

Java Pong game – BSc Computing project Type 4

By Alex Pearce

Student ID apearc03

Word count – 11,842

BSc Computing Project Report, Birkbeck College.

University of London, 2018.

This report is the result of my own work except where explicitly stated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Key.

Book references throughout the report body are structured as "Reference text" [Reference number,"page(s)" page number(s)].

Web references are structured as "Reference text"[Reference number(s)].

Abstract

This written report includes the development timeline of my experience creating Pong with Java. I took an iterative approach and split the project into separate phases. I discovered a lot about project management and my ability to develop software. Planning code before implementation is extremely important and reduces that chance of future problems.

Allocating estimated time slots for each phase/iteration of development is important. It can be easy to spend too long on a particular area and neglect others. The project requirements did change throughout development but I was able to react accordingly and adapt. I believe the end result was successful and reflected my original plans for the project.

Table of Contents

1. Introduction.	5
1.1. Project Overview.....	5
1.2. Design plans and Functionality.	5
1.3. Technologies.	8
1.4. Objectives.....	10
1.5. Software development methodology.....	11
2. Project setup.	13
2.1. LibGDX project creation.	13
2.2. GIT setup.	17
2.3. Initial class structure.	19
3. First iteration of development. Screen creation and application foundation.	20
3.1. The first iteration of code.	20
3.2. The screen functionality unit test.	24
3.3. Executing the initial code.....	25
4. Second iteration. Android emulation, database setup and login screen development.	26
4.1. Android emulation and code refactoring.	26
4.2. Database functionality setup.	27
4.3. Login Screen development.	29
5. Third iteration. Building the game and further code improvements.	32
5.1. Code refactoring.	32
5.2. Game creation.....	32
5.3. Further application improvements and creating the scoring system.....	36
6. Fourth iteration. Building the high scores screen. Adding sound and art to the game.....	38
6.1. Event Logger incorporation.....	38
6.2. The high score screen.	38
6.3. Adding sound and background art to the game	43
7. Fifth iteration. Adding a settings screen and a simple web service.....	45
7.1. Settings screen development.	45
7.2. Web service for database password.....	48
8. Deploying the game.	49
8.1. Finished game diagrams.	49
8.2. Deploying the game to Desktop.	51
8.3. Deploying the game to Android.....	52

9. Conclusion.....	53
9.1. My experience.....	53
9.2. Improvements that could have been made.....	53
9.3. Parts of the project that went well.	54

1. Introduction.

1.1. Project Overview.

For my project I have decided to build my own version of Pong. Pong is a classic arcade video game that was initially released in 1972. It features two paddles moving up and down. The aim of the game is to deflect the ball past the other player. [1]

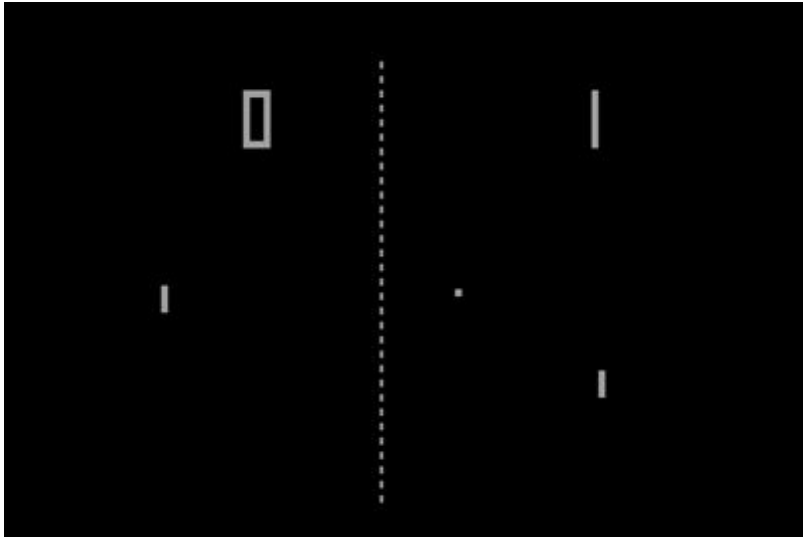


Figure 1. Basic pong game. [2]

I have chosen this as my project because I have an interest in games and their development process. I believe that this is a good choice as it will require the use of technologies and features that are relevant to many areas of software development. The project will also include building a complete application; from initial design to deployment and being fully involved in this process will be great experience. I also like that the game will have lots of room for change and extra functionality, depending on time and the development status. As a final point, I think it will be a fantastic opportunity to showcase lots of the skills and knowledge I have gained throughout my undergraduate degree.

The game will be accompanied by this written report which will be updated throughout each stage of development. The rationale behind the design and software methodologies used will be explained in detail.

1.2. Design plans and Functionality.

The initial functionality of the game I want to implement will consist of a graphical user interface with three screens. The title screen will be loaded on game launch and include a title with two options. The options will be the entry point to the game and high score screen.

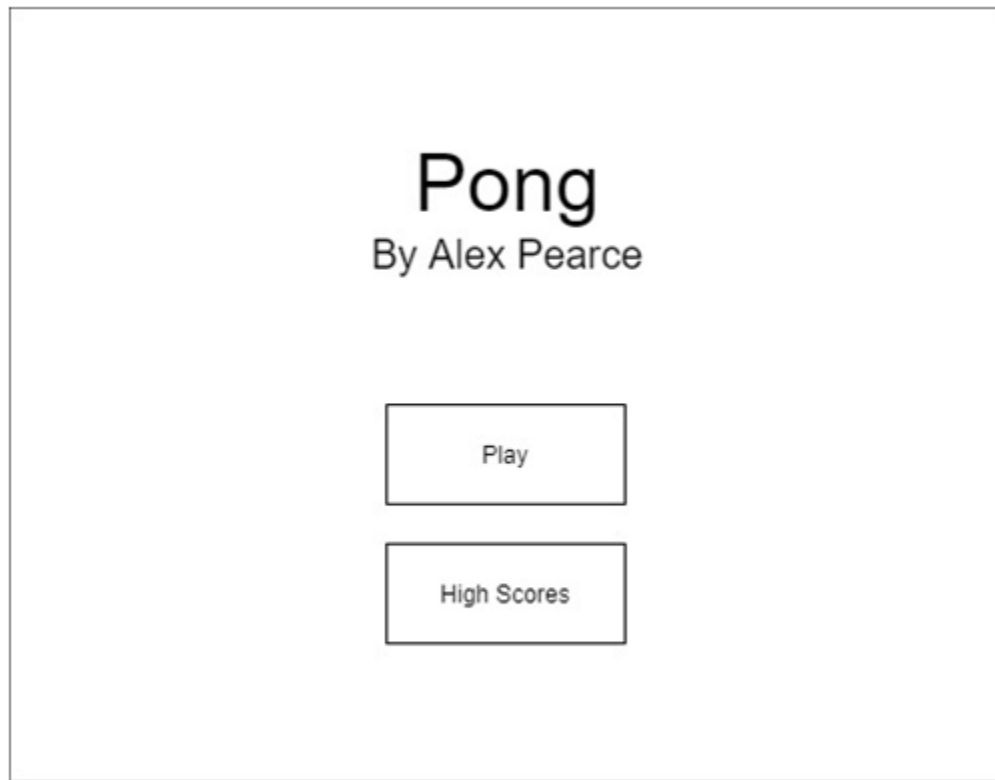


Figure 2. Initial title screen concept.

As the project progresses the title screen may include additional features. A few ideas I have are an option for local two player along with single player. Another useful option to add would be a settings selection to modify the game's resolution and sound levels.

The second screen would be the game, accessed through the play button on the title screen. As a minimum, the core game will include the players on each side with the scores placed at the top of the screen.

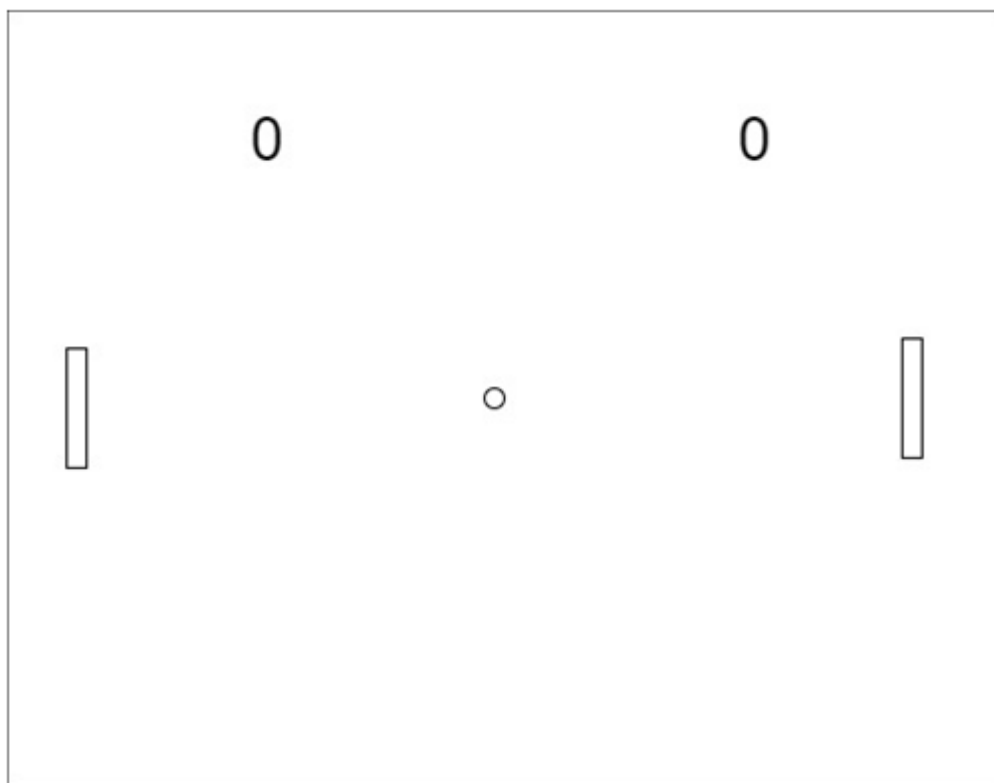


Figure 3. Basic game screen layout.

I will be creating the ball and platform sprites myself. This would allow me to make them a little bit more exciting than just hollow shapes. Also, I would really like to add background music and sounds for scoring or making contact with the ball.

