise
    if (isFalling() || !isAlive) {
        rotation += 480 * delta;
        if (rotation > 90) {
            rotation = 90;
        }
    }
}

}

public void onRestart(int y) {
    rotation = 0;
    position.y = y;
    velocity.x = 0;
    velocity.y = 0;
    acceleration.x = 0;
    acceleration.y = 460;
    isAlive = true;
}

public void updateReady(float runTime) {
    position.y = 2 * (float) Math.sin(7 * runTime) + originalY;
}

public boolean isFalling() {
    return velocity.y > 110;
}

public boolean shouldntFlap() {
    return velocity.y > 70 || !isAlive;
}

public void onClick() {
    if (isAlive) {
        AssetLoader.flap.play();
        velocity.y = -140;
    }
}
}

```



```

    public void die() {
        isAlive = false;
        velocity.y = 0;
    }

    public void decelerate() {
        acceleration.y = 0;
    }

    public float getX() {
        return position.x;
    }

    public float getY() {
        return position.y;
    }

    public float getWidth() {
        return width;
    }

    public float getHeight() {
        return height;
    }

    public float getRotation() {
        return rotation;
    }

    public Circle getBoundingCircle() {
        return boundingCircle;
    }

    public boolean isAlive() {
        return isAlive;
    }
}

```

## Changes to ScrollHandler class

```

package com.mentormate.mmgame.gameobjects;

import com.mentormate.mmgame.gameworld.GameWorld;
import com.mentormate.mmgame.mmhelpers.AssetLoader;

public class ScrollHandler { // ScrollHandler will create all six objects that
                                // we need.
    private Grass frontGrass, backGrass;
    private Pipe pipe1, pipe2;
    private BonusLogo bonusLogo1, bonusLogo2;
    private GameWorld gameWorld;

    // ScrollHandler will use the constants below to determine
    // how fast we need to scroll and also determine
    // the size of the gap between each pair of pipes.

    // Capital letters are used by convention when naming constants.
    public static final int SCROLL_SPEED = -59;
    public static final int PIPE_GAP = 143;
}

```

```

// Constructor receives a float that tells us where we need to create our
// Grass and Pipe objects.
public ScrollHandler(GameWorld gameWorld, float yPos) {
    this.gameWorld = gameWorld;
    frontGrass = new Grass(0, yPos, 143, 11, SCROLL_SPEED);
    backGrass = new Grass(frontGrass.getTailX(), yPos, 143, 11,
        SCROLL_SPEED);
    pipe1 = new Pipe(210, 0, 22, 60, SCROLL_SPEED, yPos);
    bonusLogo1 = new BonusLogo(pipe1.getTailX() + PIPE_GAP / 2, 50, 16,
11,
        SCROLL_SPEED);
    pipe2 = new Pipe(pipe1.getTailX() + PIPE_GAP, 0, 22, 60,
SCROLL_SPEED,
        yPos);
    bonusLogo2 = new BonusLogo(pipe2.getTailX() + PIPE_GAP / 2, 50, 16,
11,
        SCROLL_SPEED);
}

public void updateReady(float delta) {

    frontGrass.update(delta);
    backGrass.update(delta);

    // Same with grass
    if (frontGrass.isScrolledLeft()) {
        frontGrass.reset(backGrass.getTailX());
    } else if (backGrass.isScrolledLeft()) {
        backGrass.reset(frontGrass.getTailX());
    }

}

public void update(float delta) {
    // Update our objects
    frontGrass.update(delta);
    backGrass.update(delta);
    pipe1.update(delta);
    bonusLogo1.update(delta);
    pipe2.update(delta);
    bonusLogo2.update(delta);

    // Check if any of the pipes are scrolled left,
    // and reset accordingly the bonusLogo
    if (pipe1.isScrolledLeft()) {
        pipe1.reset(pipe2.getTailX() + PIPE_GAP);
        bonusLogo2.reset(pipe2.getTailX() + PIPE_GAP / 2);
    } else if (pipe2.isScrolledLeft()) {
        pipe2.reset(pipe1.getTailX() + PIPE_GAP);
        bonusLogo1.reset(pipe1.getTailX() + PIPE_GAP / 2);
    }

    // Same with grass
    if (frontGrass.isScrolledLeft()) {
        frontGrass.reset(backGrass.getTailX());
    } else if (backGrass.isScrolledLeft()) {
        backGrass.reset(frontGrass.getTailX());
    }
}

```

