Roodle - LibGdx Game

End of Module Report for AA_601

Adam Field

16688

For the latest version of this report please see https://github.com/bishop84/Roodle

Instructions

Go to this GitHub repository - https://github.com/bishop84/Roodle

Download the files (Clone or ZiP)

Import the Roodle, Roodle-Android and Roodle-Desktop into your IDE.

Open MAMP, place the Roodle_PHP folder in your htdocs. Make sure the MAMP root is set to this folder

Pages available are:

localhost:8888/
localhost:8888/?page=getScores
localhost:8888/?page=insertScores&score={score}&user={name}

To run the Android version open and run the MainActivity.java in the Roodle-android folder. Android version uses the phones accelerometer to control Roodle.

To run the Desktop version open and run the Main.java in the Roodle-desktop folder

There are a range of controls but these are the important ones.

A = Left
D = Right
R = Reset
Arrow Keys = Camera Movement
, = Zoom Out
. = Zoom In
/ = Reset Zoom
ESC = Return to main menu

This hard copy is dated 22nd April 2014. For the latest version please see the Git Repository

Introduction

This report is the result of a three month long university module. The objective for this project was to create an Android Application with a remote database connection. For this project I decided to utilise the java framework, LibGdx. The result of the project is a Java game called ‘Roodle’. The Guidelines state that the purpose of this exercise is to ‘get a deeper understanding and experience of the full product lifecycle’.

Figure 01 - LibGdx Framework

Game Overview

The game is a simple endless track game. The background scrolls vertically down the screen giving the impression that the player is moving forward. The player can move about the X axis but is fixed about Y. When playing the desktop version the keys A and D are used to move the player, on Android you simply tilt the phone. As the player ‘moves’ they have to collect items and avoid the obstacles. As the game progresses the speed of the scroll increases. This makes collision with the obstacles an inevitability there by bringing the game to a close.

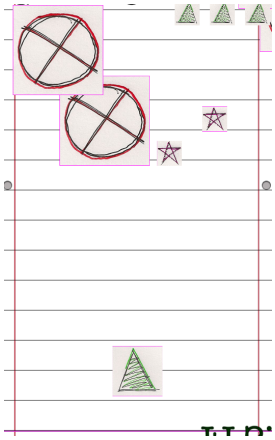

Figure 02 - In game view of Roodle obstacles and stars.

LibGdx FrameWork

The Libgdx Framework prides itself on a ‘write once play anywhere’ structure. Whilst I have found this to be useful it has also created its fair share of problems. Some of which will be explained later on. The core code is written in the Roodle Folder and the other device specific folders act as starter classes to instantiate the game. To run the game you need to select the Main class for the device. (Whilst the game does play on the phone the experience is a little better on the desktop version)

Figure 03 - The Folder Structure for the Roodle Game

Setup GUI and Libraries

There is definitely a steep learning curve to the framework and to be honest I still feel at the bottom of it. The initial stages of the setup were quite confusing and it took a while to get the equivalent of “Hello World” up and running. The framework does come with a setup.jar file that creates the projects for you. Once the files are imported there were other libraries that needed including such as the gdx-tools.jar, json-simple-1.1.1.jar. This involved locating the specific file and copying it into the desktop library folder and then configuring the build path so that the other starter classes can make use of it.

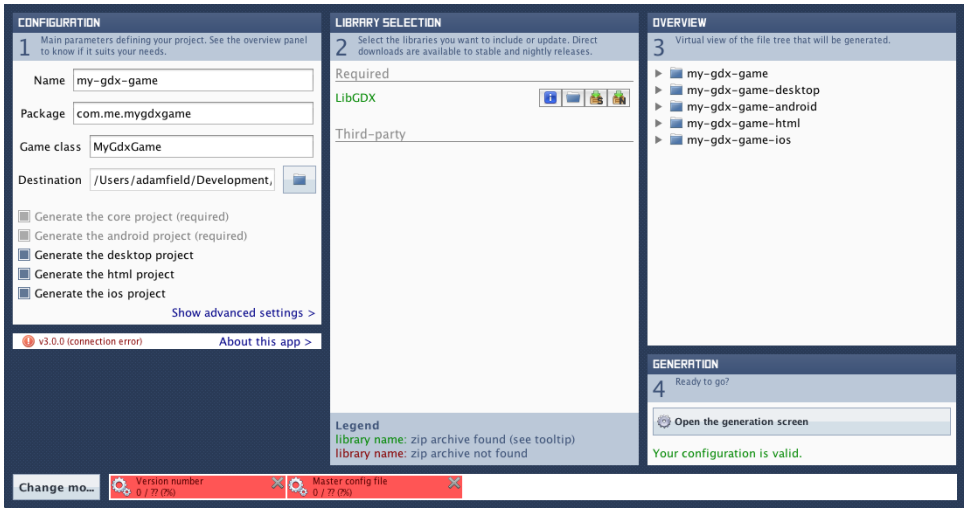

Figure 04 - The Setup GUI for LibGdx

Background Movement

Once the basics were up and running I was able to move the player around but the background scroll wasn’t very good. I was wrapping the background so that it looped over the image however I couldn’t get the two objects to align properly. I eventually changed the logic slightly so that rather than having two background objects I had three. When the background ‘falls’ below a defined limit it repositions up three spaces. This allows for a better transition, it can still glitch now and