The last fundamental screen will be the high score screen. Scores will be stored in a database after each game session. They will be represented as a series of rankings. Below is a very basic concept of what the high score screen will resemble.

High scores		
Name	Date	Score
Tom	01/07/2017	75
Ben	03/08/2017	72
Jason	05/06/2017	65
David	11/10/2017	58
Bob	25/01/2017	53

Home

Figure 4. High score template screen.

If time permits it, there is definitely room to develop this screen further. Functionalities like displaying a certain number of rankings such as top ten, top fifty or the top one hundred. There will also be the opportunity to only show the scores for the current user.

1.3. Technologies.

I will be using a variety of technologies for this project. I have chosen them carefully based on what I would like to achieve with the application.

I'll be building the game with the Java programming language. I have a lot of experience working with Java and believe it is suitable for this type of project. The benefits of Java include it being open source, extremely well established and cross platform with the use of the Java virtual machine. It is also a very flexible language with lots of libraries and frameworks. I will be using the Eclipse integrated development environment, a fantastic tool for Java development. [22]

For this project I will be using a Java framework called LibGDX [8], one of the more well established Java game development tools. This framework provides helpful tools but also leaves me with a lot of flexibility from a development stand point. LibGDX is specifically built for creating 2D and 3D games. It has lots of customisation for adding physics, graphics, input and audio utilities to a game. Games developed with this framework can be published across many platforms such as Windows, Mac, Android and IOS. Another reason for using LibGDX is the wide availability of documentation and community support. I think that the game will be a lot more polished and smoother to develop than If I were to use the native Java libraries.[11, 11.1, 11.2, 12]

A couple of other game development technologies I considered were:

Lightweight Java Game Library – A low level library that provides access to APIs useful in the development of graphics, audio and parallel computing. This provides the foundations for working with graphics but I would need to develop my own high level utilities for displaying and utilising them. LibGDX also makes use of LWGL as a backend library.

JMonkeyEngine – A fully fledged game development engine. The JME software development kit includes a GUI with many tools and a viewport. The JMonkey engine incorporates many complex tools such as networking utilities, light shaders, graphical effects customisation and game logic controls. With all these features, JME is more suited for computationally intensive 3D game projects rather than my relatively small 2D pong game.

Version control is an important part of software development. Code refactoring can easily lead to bugs and the capability to revert to older versions is vital. For my project I will be using GIT for this purpose. GIT is a popular version control system for tracking changes in computer files using a repository. Commits and pushes are made to alter the repository code. Branches can also be used to work on different areas of a project without affecting the other parts. I want to use GitHub as a remote repository because it means my project will be public and I can receive feedback. Most of these procedures will be performed with GIT bash, a tool used to execute git commands. This will allow me to gain familiarity with both GIT and GitHub, two important technologies in modern software development. [16, 17]

Testing code regularly has proven to be extremely effective when building applications. I will attempt to implement testing solutions throughout the project to help me target errors. This testing will make use of the console within Eclipse and possibly the testing framework, JUnit.

For the high score screen, the plan is to incorporate a database which stores information for each game session that takes place. The initial concept is to record a player's username, date and score once a game has ended. A connection would then be established to a database so the insert query can be performed. The player will be able to access high scores with the use of a select query.

I have past experience using MySQL so that is the database I will be utilising. MySQL is a popular open source relational database management system. The Java development kit includes the Java database connectivity API for database use. Below is an example table with the values I will be storing. [13]

Username(VarChar)	Date(DateTime)	Score(Integer)
Alex	01/10/17 19:30	500

Figure 5. Database table example with attributes and data types.

1.4. Objectives.

With this project I have several goals that I hope to achieve. My main priority is to create a fully functional, professional standard, game application. This means deploying a software product that is of high quality and has gone through suitable testing. I would also like to successfully implement all the technologies I have set out to use. Modern applications regularly use a full stack of different technologies to achieve their desired functionality and this is a great opportunity to replicate that.

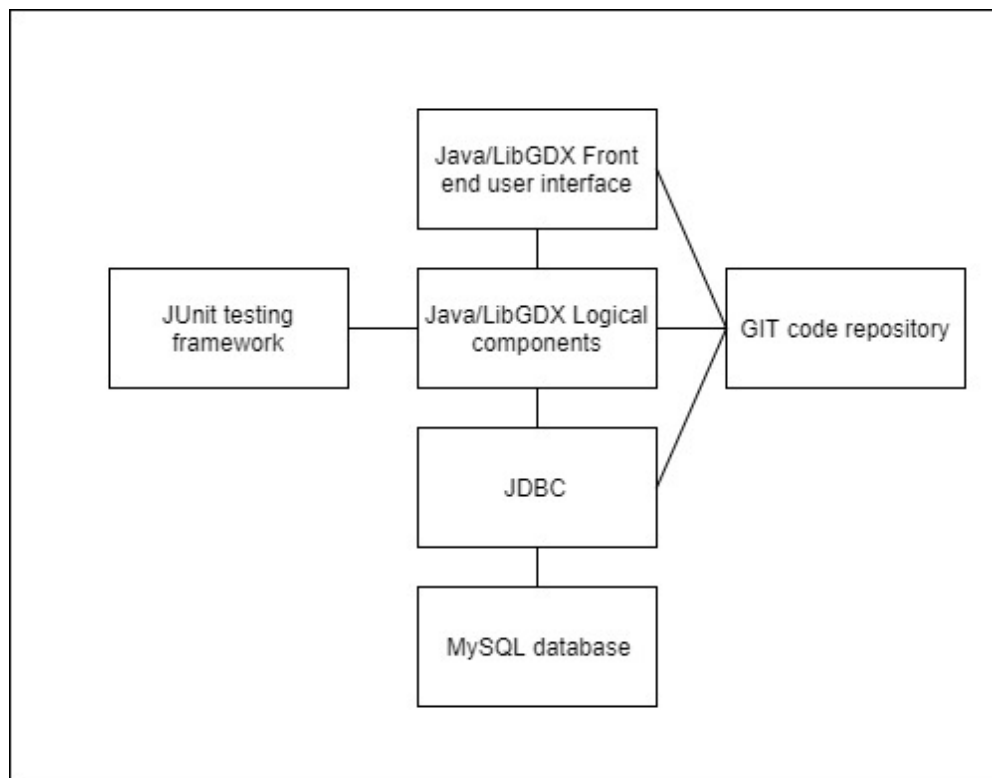


Figure 6. Technology stack.

Another goal is to provide a detailed written report to accompany the game. Relevant documentation is extremely vital when building a software application, it helps others to understand the rationale behind the design of the product. I would certainly have to maintain documentation if I was to enter a working environment in software development.

I would also like to improve my project management ability. This is an application of considerable complexity and separating my time for each phase of development correctly is of high importance. Effective management will lead to a successful project result.

Finally, I aim to further my programming knowledge and skills. Programming is important to me as a hobby and a career. I'll be working with popular open source software such as Java and MySQL. I'll definitely be utilising these technologies in new ways for this project so I hope to learn a lot.

1.5. Software development methodology.

I have chosen to follow the Agile software development methodology for this project. Agile promotes the following values that align well with my project.

Iterative development cycles are encouraged, this works well because the requirements of my project are likely to change and additional features may be added. I can reassess the state of the game at the end of each cycle, update the written report, and plan the targets for the next phase.

Agile projects tend to break products into small working increments. I'd like to implement the game in this fashion with each functionality working independently. This will reduce the chance of potential bugs and make it easier to add new features to the game. The object orientated nature of Java should make this achievable.

Also, Agile promotes unit testing which I definitely want to incorporate into my project. Code refactoring often takes place with this software development methodology and reducing the chance of errors is vital. [4, 4.1]



Figure 7. The iterative style of development that Agile incorporates.

There are alternative development methodologies that could be used for the project but do not have the adequacy that Agile does.

I could approach the project with a traditional waterfall style of development. This would mean separating each process into clearly defined segments, progressing only after completing each phase. The reason I haven't chosen this methodology is because the project functionality will probably change. This will make it very difficult to return to a previously completed area of development and alter the existing implementation.

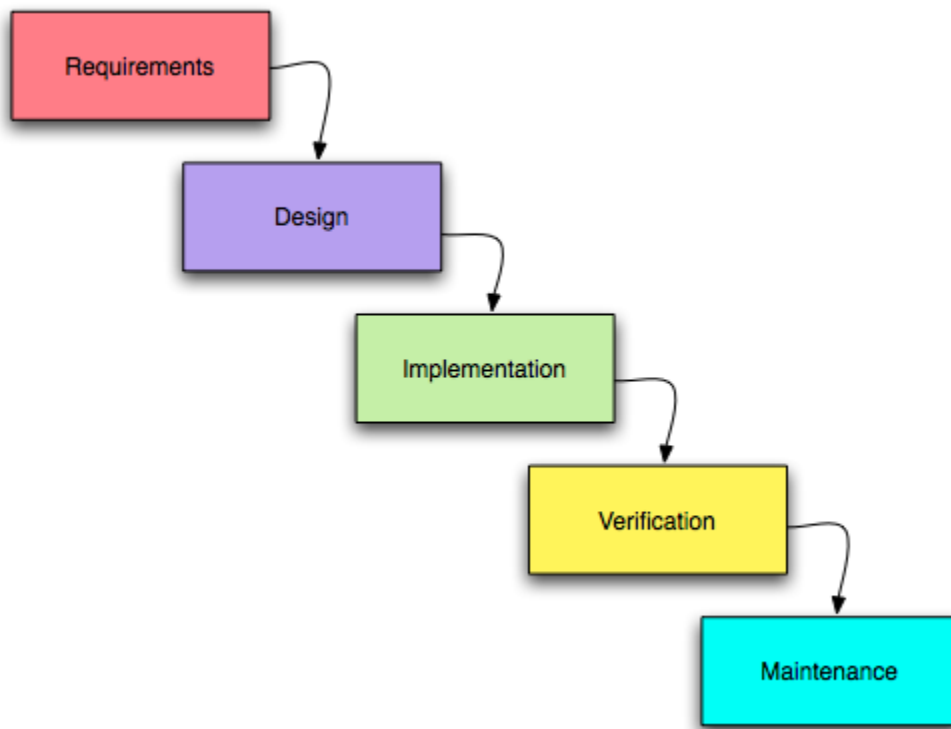


Figure 8. The waterfall development process. Each stage is completed before continuing with production.