```

    }

}

public void stop() {
    frontGrass.stop();
    backGrass.stop();
    pipe1.stop();
    pipe2.stop();
    bonusLogo1.stop();
    bonusLogo2.stop();
}

// Return true if ANY pipe hits the logo.
public boolean collides(Logo logo) {
    return (pipe1.collides(logo) || pipe2.collides(logo));
}

public boolean scored(Logo logo) {
    if (bonusLogo1.collides(logo)) {
        if (!bonusLogo1.isScored()) {
            bonusLogo1.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            bonusLogo1.setVisible(false);
            return true;
        } else {
            return true;
        }
    } else if (bonusLogo2.collides(logo)) {
        if (!bonusLogo2.isScored()) {
            bonusLogo2.setScored(true);
            AssetLoader.coin.play();
            addScore(1);
            bonusLogo2.setVisible(false);
            return true;
        } else {
            return true;
        }
    } else {
        return false;
    }
}

}

public void onRestart() {
    frontGrass.onRestart(0, SCROLL_SPEED);
    backGrass.onRestart(frontGrass.getTailX(), SCROLL_SPEED);
    pipe1.onRestart(210, SCROLL_SPEED);
    bonusLogo1.onRestart(pipe1.getTailX() + PIPE_GAP / 2, SCROLL_SPEED);
    pipe2.onRestart(pipe1.getTailX() + PIPE_GAP, SCROLL_SPEED);
    bonusLogo2.onRestart(pipe2.getTailX() + PIPE_GAP / 2, SCROLL_SPEED);
}

// The getters for our five instance variables
public Grass getFrontGrass() {
    return frontGrass;
}

public Grass getBackGrass() {

```

```

        return backGrass;
    }

    public Pipe getPipe1() {
        return pipe1;
    }

    public Pipe getPipe2() {
        return pipe2;
    }

    public BonusLogo getBonusLogo1() {
        return bonusLogo1;
    }

    public BonusLogo getBonusLogo2() {
        return bonusLogo2;
    }

    private void addScore(int increment) {
        gameWorld.addScore(increment);
    }
}

```

The biggest refactoring occurred in the GameRenderer class. However, the changes are minor.

```

package com.mentormate.mmgame.gameworld;

import java.util.List;

import aurelienribon.tweenengine.Tween;
import aurelienribon.tweenengine.TweenEquations;
import aurelienribon.tweenengine.TweenManager;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL10;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.Animation;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer.ShapeType;
import com.mentormate.mmgame.gameobjects.BonusLogo;
import com.mentormate.mmgame.gameobjects.Grass;
import com.mentormate.mmgame.gameobjects.Logo;
import com.mentormate.mmgame.gameobjects.Pipe;
import com.mentormate.mmgame.gameobjects.ScrollHandler;
import com.mentormate.mmgame.mmhelpers.AssetLoader;
import com.mentormate.mmgame.mmhelpers.InputHandler;
import com.mentormate.mmgame.tweenaccessors.Value;
import com.mentormate.mmgame.tweenaccessors.ValueAccessor;
import com.mentormate.mmgame.ui.SimpleButton;

public class GameRenderer {

    // Game Objects
    private Logo logo;

    // Game Assets

```

```

private Animation logoAnimation;
private TextureRegion bg, grass;
private TextureRegion logoMid, mmGameLogo;
private TextureRegion barTopUp, barTopDown, bar, bonusLogoAndroid,
    bonusLogoApple;
private TextureRegion ready, gameOver, highScore, scoreboard, star,
noStar,
    retry;

private ScrollHandler scroller;
private Grass frontGrass, backGrass;
private Pipe pipe1, pipe2;
private BonusLogo bLogo1, bLogo2;

private GameWorld myWorld;
private OrthographicCamera cam;
private ShapeRenderer shapeRenderer;

private SpriteBatch batcher;

private int midPointY;

// Tween stuff
private TweenManager manager;
private Value alpha = new Value();

private Color transitionColor;

// Buttons
private List<SimpleButton> menuButtons;

public GameRenderer(GameWorld world, int gameHeight, int midPointY) {
    myWorld = world;

    // The word "this" refers to this instance.
    // We are setting the instance variables' values to be that of the
    // parameters passed in from GameScreen.

    this.midPointY = midPointY;
    this.menuButtons = ((InputHandler) Gdx.input.getInputProcessor())
        .getMenuButtons();

    cam = new OrthographicCamera();
    cam.setToOrtho(true, 136, gameHeight);

    batcher = new SpriteBatch();
    batcher.setProjectionMatrix(cam.combined);
    shapeRenderer = new ShapeRenderer();
    shapeRenderer.setProjectionMatrix(cam.combined);