then but for the most part it is smooth. I have monitored the distance between them and there is some creep due to the use of deltaTime. This however is minimal and the game would likely be over before a noticeable difference occurred.

```
private void moveBackground(float deltaTime) {
    bG1.velocity.y = deltaTime*-gameSpeed;
    bG2.velocity.y = deltaTime*-gameSpeed;
    bG3.velocity.y = deltaTime*-gameSpeed;


    if(bG1.position.y < -Constants.VIEWPORT_HEIGHT*1.5){
        //bG1.velocity.y = deltaTime*speed;
        bG1.position.y += 3* bG1.dimension.y;
        //System.out.println("BG1 Y: " + bG1.position.y);
    }

    if(bG2.position.y < -Constants.VIEWPORT_HEIGHT*1.5){
        //bG1.velocity.y = deltaTime*speed;
        bG2.position.y += 3* bG2.dimension.y;
        //System.out.println("BG2 Y: " + bG2.position.y);
    }

    if(bG3.position.y < -Constants.VIEWPORT_HEIGHT*1.5){
        //bG1.velocity.y = deltaTime*speed;
        bG3.position.y += 3* bG3.dimension.y;
        //System.out.println("BG3 Y: " + bG3.position.y);
    }
```

Cameras

Roodle uses two cameras, one for the world view, Roodle, Objects and Background and one for the GUI, lives and score. The camera helper class provides some tools that have been really helpful with debugging and positioning things when in the game world. Checking that objects are removed if they fall out of view and checking the background movement wouldn't have been so easy without these. The comma, period and arrow keys are used in the Desktop version to control the movement of the world camera.

I came into some difficulty with regard to using the camera class their positioning and scales. The Gui for example couldn't be connected to the World Camera because the fonts would appear incredibly large , this is because the worlds scale is more comparable to meters than pixels. These cameras could in theory be used to move over the background making the player actually move however in this case the background extends the abstract game object class and as a result has a velocity.

Once the basics were working the next step was to create the Assets. This was another area where I lost considerable time. The main files are stored in the assets-raw folder in the Roodle-desktop folder structure. In this case I have two folders one for the UI and the other for the game. The Main Desktop class uses a TexturePacker class to create what is known as a pack file. This texturepacker rebuilds this atlas every time (The System Console should highlight this process). Sometimes however the texturepacker doesn't create it properly if you are miss managing an asset and so it fails to load properly.

```
io.afield.roodle.Assets.Assets: # of assets loaded: 0
Exception in thread "LWJGL Application" com.badlogic.gdx.utils.GdxRuntimeException: Asset not loaded: images/roodle.pack
        at com.badlogic.gdx.assets.AssetManager.get(AssetManager.java:105)
        at io.afield.roodle.Assets.Assets.init(Assets.java:49)
        at io.afield.roodle.RoodleGame.create(RoodleGame.java:18)
        at com.badlogic.gdx.backends.lwjgl.LwjglApplication.mainLoop(LwjglApplication.java:136)
        at com.badlogic.gdx.backends.lwjgl.LwjglApplication$1.run(LwjglApplication.java:114)
```

Figure 05 - Problems with loading roodle.pack

The asset manager should be capable of manipulating other assets such as sound however time has been a limiting factor yet again. When a new asset region is required you can simply reference it like below.

```
private void init(){

    backgroundRegion1 =
            Assets.instance.background.background;

}
```