2. Project setup.

2.1. LibGDX project creation.

Following the initial planning phase I was now able to begin development. After installing GIT the first step was to generate my project using the LibGDX JAR.



Figure 9. LibGDX project generator.

This tool is designed to simplify the initial build process of the project to include the necessary jar files and libraries. The names of the project, package and game class can be chosen. The target platforms can also be selected as this determines which launcher classes are included. The generator utilizes Gradle, a general purpose software build management system. [10, 10.1,10.2]

Once successfully generated, the foundation files will be located in the selected destination. I then went on to import these files into the Eclipse IDE. Two Java classes were generated as I only chose the desktop platform when I created the project. This is the core game class that was generated. [11.3,11.4] [27, pages 46-51]

```

6  public class pong extends ApplicationAdapter {
7
8
9  @Override
10 public void create () {
11
12 }
13
14 @Override
15 public void render () {
16
17 }
18
19 @Override
20 public void dispose () {
21
22 }
23 }
24

```

Figure 10. Core game class.

It contains three empty overridden methods inherited from the abstract class `ApplicationAdapter`. A game is fundamentally a set of assets that are controlled by game logic. These methods are extremely important for controlling the game logic and determining the game life cycle.

Below is an activity diagram showing the work flow of the game life cycle. The create method is called once when the game starts and will initialise the application, load assets into memory and setup the game screen. The render method is called every frame to update the game screen after handling system events. The resize, resume and pause methods are called when there is appropriate user input. Finally, the dispose method is called when exiting the game. Any open resources should be disposed of. [10.1, 12.1] [27, pages 75-76]

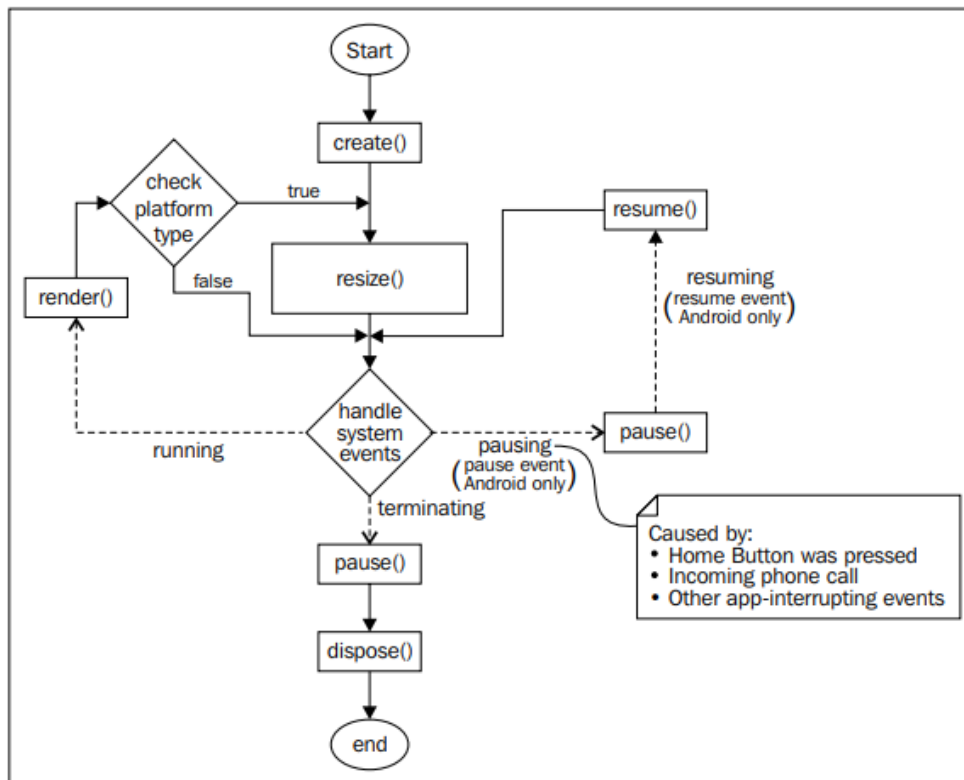


Figure 11. Typical game life cycle.[27, page 75]

The second class to be generated is the desktop launcher. This class contains the main method for the application and is used to launch the game. It contains an object which is an instance of a `LwjglApplicationConfiguration`, a class utilising the lightweight Java game library to display graphics. [12.1]

```

7 public class DesktopLauncher {
8     public static void main (String[] arg) {
9         LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
10        new LwjglApplication(new pong(), config);
11    }
12 }
13 }
14

```

Figure 12. Desktop launcher class.

When the code is run, the following output is produced. An empty window with a title and basic toolbar.

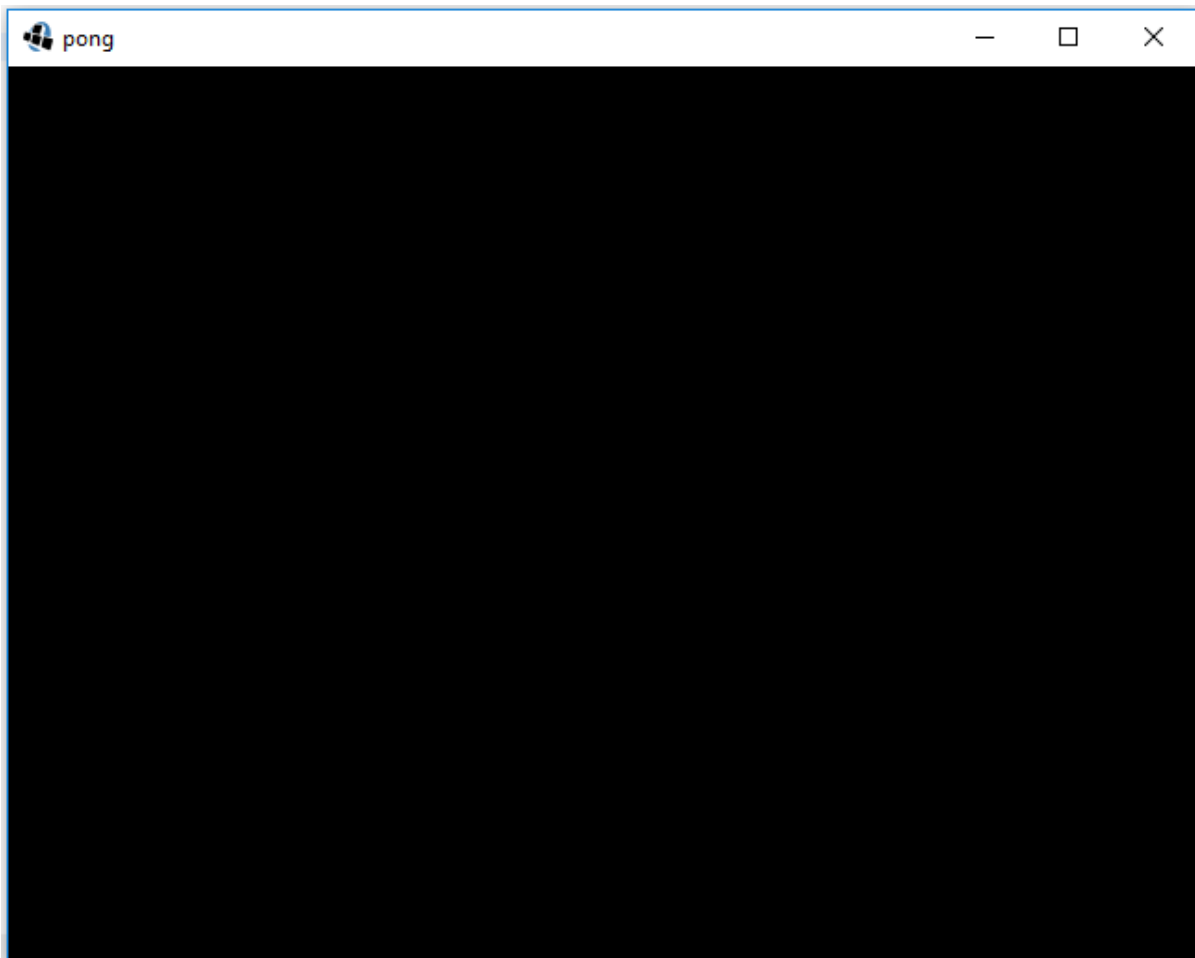
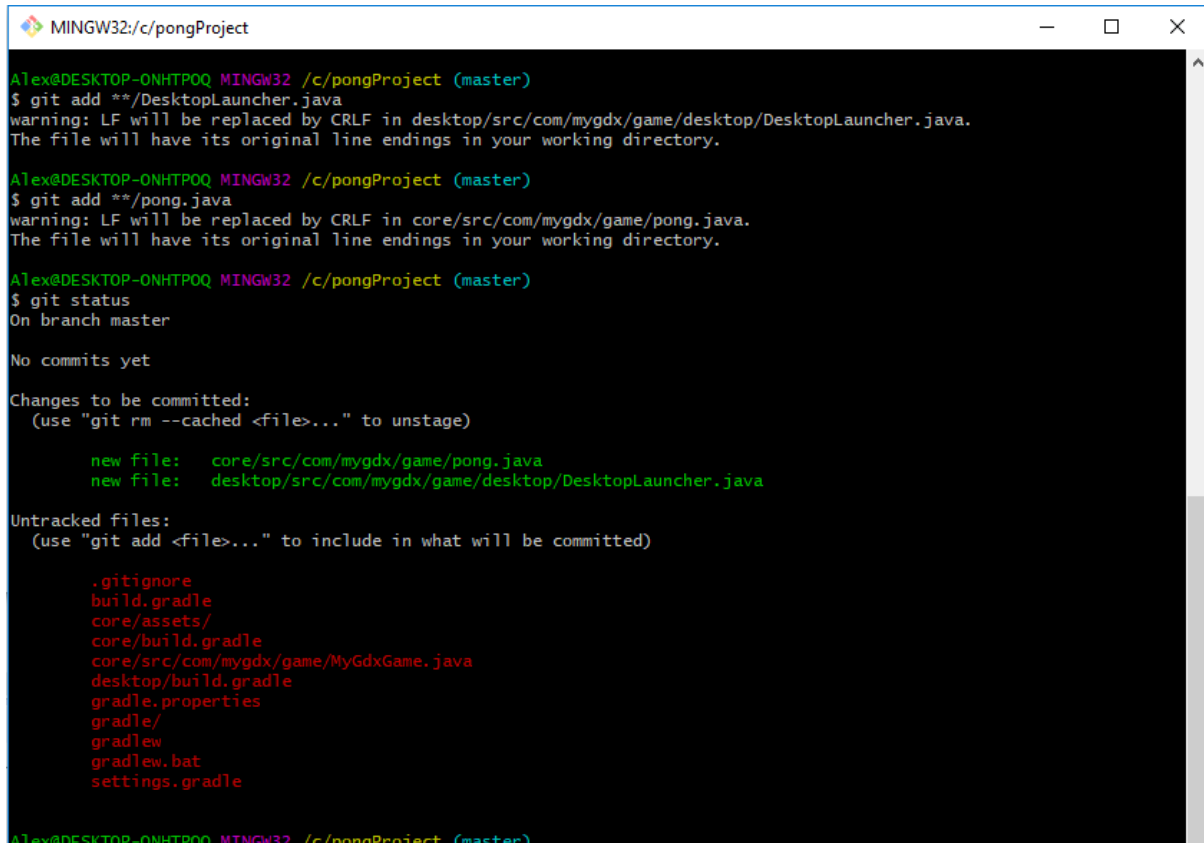


Figure 13. Basic game window.