    // Call helper methods to initialize instance variables
    initGameObjects();
    initAssets();

    transitionColor = new Color();
    // setupTween
    prepareTransition(255, 255, 255, .5f);
}

private void initGameObjects() {
    logo = myWorld.getLogo();

```

```

        scroller = myWorld.getScroller();
        frontGrass = scroller.getFrontGrass();
        backGrass = scroller.getBackGrass();
        pipe1 = scroller.getPipe1();
        pipe2 = scroller.getPipe2();
        bLogo1 = scroller.getBonusLogo1();
        bLogo2 = scroller.getBonusLogo2();
    }

    private void initAssets() {
        bg = AssetLoader.bg;
        mmGameLogo = AssetLoader.gameLogo;
        grass = AssetLoader.grass;
        logoAnimation = AssetLoader.logoAnimation;
        logoMid = AssetLoader.logo;
        barTopUp = AssetLoader.barTopUp;
        barTopDown = AssetLoader.barTopDown;
        bar = AssetLoader.bar;
        bonusLogoAndroid = AssetLoader.androidLogo;
        bonusLogoApple = AssetLoader.appleLogo;
        highScore = AssetLoader.highScore;
        scoreboard = AssetLoader.scoreboard;
        retry = AssetLoader.retry;
        ready = AssetLoader.ready;
        star = AssetLoader.star;
        noStar = AssetLoader.noStar;
        gameOver = AssetLoader.gameOver;
    }

    private void drawGrass() {
        // Draw the grass
        batcher.draw(grass, frontGrass.getX(), frontGrass.getY(),
            frontGrass.getWidth(), frontGrass.getHeight());
        batcher.draw(grass, backGrass.getX(), backGrass.getY(),
            backGrass.getWidth(), backGrass.getHeight());
    }

    private void drawBarTops() {
        // Temporary code! Sorry about the mess :)
        // We will fix this when we finish the Pipe class.

        batcher.draw(barTopUp, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe1.getX() - 1,
            pipe1.getY() + pipe1.getHeight() + 45, 24, 14);

        batcher.draw(barTopUp, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() - 14, 24, 14);
        batcher.draw(barTopDown, pipe2.getX() - 1,
            pipe2.getY() + pipe2.getHeight() + 45, 24, 14);
    }

    private void drawLogos() {
        // Draw the first logo
        if (bLogo1.isVisible() && logo.isAlive()) {
            batcher.draw(bonusLogoAndroid, bLogo1.getX(), bLogo1.getY(),
                bLogo1.getWidth(), bLogo1.getHeight());
        }

        // Draw the second logo
        if (bLogo2.isVisible() && logo.isAlive()) {

```

```

        batcher.draw(bonusLogoApple, bLogo2.getX(), bLogo2.getY(),
                     bLogo2.getWidth(), bLogo2.getHeight());
    }
}

private void drawPipes() {
    // Temporary code! Sorry about the mess :)
    // We will fix this when we finish the Pipe class.
    batcher.draw(bar, pipe1.getX(), pipe1.getY(), pipe1.getWidth(),
                 pipe1.getHeight());
    batcher.draw(bar, pipe1.getX(), pipe1.getY() + pipe1.getHeight() +
45,
                 pipe1.getWidth(), midPointY + 66 - (pipe1.getHeight() +
45));

    batcher.draw(bar, pipe2.getX(), pipe2.getY(), pipe2.getWidth(),
                 pipe2.getHeight());
    batcher.draw(bar, pipe2.getX(), pipe2.getY() + pipe2.getHeight() +
45,
                 pipe2.getWidth(), midPointY + 66 - (pipe2.getHeight() +
45));
}

private void drawLogoCentered(float runTime) {
    batcher.draw(logoAnimation.getKeyFrame(runTime), 59, logo.getY() -
15,
                 logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                 logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
}

private void drawLogo(float runTime) {
    if (logo.shouldntFlap()) {
        batcher.draw(logoMid, logo.getX(), logo.getY(),
                     logo.getWidth() / 2.0f, logo.getHeight() / 2.0f,
                     logo.getWidth(), logo.getHeight(), 1, 1,
logo.getRotation());
    } else {
        batcher.draw(logoAnimation.getKeyFrame(runTime), logo.getX(),
                     logo.getY(), logo.getWidth() / 2.0f,
                     logo.getHeight() / 2.0f, logo.getWidth(),
logo.getHeight(),
                     1, 1, logo.getRotation());
    }
}