Figure 06 - Utilising assets

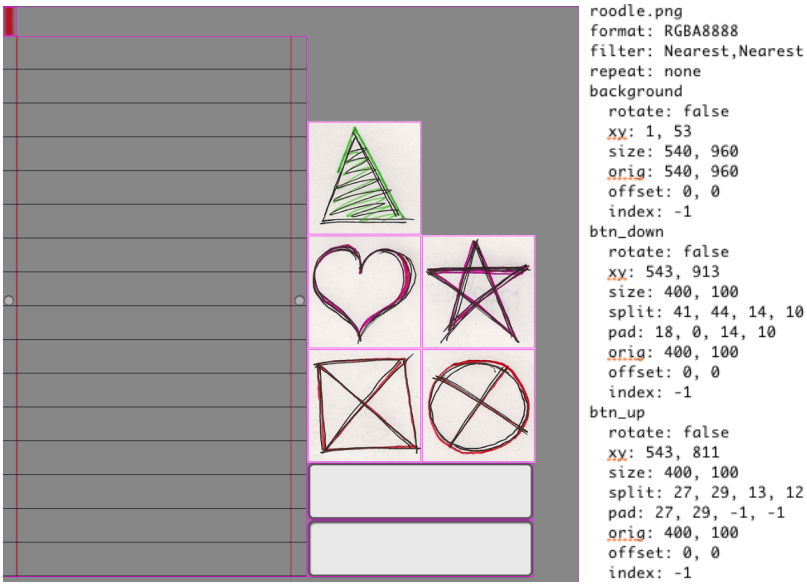

Figure 07 - Pack files

The positioning of things on the screen has been troublesome. Unlike the nice xml used in the android, Libgdx renders the screen. This results in having to position every object quite carefully. I have also found that the origins of objects can cause misalignment, LibGdx defaults the 0,0 to bottom left. The origin needs to be moved in order for the object to rotate around the center. There are multiple places where the width, height, x and y positions, scales etc can be set yet they don't always have an effect, this has made checking and correcting issues to be quite time consuming.

Server

The server side of my project has been an area where I have failed to successfully get the application working properly. I initially created a very simple http request using the LibGdx Net class, this works fine when running the Desktop version but it wouldn't translate onto the Android phone.

There were two problems that most people had, these were;

1. Make sure the permissions in the Android phone were set in the Manifest
2. Make sure the Computers Ip address was used instead of localhost.

Some people also suggested using the phones localhost on 10.0.2.2:8080 however as the server was on my computer this seemed a bit strange to do this. As a result it has meant that a

significant portion of my time has been spent trying to get this to function without success and so I have been unable to handle the results.

I had managed to return the get requests from the server and was printing out the Json arrays in the System Console however I have yet been unable to work with them effectively. The Libgdx has its own way of dealing with Json and I have struggled with the examples available.

This then led me to look into an alternative scenario. This one was slightly more complex and was following this example.

This involves using a JsonClient class with two methods, *sendPost* and *sentRequest.* These take various arguments that should get called in the class where you need to make the request, in the example I use this is done on the ScoresScreen.

```
private void getScores1(){

    Score score = new Score();

    ResponseCallback<Score> callback = new ResponseCallback<Score>() {
        public void onResponse(Score returnObject) {
            System.out.println("Json ok");
        }
        public void onFail(JsonClientException exception) {
            System.out.println("Json request failed: " + exception.getMessage());

        }
    };
    JsonClient.getInstance().sendPost(score, "?page=getScores", callback, Score.class);
}
```

Figure 08 - GetScores Server Code

This JsonClient creates the request and manages the response. The JsonUtil Class would then manage the JSON object to be able to read and write the information.

Recently I have been getting the server status 500 which has also hampered progress. I am unsure as to why this has happened however as this is an internal problem I hope that you will not suffer from the same issues. Any advice on why this might be happening would be greatly recieved.

```
Score
Json request failed: Exception when requesting URL http://localhost:8888/?page=getScores
Message: Received http status: 500
Json request failed: Exception when requesting URL http://localhost:8888/?page=getScores
Message: Received http status: 500
```

Figure 09 - Internal Server Error 500

Conclusion

In it current state the game is not as complete as I would have hoped. The server aspect of this game have caused significant problems that on my own I have been unable to overcome. This is the major downfall of this project but I hope that I have managed to convey some understanding of the fundamentals behind the interaction with servers without getting it fully functional.

I believe that this project unfortunately deviated from its original track. Coding the game aspects along with the associated classes was incredibly time consuming and fraught with problems. This meant that I have failed to give the server aspect enough time. The knock-on effects of this aspect has also lead to other issues with regard to printing out the results from the http requests.

I have no doubt underestimated the difficulty in creating such a project and my own capabilities. It is a great shame not to be able to hand over a more polished game but I do believe that I have tried quite hard to get things working on my own. This project has definitely raised the question of time management. Failing to get the server stuff working sooner has meant that I have had little time to search for solutions to my issues. In future, a broader initial research would provide a way of indicating potential problem areas.

The book that I followed was very comprehensive and the example was incredibly complex. As a result this led to slightly unnecessary additions of things like the Asset Manager and Cameras. Without these the game would still function but these additions are aimed at a much more complex setup than I needed. They are still very useful tools that I can bring to the next project.

I fully intend to continue working with Java and I hope to use it for some aspects of my Major Project. Despite not having a polished product I believe the knowledge I have gained from this process will be useful in my long term learning. As a learning outcome this project has achieved its objectives despite the project not being in a completed state.