2.2. GIT setup.

At this point I decided to create a git repository and commit the foundation classes. I did this with the use of GIT bash. Firstly, I created an empty GIT repository in my pongProject folder with the git init command. I then added the two generated Java files to the repository. This was followed by a status command to check the tracked files and a commit to add the files to the repository.



```

MINGW32/c/pongProject

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git add **/DesktopLauncher.java
warning: LF will be replaced by CRLF in desktop/src/com/mygdx/game/desktop/DesktopLauncher.java.
The file will have its original line endings in your working directory.

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git add **/pong.java
warning: LF will be replaced by CRLF in core/src/com/mygdx/game/pong.java.
The file will have its original line endings in your working directory.

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

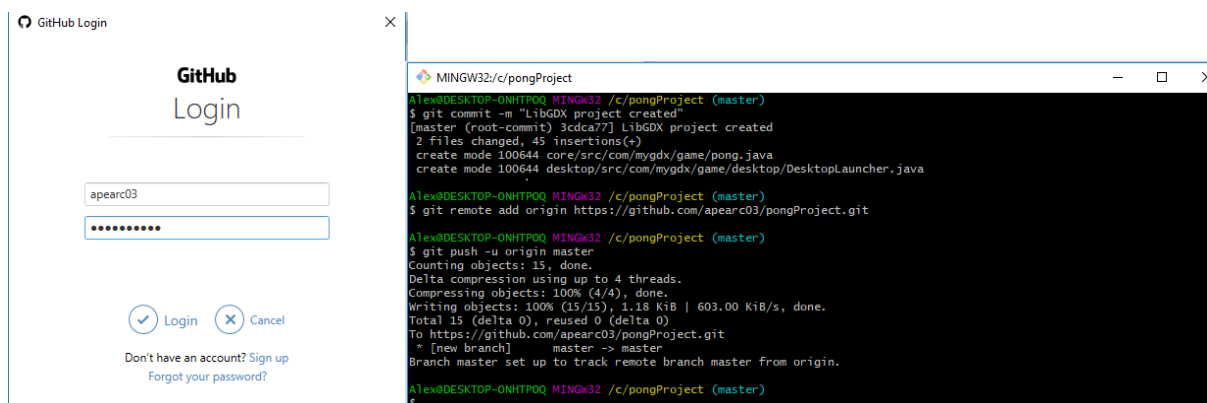
        new file:   core/src/com/mygdx/game/pong.java
        new file:   desktop/src/com/mygdx/game/desktop/DesktopLauncher.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        build.gradle
        core/assets/
        core/build.gradle
        core/src/com/mygdx/game/MyGdxGame.java
        desktop/build.gradle
        gradle.properties
        gradle/
        gradlew
        gradlew.bat
        settings.gradle
  
```

Figure 14. GIT local repository creation.

Following the commit to my local repository, I wanted to also push the project to GitHub. I created a project on the GitHub website and added it as remote repository. The command `git push -u origin master` involves pushing a master branch of my project to the remote name I specified, in this case it is origin. Before the push could complete, I was prompted to log in to my GitHub account. [16.1, 16.2, 16.3, 16.4, 16.5, 17, 18]



GitHub Login

GitHub Login

apearc03

☒ Login
 ☐ Cancel

Don't have an account? Sign up
 Forgot your password?

```

MINGW32/c/pongProject

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git commit -m "LibGDX project created"
[master (root-commit) 3cdca77] LibGDX project created
2 files changed, 45 insertions(+)
create mode 100644 core/src/com/mygdx/game/pong.java
create mode 100644 desktop/src/com/mygdx/game/desktop/DesktopLauncher.java

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git remote add origin https://github.com/apearc03/pongProject.git

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$ git push -u origin master
Counting objects: 15, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (15/15), 1.18 KiB | 603.00 KiB/s, done.
Total 15 (delta 0), reused 0 (delta 0)
To https://github.com/apearc03/pongProject.git
 * [new branch] master -> master
Branch master set up to track remote branch master from origin.

Alex@DESKTOP-ONHTPOQ MINGW32 /c/pongProject (master)
$
  
```

Figure 15. GitHub repository push.

The URL for my GitHub page is:

<https://github.com/apearc03/pongProject>

Now that I have my basic project classes generated and my GIT repositories ready I can begin to delve into the LibGDX framework.

2.3. Initial class structure.

Before implementing code with little planning I wanted to come up with a class structure that I could use for guidance. Having a good idea of which classes I need will allow me to plan out my development phases and workflow. Class diagrams are also used as a layout for a system to visualise how each component interacts with one another.

Below is a basic class diagram I have developed for my Pong game. This will evolve as the project progresses.

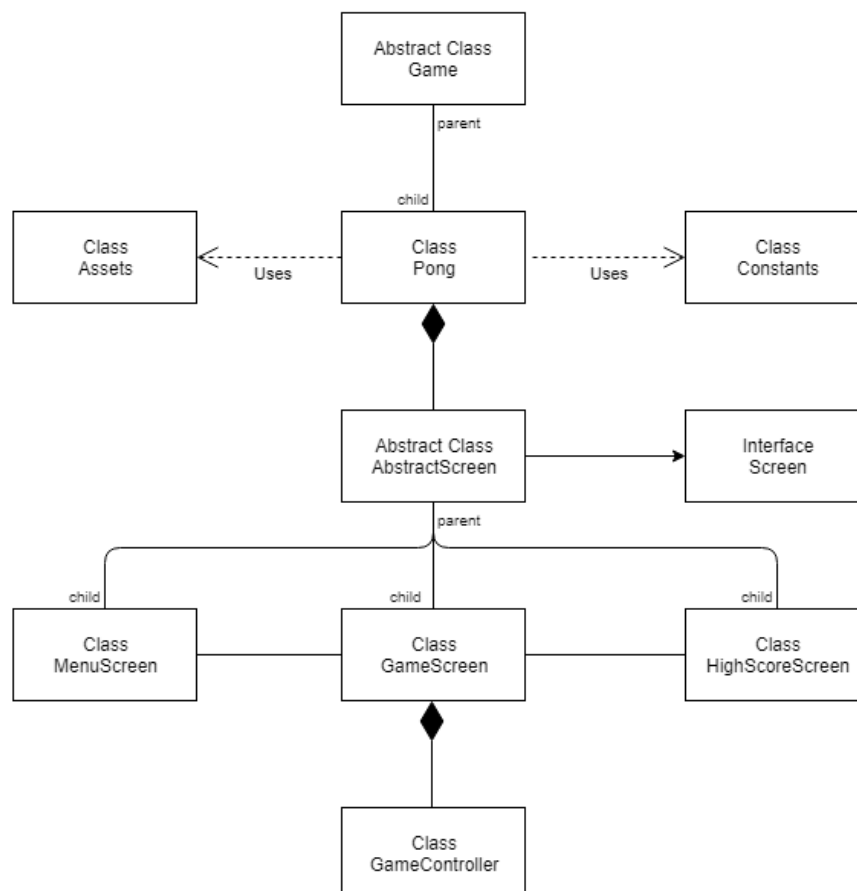


Figure 16. Class structure.

Starting at the top is the abstract class Game which implements the ApplicationListener interface. This class includes all the methods required for the game life cycle, resume, pause, render, resize and dispose. However, It also includes the setScreen method which is used for applications with more than one screen. [27, pages 75-76]

Next is the Pong class which is the main game class that was automatically generated by the LibGDX project creator. This class will extend the Game class and implement all the abstract methods. This is the foundation class of the application and the game life cycle will run from here.

The Constants class will be a handy container for any constant variables I need to reference.

The AbstractScreen class will implement the LibGDX Screen interface to implement the methods for screen functionality. The class will contain an instance variable of type Game. This will be initialized by the constructor and will take an instance of the Pong class. This is useful for calling the setScreen method within the different screen classes. Additionally, this class contains the show and hide methods. Show is called when the screen becomes the currently selected screen. On the other hand, hide is called when the screen is no longer the selected screen. [27, pages 228-230]

AbstractScreen will have three child classes, one for each concrete screen class I intend to implement. The menu, game and high score screens. These classes will contain the logic and input code for each screen. I may need to add an extra class to handle database connectivity for the high score screen.

The GameScreen class is a little different because it will contain the main Pong game for the project. This will make the class more complex so I have decided to split the code up add a separate class for the game logic. GameScreen will be responsible for rendering the game whilst the GameController class will take care of the sprite manipulation.

3. First iteration of development. Screen creation and application foundation.

3.1. The first iteration of code.

The first iteration of development is extremely important for setting up the initial code for the application. If this is done effectively then it makes any further code introduction a lot easier.

I began with creating the templates for all the classes I previously mentioned in my class diagram. This included implementing or extending any necessary LibGDX interfaces/classes. After that I went to code the basic functionality of my game.