private void drawMenuUI() {
    batcher.draw(mmGameLogo, 136 / 2 - 56, midPointY - 50,
                 mmGameLogo.getRegionWidth() / 1.2f,
                 mmGameLogo.getRegionHeight() / 1.2f);

    for (SimpleButton button : menuButtons) {
        button.draw(batcher);
    }
}

private void drawScoreboard() {

```

```

        batcher.draw(scoreboard, 22, midPointY - 30, 97, 37);

        batcher.draw(noStar, 25, midPointY - 15, 10, 10);
        batcher.draw(noStar, 37, midPointY - 15, 10, 10);
        batcher.draw(noStar, 49, midPointY - 15, 10, 10);
        batcher.draw(noStar, 61, midPointY - 15, 10, 10);
        batcher.draw(noStar, 73, midPointY - 15, 10, 10);

        if (myWorld.getScore() > 2) {
            batcher.draw(star, 73, midPointY - 15, 10, 10);
        }

        if (myWorld.getScore() > 17) {
            batcher.draw(star, 61, midPointY - 15, 10, 10);
        }

        if (myWorld.getScore() > 50) {
            batcher.draw(star, 49, midPointY - 15, 10, 10);
        }

        if (myWorld.getScore() > 80) {
            batcher.draw(star, 37, midPointY - 15, 10, 10);
        }

        if (myWorld.getScore() > 120) {
            batcher.draw(star, 25, midPointY - 15, 10, 10);
        }

        int length = (" + myWorld.getScore()).length();

        AssetLoader.whiteFont.draw(batcher, " + myWorld.getScore(),
            104 - (2 * length), midPointY - 20);

        int length2 = (" + AssetLoader.getHighScore()).length();
        AssetLoader.whiteFont.draw(batcher, " + AssetLoader.getHighScore(),
            104 - (2.5f * length2), midPointY - 3);
    }

    private void drawRetry() {
        batcher.draw(retry, 36, midPointY + 10, 66, 14);
    }

    private void drawReady() {
        batcher.draw(ready, 36, midPointY - 50, 68, 14);
    }

    private void drawGameOver() {
        batcher.draw(gameOver, 24, midPointY - 50, 92, 14);
    }

    private void drawScore() {
        int length = (" + myWorld.getScore()).length();
        AssetLoader.shadow.draw(batcher, " + myWorld.getScore(),
            68 - (3 * length), midPointY - 82);
        AssetLoader.font.draw(batcher, " + myWorld.getScore(),
            68 - (3 * length), midPointY - 83);
    }

    private void drawHighScore() {
        batcher.draw(highScore, 22, midPointY - 50, 96, 14);
    }

```



```

public void render(float delta, float runTime) {

    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

    shapeRenderer.begin(ShapeType.Filled);

    // Draw Background color 108, 180, 226, 0.5f
    shapeRenderer.setColor(108 / 255.0f, 180 / 255.0f, 226 / 255.0f, 1);
    shapeRenderer.rect(0, 0, 136, midPointY + 66);

    // Draw Grass
    shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 66, 136, 11);

    // Draw Dirt
    shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);
    shapeRenderer.rect(0, midPointY + 77, 136, 52);

    shapeRenderer.end();

    batcher.begin();
    batcher.disableBlending();

    batcher.draw(bg, 0, midPointY + 23, 136, 43);

    drawPipes();

    batcher.enableBlending();
    drawBarTops();

    if (myWorld.isRunning()) {

        drawLogo(runTime);
        drawScore();
    } else if (myWorld.isReady()) {
        drawLogo(runTime);
        drawReady();
    } else if (myWorld.isMenu()) {
        drawLogoCentered(runTime);
        drawMenuUI();
    } else if (myWorld.isGameOver()) {
        drawScoreboard();
        drawLogo(runTime);
        drawGameOver();
        drawRetry();
    } else if (myWorld.isHighScore()) {
        drawScoreboard();
        drawLogo(runTime);
        drawHighScore();
        drawRetry();
    }

    drawGrass();

    batcher.end();
    batcher.begin();
    drawLogos();
    batcher.end();

    drawTransition(delta);
}

```

```

    }

    public void prepareTransition(int r, int g, int b, float duration) {
        transitionColor.set(r / 255.0f, g / 255.0f, b / 255.0f, 1);
        alpha.setValue(1);
        Tween.registerAccessor(Value.class, new ValueAccessor());
        manager = new TweenManager();
        Tween.to(alpha, -1, duration).target(0)
            .ease(TweenEquations.easeOutQuad).start(manager);
    }