This is part of my first implementation of the pong class, which is the starting point of the application. I have included all the initial variables I need.



```

16 |
17 | public class Pong extends Game {
18 |
19 |     private screenFunctionalityTest screenTest; //testing only
20 |
21 |
22 |
23 |     private OrthographicCamera camera;
24 |     private SpriteBatch batch;
25 |     private BitmapFont font;
26 |
27 |     private Skin skin;
28 |
29 |     private MenuScreen menuScreen;
30 |     private GameScreen gameScreen;
31 |     private HighScoreScreen highScoreScreen;
32 |
33 |
34 | public void create () {
35 |
36 |     Gdx.graphics.setTitle(Constants.title);
37 |
38 |     camera = new OrthographicCamera();
39 |
40 |     batch = new SpriteBatch();
41 |
42 |     font = new BitmapFont();
43 |
44 |     skin = new Skin(Gdx.files.internal("uiskin.json"));
45 |
46 |     menuScreen = new MenuScreen(this);
47 |     gameScreen = new GameScreen(this);
48 |     highScoreScreen = new HighScoreScreen(this);
49 |
50 |     screenTest = new screenFunctionalityTest(this); //testing purposes
51 |
52 |
53 |     this.setScreen(menuScreen);
54 | }

```

Figure 17. The main bulk of the pong class. First phase.

The first variable is an instance of screenFunctionalityTest. This is a class I created as a unit test to check that the screen transitions were performing correctly.

A camera is necessary to view the game area. I have used an orthographic camera as it is suitable for representing objects in two dimensions. A Spritebatch is used to draw any textures or sprites to the screen within the render method. BitmapFonts are graphical representations of fonts, with these I can write basic text to the screen. [10.2, 10.4, 12.3]

Scene2d is a library built onto LibGDX. It is great for building and designing an organised screen of interactive widgets. Scene2d uses a stage and actor class system to render objects to the screen. Stages represent the screen whereas actors include all sorts of buttons, text fields and images. Skins are part of the scene2d library and make it easy to style elements of the scene2d library. Skins make use of Json, png, atlas and fnt files to apply settings for the widget appearance. [12.8, 12.9]

I also created one instance of each screen type that I will use. For design purposes I want to only ever use these instances rather than creating a new screen instance each time there is a screen change.

The create method of this class initialises all the variables. This is where the application will begin so I have also called the setScreen method with the menuScreen as a parameter. This means the game will load up to the menu screen.

After that, I began work on the screen classes. By the end of game development these screens and their features will be very different. However, as I am only concerned with the basic functionality for this iteration, I have made the screens similar to test the transitions. Below is the top half of the MenuScreen class which includes the instance variables and the constructor.



```

17
18 public class MenuScreen extends AbstractScreen{
19
20     private Table table;
21     private Stage stage;
22     private Label label;
23     private TextButton gameButton;
24     private TextButton highScoreButton;
25     private boolean isSelected; //testing purposes
26
27     //Constructor initialises stage, table, widgets and input for the screen
28     public MenuScreen(final pong pongGame) {
29         super(pongGame);
30         stage = new Stage(new StretchViewport(Constants.VIEWPORT_WIDTH, Constants.VIEWPORT_HEIGHT, pongGame.getCamera()));
31         table = new Table();
32         stage.addActor(table);
33
34         table.setFillParent(true);
35         table.setWidth(stage.getWidth());
36         table.align(Align.center|Align.bottom);
37
38         label = new Label("Menu Screen", pongGame.getSkin());
39         gameButton = new TextButton("Play", pongGame.getSkin());
40         highScoreButton = new TextButton("High Scores", pongGame.getSkin());
41
42         table.add(label).padBottom(170);
43         table.row();
44         table.add(gameButton).padBottom(50);
45         table.row();
46         table.add(highScoreButton).padBottom(50);
47
48         gameButton.addListener(new ClickListener() {
49             @Override
50             public void clicked(InputEvent event, float x, float y) {
51                 // TODO Auto-generated method stub
52                 super.clicked(event, x, y);
53
54                 pongGame.setScreen(pongGame.getGameScreen());
55             }
56         });
57
58         highScoreButton.addListener(new ClickListener() {
59             @Override
60             public void clicked(InputEvent event, float x, float y) {
61                 // TODO Auto-generated method stub
62                 super.clicked(event, x, y);
63                 pongGame.setScreen(pongGame.getHighScoreScreen());
64             }
65         });
66     }
67 }
68

```

Figure 18. First iteration of the instance variables and constructor within the MenuScreen class.

Similar to the pong class, I created all the variables that I needed at the top of the class. All of these are made private to take advantage of encapsulation. I've then included four scene2d actor variables that I can add to the stage, a Table, Label and two TextButtons. The Table is used as a layout system for a page. Actors can be added to the table and positioned accordingly. Attributes like padding, spacing and rows can be altered. I have used the Label as a screen title and the TextButtons as the entry points for the other two screens. There is also a boolean variable called isSelected which I have used for my screenFunctionalityTest class. [12.8]

The constructor of this class includes a lot of code. It takes an instance of the pong class as a parameter so that I can access the members of that class. The stage is created with a viewport object parameter to allocate the area it will cover. All of the declared variables are initialised and the table is added to the stage. The other actors are added to the table with any styling options specified.

The constructor also contains two calls to the `addListener` method, one for each button. On this occasion, the method takes an anonymous instance of a `ClickListener` object which contains the `clicked` method. This overridden method contains the code that will be executed if the button is clicked. I have called the `setScreen` method from the pong class to transition screens dependent on the button clicked. [10.3, 12.1, 12.4]

Below is the lower half of the `MenuScreen` class. It contains the significant methods for the class. Inside the `show` method I have called the `setInputProcessor` method with the stage as a parameter. This means that all inputs will be directed to the currently active stage object as opposed to input code for other screens. This is followed by two lines of code to assist my `screenFunctionalityTest` class. It calls the `testScreen` method when the screen is shown.

```

69     }
70
71
72
73     //This method is called when the screen is selected
74     @Override
75     public void show() {
76         Gdx.input.setInputProcessor(stage);
77         isSelected = true; //testing purposes
78         pongGame.getScreenTest().testScreens(); //testing
79     }
80     //Method repeatedly called to render and update screen
81     @Override
82     public void render(float delta) {
83         Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
84         stage.act(Gdx.graphics.getDeltaTime());
85         stage.draw();
86         pongGame.getBatch().begin();
87         pongGame.getFont().draw(pongGame.getBatch(), "FPS: " + Gdx.graphics.getFramesPerSecond(), 20, 50);
88         pongGame.getBatch().end();
89     }
90
91     @Override
92     public void resize(int width, int height) {
93         stage.getViewport().update(width, height, false);
94     }
95
96     @Override
97     public void pause() {
98         // TODO Auto-generated method stub
99     }
100
101     @Override
102     public void resume() {
103         // TODO Auto-generated method stub
104     }
105
106     @Override
107     public void hide() {
108         isSelected = false; //testing purposes
109     }
110
111     @Override
112     public void dispose() {
113         stage.dispose();
114     }
115
116     public boolean isSelected() { //testing
117         return isSelected;
118     }
119
120 }
121

```

Figure 19. First iteration of the methods within the `MenuScreen` class.

The `render` method is called many times per second to draw sprites, images and actors to the screen. The first line of code clears the screen so that the application is ready to draw without older objects from previous renders remaining on the screen. Then `act` is

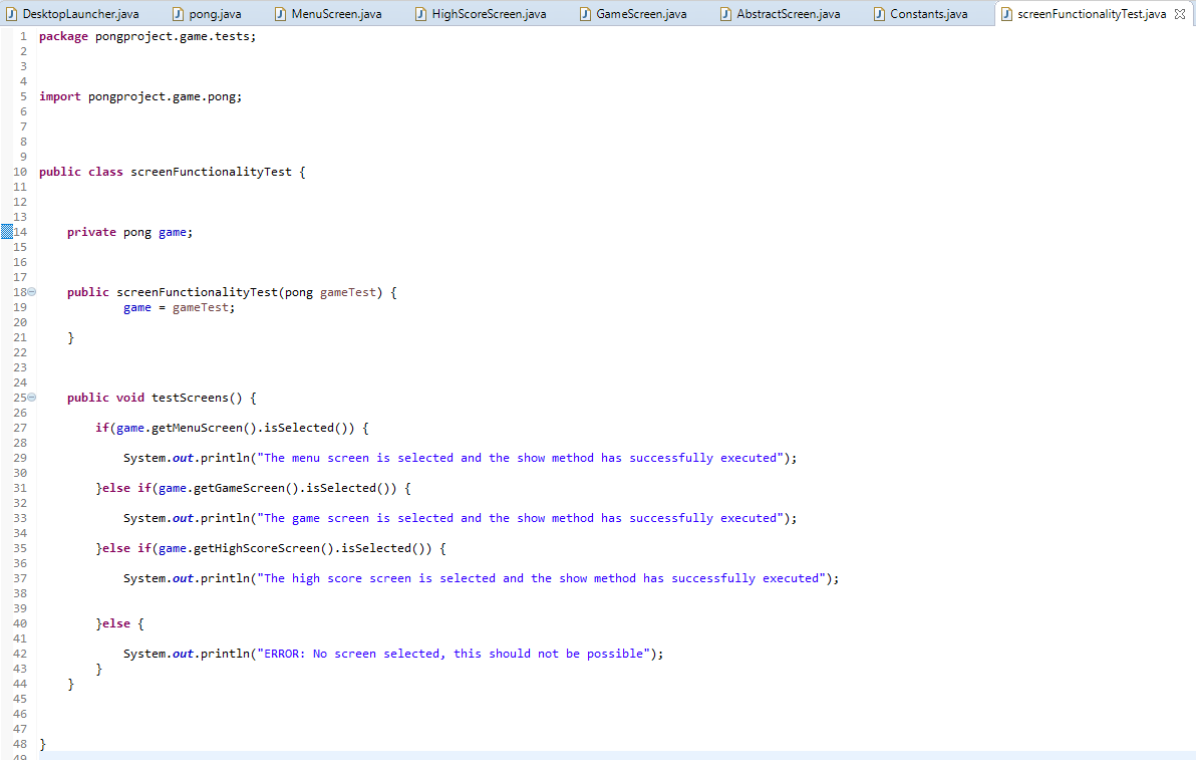
called on the Stage which in turn calls the act method on all it's child actors. The act method will update their position, size or any other dynamic attributes. The actors are then drawn to the screen by the Stage. I also added a line of code to draw the current frames per second to the screen. This is very helpful as it allows me to see the performance of the game. If there are any sudden frames per second drops then I can see which screen or functionality is the cause. [27, page 97] [5.3, 10.2, 11.1]

Beyond the render method there is the resize method. I have added a line of code here which will update the stage according to the newly sized window. This is extremely important to maintain the general structure and ratio of the actors within the stage.

The HighScoreScreen and GameScreen classes are structured in a very similar way at this point of development. However, they contain one button rather than two. They only have the functionality to return to the main menu.

3.2. The screen functionality unit test.

During this iteration I developed a simple unit class for testing the screen transitions. Figure 20 shows my first test class.



```

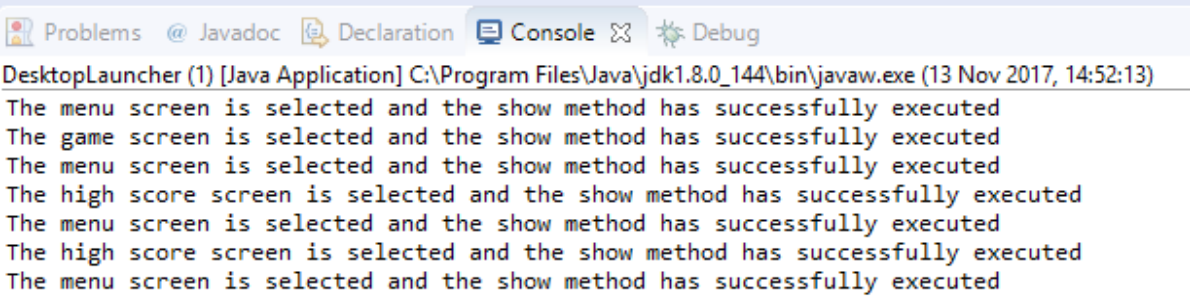
1 package pongproject.game.tests;
2
3
4
5 import pongproject.game.pong;
6
7
8
9
10 public class screenFunctionalityTest {
11
12
13
14     private pong game;
15
16
17
18     public screenFunctionalityTest(pong gameTest) {
19         game = gameTest;
20     }
21
22
23
24
25     public void testScreens() {
26
27         if(game.getMenuScreen().isSelected()) {
28             System.out.println("The menu screen is selected and the show method has successfully executed");
29
30         }else if(game.getGameScreen().isSelected()) {
31             System.out.println("The game screen is selected and the show method has successfully executed");
32
33         }else if(game.getHighScoreScreen().isSelected()) {
34             System.out.println("The high score screen is selected and the show method has successfully executed");
35
36         }else {
37             System.out.println("ERROR: No screen selected, this should not be possible");
38         }
39     }
40 }
41
42
43
44
45
46
47
48
49

```

Figure 20. ScreenFunctionalityTest Class.

This unit test was instantiated within the main pong class. The testScreen method was then called within the show method of each screen. If that screen is selected and the show method executes successfully then a message is printed to the console. The below image shows the method executing after every screen transition. The test did aid me in fixing a bug where multiple screen instances were being created when a

button was clicked. Only a single line should be printed to the console log when a screen transition is made but with the error, the number of lines was incrementing. I was able to realise that I had erroneous code which create additional objects with each button click.

A screenshot of an IDE's console window. The top bar shows tabs for 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Debug'. The 'Console' tab is active. The title bar of the console window reads 'DesktopLauncher (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (13 Nov 2017, 14:52:13)'. The console contains seven lines of text, all stating 'The menu screen is selected and the show method has successfully executed'.

```
DesktopLauncher (1) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (13 Nov 2017, 14:52:13)
The menu screen is selected and the show method has successfully executed
The game screen is selected and the show method has successfully executed
The menu screen is selected and the show method has successfully executed
The high score screen is selected and the show method has successfully executed
The menu screen is selected and the show method has successfully executed
The high score screen is selected and the show method has successfully executed
The menu screen is selected and the show method has successfully executed
```

Figure 21. ScreenFunctionalityTest messages printed to the console.

3.3. Executing the initial code.

It is important to regularly execute the application to manage progress and check for issues. After the current iteration, the game runs with no difficulties and I am happy with the result.

Below is an image containing three separate instances of the game running. They are all resized differently to show the resize method in effect. All the widgets remain in position and are scaled accordingly. This is because of the StretchViewport I specified for my Stage objects. Screen transitions are made by clicking the text buttons. The FPS counter is also shown in the bottom left and is updated every frame. For the moment, I have used a default LibGDX skin for the widgets.

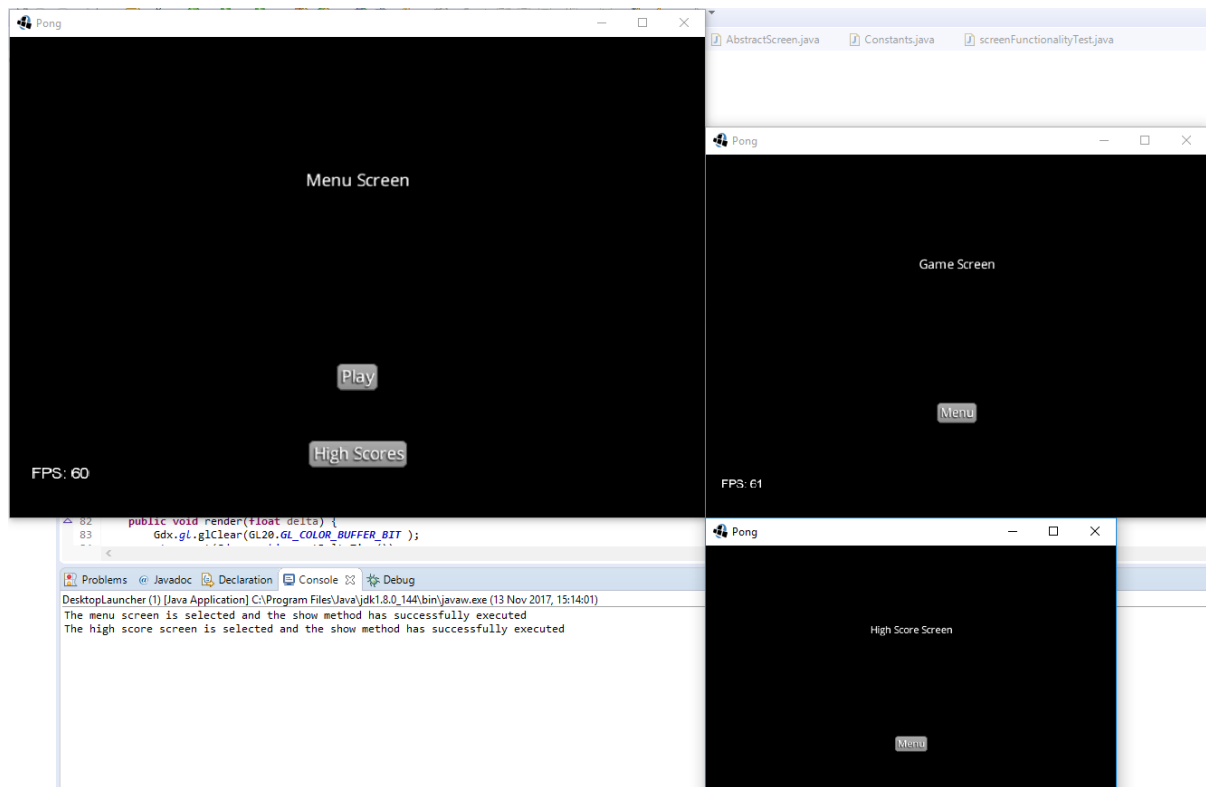


Figure 22. Three instance of the game running after the first iteration of development.

4. Second iteration. Android emulation, database setup and login screen development.

4.1. Android emulation and code refactoring.

For the second iteration of development I decided that it would be a good idea to try and test my application on a platform other than a desktop environment. This would allow me to see if the game would operate differently and if there were any obvious bugs. I began by adding the Android, IOS and html launchers to my project. I am not certain that the final program will be available for each of these platforms but my goal is to include at least two.

I wanted to attempt launching the game on Android with an emulator rather than using a physical device. This allowed me to designate the android software development kit version, screen resolution and processor type. I also wanted to run the emulator with android studio as opposed to eclipse. This was just to test that the code executed

properly in a different development environment. After adequate setup, the application ran smoothly on the android emulator. [27, pages 21-29, 79]

Later on however, I discovered that accessing a MySQL database with native Android through the JDBC is troublesome and usually requires a web service for database operations. This will definitely affect my decision to deploy the final game to android or not. [6.1]

I followed with a little code refactoring. I removed the abstractScreen class as it was not utilized very well. There just wasn't enough common code between the child classes to warrant the class. I also found that implementing variables within the child classes helped me to keep better track of the game life cycle. Each screen class I created now implements the LibGDX Screen interface. I capitalized the first letter of the pong game class to Pong. Another change I made was to remove the table scene2d class from my screens. I chose to do this because the positioning of widgets became increasingly difficult as I added more to the screen. The screen classes now contain the original stage instance variable. All scene2d actors are now added to the stage instead of the table. I found this change effective and I could position my actor objects appropriately. These changes were then committed and pushed to GitHub.

4.2. Database functionality setup.

The next big part of development was to include database functionality. The database will be used to record the names and scores of players. My plan to make this work was to propose a small login system for the application. This would simply include a username and password. This would prevent multiple users with the same username clashing on the high score screen.

I needed to plan the database schema I would using for the project. I plan to use two tables, pong_users and pong_scores. A diagram representing the schema is below. I will be hashing the passwords for security before storing them in the database. The primary key for pong_users is the username and password. For pong_scores, it is the username, date and score. The username in pong_users is also the foreign key for the username field in pong_scores. This prevents the possibility that an unregistered user can be entered into the pong_scores table.

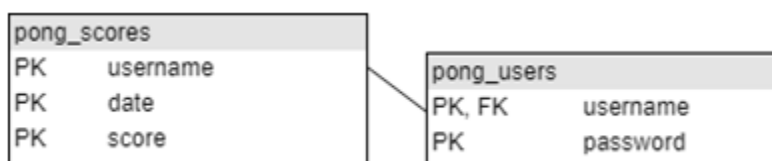


Figure 23. The database table schema.