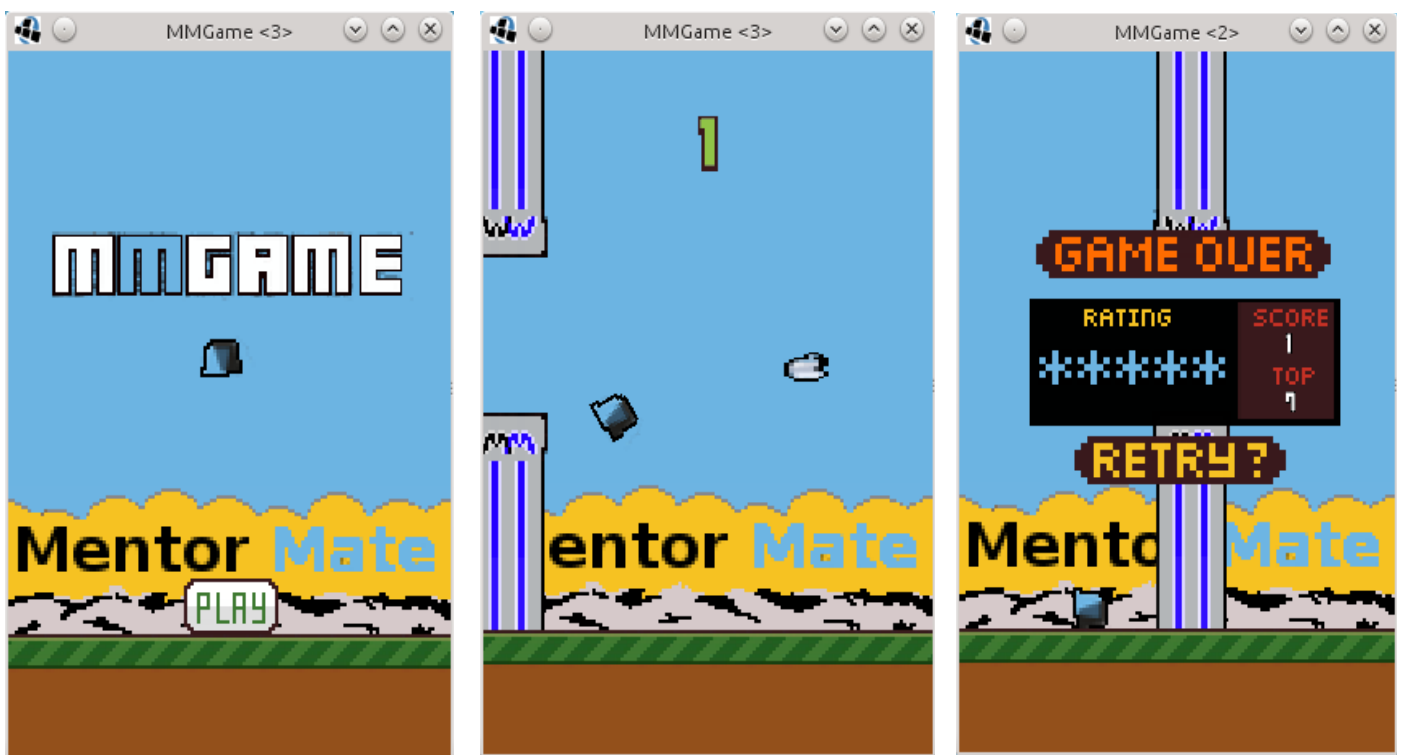
    private void drawTransition(float delta) {
        if (alpha.getValue() > 0) {
            manager.update(delta);
            Gdx.gl.glEnable(GL10.GL_BLEND);
            Gdx.gl.glBlendFunc(GL10.GL_SRC_ALPHA,
GL10.GL_ONE_MINUS_SRC_ALPHA);
            shapeRenderer.begin(ShapeType.Filled);
            shapeRenderer.setColor(transitionColor.r, transitionColor.g,
                transitionColor.b, alpha.getValue());
            shapeRenderer.rect(0, 0, 136, 300);
            shapeRenderer.end();
            Gdx.gl.glDisable(GL10.GL_BLEND);
        }
    }
}

```

## Source Code

[DOWNLOAD LINK](#)

AND FINALLY ENJOY PLAYING MMGAME



**THE END**