Later on, I had to decide how I would host my database. The options were to use one of my own machines at home to host the server, use my Birkbeck database or find a hosting service. Some considerations were security, consistency of database uptime and internet protocol configurations. I settled on using Amazon web services. They offer the Amazon RDS service which is ideal for hosting a relational database server. There's also a free tier which meets all my needs for this project. From here, I would use the AWS console in conjunction with MySQL workbench to manage my database. [3, 5.2, 15, 15.1, 15.2]

From this point, I could attempt to access the database from my application. I created my own databaseManager class which would include methods for creating and closing connections to the database server. The connection was made using a static method in the DriverManager class in the JDBC library. The getConnection method takes the login details for an account registered to the server and then returns a Connection object relative to the database. It is very bad practice to include these details in the source code as the information can be traced from the compiled code and attacks can be made on the database. As the game is a local desktop application I had to connect to the database in the source code rather than on a secure remote server. Therefore I attempted to obfuscate the internal login details as much as I could. This was by no means a solution but it is better than nothing. I used a separate class for the credentials and encoding. The databaseManager also contains a method to check if the connection was still alive and another to close the connection. [9] [26, pages 105-106]

The connection worked and then it was on to implementing functionality to the game. I needed to decide on which point in the program I attempt to connect to the database. A couple of ideas were to connect only on start up or attempt to connect every time the player accesses the menu screen. I tried several options but I noticed that attempting to connect to a database is a resource intensive task, especially if no connection is made. It wasn't a big issue but I noticed that the application hangs for a few seconds whilst it attempts to connect and I wanted to avoid this. I tried to create a solution using the setLoginTimeout method within the DriverManager class to avoid the lengthy wait to connect to the database. This seemed to have minimal effectiveness.

I wanted to minimize the amount of times an attempt is made so I came up with a solution. The game would attempt to make a connection on start up. If that fails then the program loads to the menu and the game can be played without connection but no score is recorded. The high score screen button is also disabled. I added a retry connection button to the menu screen that can be used to attempt connection whenever clicked. This allows the player to manually choose when to reconnect. The connection status is checked in the show method of the menu screen. This was a way of continuously checking the connection which means that I can display or hide the retry connection button depending on the result. If the retry button is clicked and a connection is made then the retry button is made invisible and the high score screen

button is made visible. Below are images of the program working with and without connection to the database.

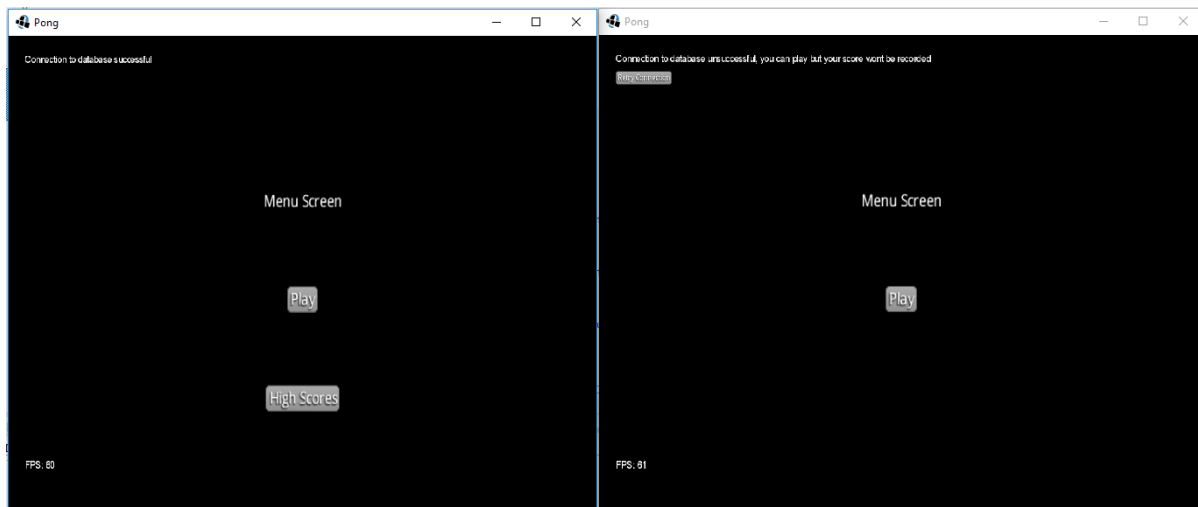


Figure 24. Two instances of the game executed. One with the database running and one with the database stopped.

4.3. Login Screen development.

The next step was developing a way for a user to login or create an account. I decided to add a loginScreen class as my menuScreen class was becoming large. I then added code to the clickListeners of the play button in the menuScreen so that the loginScreen was selected if there was a database connection. If no connection was present then the play button would skip the login and access game screen directly.

The loginScreen needed two user input textFields for the username and password. When these fields are filled with values, a play button can be clicked to call validation methods. ValidateUsername and ValidatePassword are the two methods within the login class, they assert that the entered values follow certain rules. A username and password must be between four and ten characters and can only contain numbers and letters, no other characters are valid. This is achieved with the use of regular expression pattern matching. Also, the password cannot equal the username. This process helps to make the program more efficient as it asserts valid inputs before methods are called which access the database. Below are some of the error combinations that can occur. [27, pages 235-236] [12.8]

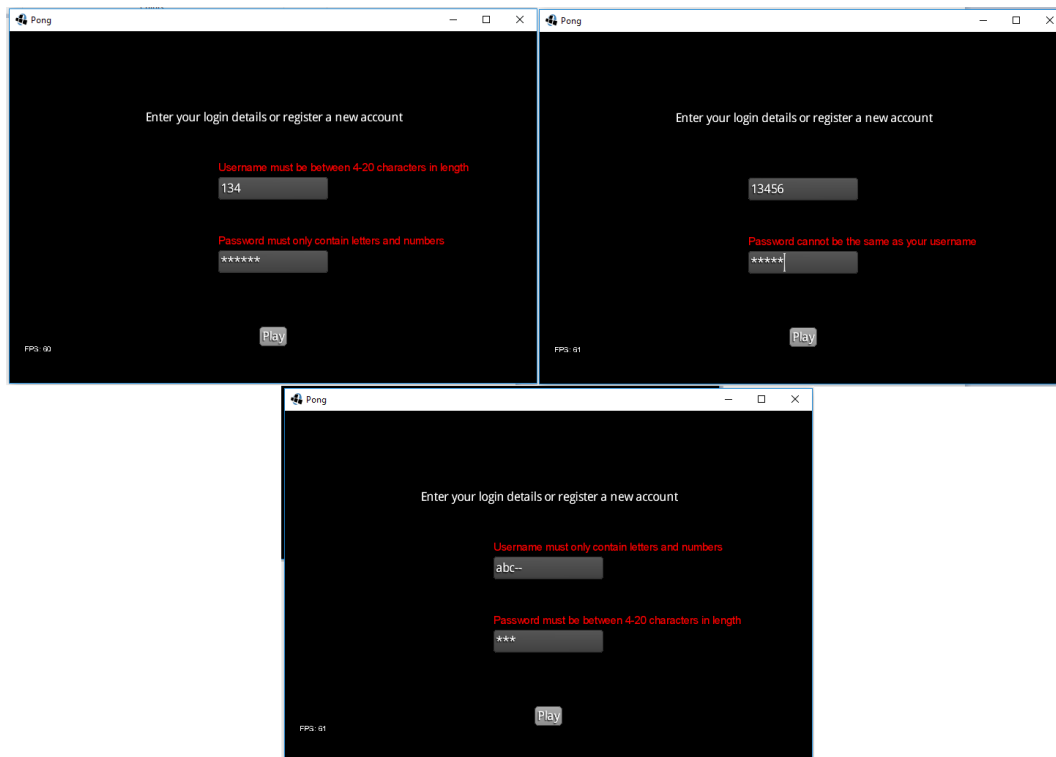


Figure 25. The different possible validation errors.

Once a valid username and password have been entered, the play button will access the methods within the `databaseManager` class to verify the details against the database. If there is no connection to the database then the user is returned to menu screen where they can attempt to reconnect or play offline. Within the `databaseManager`, `checkLogin` is called which takes the validated username and password and performs a select all query to the `pong_users` table. This method returns a boolean which determines whether the next screen is selected.

If the username and the hashcode of the password both match entries inside the table, then this is an existing user account and `true` is returned. If the username matches but the password doesn't then this could be an example of trying to register with an existing username or a login with the wrong password. `False` is returned and an error message is displayed. The final outcome is that a username and password are not found within the table and in that case, the `addAccount` method is called which inserts the new login details to the table. `True` is returned and the game will progress to the next screen.

I needed to implement testing to check the results. I made several adjustments to the code so that I could see the execution flow of the program. I tested the login system for three different scenarios. An incorrect login attempt, an existing user and a new user. Below is a screenshot of the console throughout the three separate login attempts.

```

DesktopLauncher (3) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (24 Nov 2017, 01:47:52)
The menu screen is selected and the show method has successfully executed
The log in screen is selected and the show method has successfully executed
Username matches but not password, an error message has been returned
Existing user
LOGIN SUCCESS
The game screen is selected and the show method has successfully executed
The menu screen is selected and the show method has successfully executed
The log in screen is selected and the show method has successfully executed
Values inserted
New user
LOGIN SUCCESS
The game screen is selected and the show method has successfully executed

```

Figure 26. Console printout during login attempts.

Along with using the console, I also verified the results through MySQL Workbench. I was able to query the database and check whether the login entries were successful. Below are two images. The first query was to retrieve all the entries to the pong_users tables. This was helpful to determine if the login phase of my application had correctly altered the database. The second query doesn't concern the login process but was very useful to check the number of connections to my database. I was able to test multiple clients to see how simultaneous database interaction worked. It was also useful to check if I had any unknown clients connected to my database. The testing was followed by a GitHub commit so that I was ready to move on with development. [14]

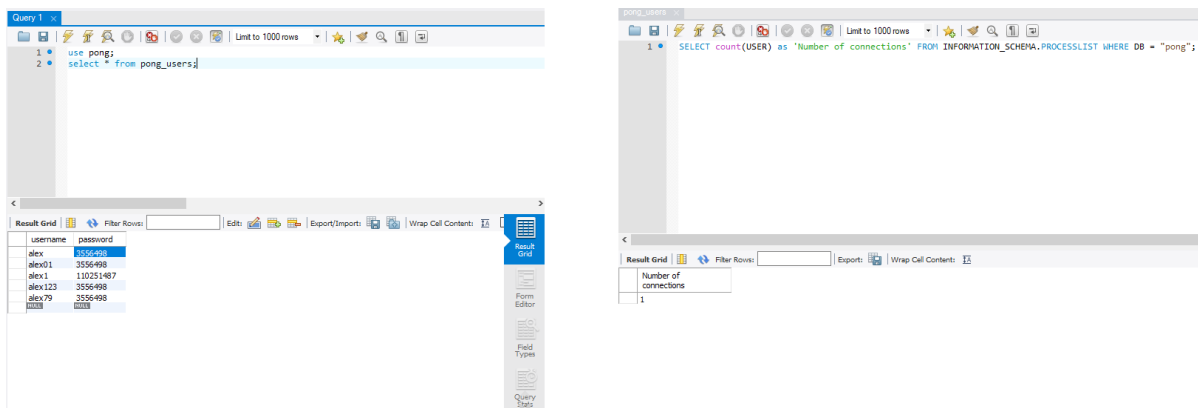


Figure 27. MySQLworkbench testing. These database entries were removed after testing was complete.

5. Third iteration. Building the game and further code improvements.

5.1. Code refactoring.

To begin this iteration, I improved the code within the application. Some of the text on the screens were String variables drawn with the SpriteBatch but I have implemented them as Label actors for the stage. This way I have much better control over the widgets on the screen and extra functionality such as resizing or hiding becomes available. I also used a tool called Hiero font creator to build my own font for the game. This was really useful for customizing the size and design of fonts for different areas of the application. [12.5]

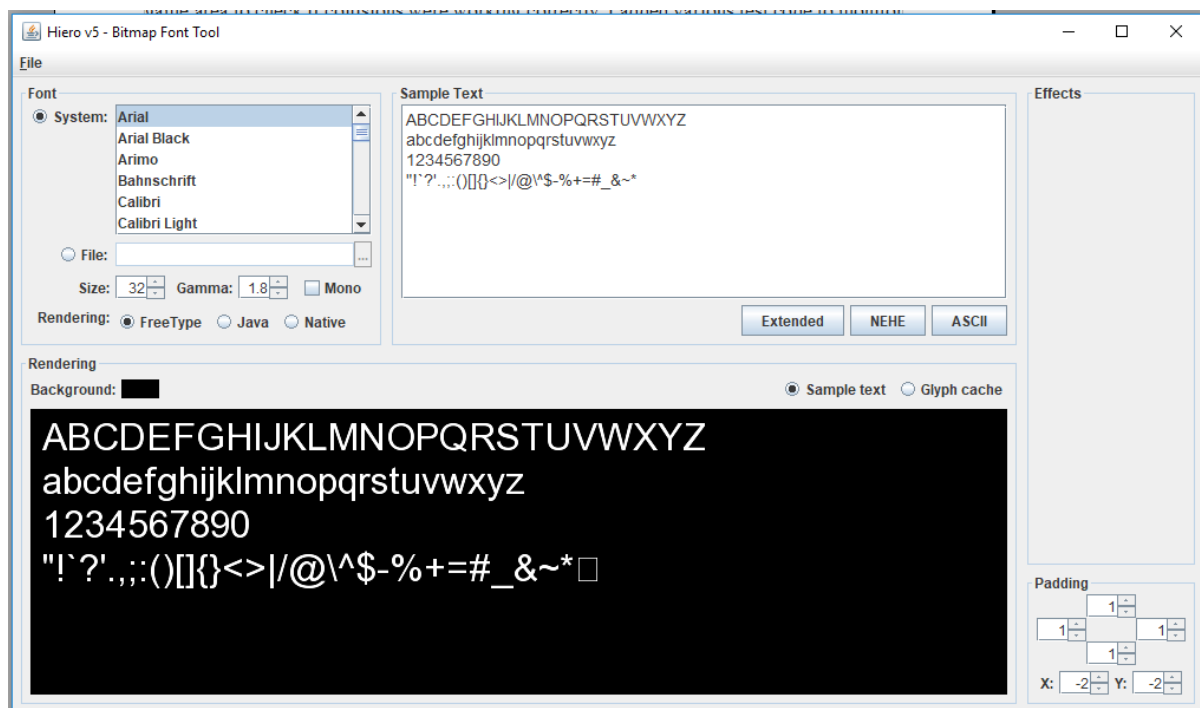


Figure 28. Hiero font creator.

5.2. Game creation.

Following this, I began work on the game screen. Before the game starts and the ball begins to move I wanted to have a grace period so that the player can get ready. To accomplish this I added a label with the value “Play” that fades out over one second. I also added another label at the very top of the screen that details the controls. This controls label fades out over five seconds. After the first fade out action has completed, the ball spawns and begins to move slowly towards the player from the centre of the screen. This movement is created with the use of an UpdatePosition(float x, float y) method that is called every frame and alters the coordinates of the ball by

the specified parameters. E.g. If the parameters 0,1 were supplied then the ball would continuously move one coordinate directly up.

I have already mentioned the ball which is one of the extra classes I had to create for the game. I also had to design the Paddle, PlayerPaddle and ComputerPaddle classes. Paddle is an abstract parent class which extends the LibGDX class rectangle and contains the foundation code for a paddle. Instances of the child classes are controlled by the GameController class and then each frame the GameScreen class renders the objects to the screen.

The next part of development was to create the player. I had to create a sprite for my paddle which was just a basic single coloured rectangle. Then I needed to give the paddle a width and height and adjust the sprite to always have the same position as the paddle. This was important for collision detection later on. Providing movement for the paddle was straight forward as it could only move up and down on the Y axis. I specified two controls as the up and down arrows which moved the paddle in the relative direction by five coordinates. The method that took input was called each frame in the GameController class so the new position could be rendered. [24]

Next up was development of the ball movement and interaction with various parts of the screen. The ball would have to deflect and reverse direction when it collides with the top and bottom Y axis. There would also have to be a continuous test for collisions with the paddles and the new direction of the ball would have to be calculated dependent on where it intersected. Scoring would be another aspect of the game. The ball would need to be reset if it reached the min or max x axis and the appropriate scores altered.

Deflecting the ball of the Y axis proved fairly simple. The X velocity remained the same and I multiplied the Y velocity by -1 to get the inverse. There was a small bug where the ball would get stuck on the side of the screen so I had to add a line of code to update the Y position of the ball by five coordinates inside the screen. Below is the method that did most of the work for this.

```

public void checkYOutOfBounds() {

    if(ball.getY() < 0) {

        ball.updatePosition(0, 5);
        ball.setVelocity(ball.getxVelocity(), ball.getyVelocity()*-1);

    }

    if(ball.getY() + ball.getBallSprite().getHeight() > Constants.VIEWPORT_HEIGHT) {

        ball.updatePosition(0, -5);
        ball.setVelocity(ball.getxVelocity(), ball.getyVelocity()*-1);

    }

}

```

Figure 29. Checking the Y coordinate of the ball remains inside the game area.

The trickiest part of creating the ball was definitely the collision with the paddles. I wanted to determine which area of the paddle was struck and then calculate the new ball direction accordingly. E.G. If the ball hit the top side of the paddle, it would deflect with a positive Y velocity. If it struck the middle then it should bounce with a Y velocity close to zero and colliding with the bottom of the paddle would result in a negative Y velocity for the ball. [5.1]

The rectangle class that both the ball and paddles inherit contains a useful method called overlaps() which checks if one rectangle touches another's coordinates. I used this to check if there was a collision and then I calculated the ballIntersect coordinate with this line of code. [8] [27, page 213]

```

ballInterSect = (playerPadd.getY()+(playerPadd.getHeight()/2))-(ball.getY()+ball.getHeight()/2);

```

Figure 30. Ball intersect.

It takes the player Y coordinate which is the bottom left corner of the paddle. Then adds half the length to get the middle of the paddle. The same process is done for the ball Y coordinate and then a subtraction is made. This results in the difference between where the middle of the ball is and the middle of the paddle at the point of collision. From here I could divide the ballIntersect by the paddle height to normalize the result, meaning a range of -1 to 1. I could then multiply the normalized variable by a number to determine the severity of the angle that the ball bounces off at.

```
ball.setVelocity(ball.getXVelocity()*-1, (Normalized*3.5f)*ball.getXVelocity()/2);
```

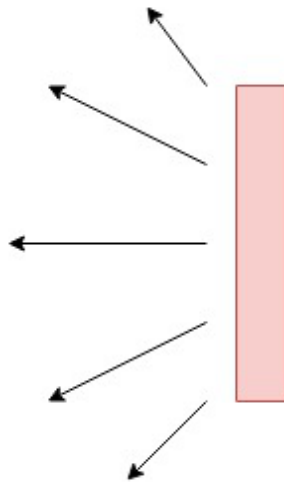


Figure 31. Ball collision formula and desired effect on collision.

This is the formula I ended up with and the desired effect, it could definitely be improved but it works. It reverses the X velocity first of all. Then multiplies the normalized variable by a number to represent the severity of the Y angle that the ball bounces off the paddle at. The Y velocity is also multiplied by half the X velocity. This is because I have designed the game so that the ball X velocity automatically increases over time until a certain limit is reached. It means that the ball will still bounce off at an angle even when the X velocity is at the maximum. [6.2]

The final part involving the ball was checking for collisions on each side of the X axis. This was fairly similar to the `checkYOutOfBounds` method but included extra functionality to start a chain of events for the scoring process and resetting the ball.

I took some time here to test the game and created a dummy paddle for the other side of the game area to check if collisions were working correctly. I added various test code to monitor the ball X velocity, Y velocity and intersect coordinates at every single frame of the game. The results were printed to the console.

After some calculation modifications, I was happy with the result and I moved on to developing the computer paddle. I had to create an artificial intelligence that would move towards the ball but remain beatable. At first I tried to move the paddle to the Y coordinate of the ball but limit the speed of the paddle. I wasn't satisfied with the results so I attempted other options. I then managed to come up with the following calculation for the computers movement. It attempts to move the middle of the paddle to the middle of the ball on the Y axis. However, It negates the current Y coordinate divided by a number which represents the difficulty. I divided by the difficulty because it had the desired effect, it also worked for both negative and positive paddle movement. The lower the number, the higher the difficulty.

```
updatePosition(((gameBall.getY()-getHeight()/2+gameBall.height/2)-getY())/difficulty);
```

Figure 32. Computer paddle formula.

To process all the event handling and position updating I have created an update method within the gameController class which is called each frame by the render method in the gameScreen class.

5.3. Further application improvements and creating the scoring system.

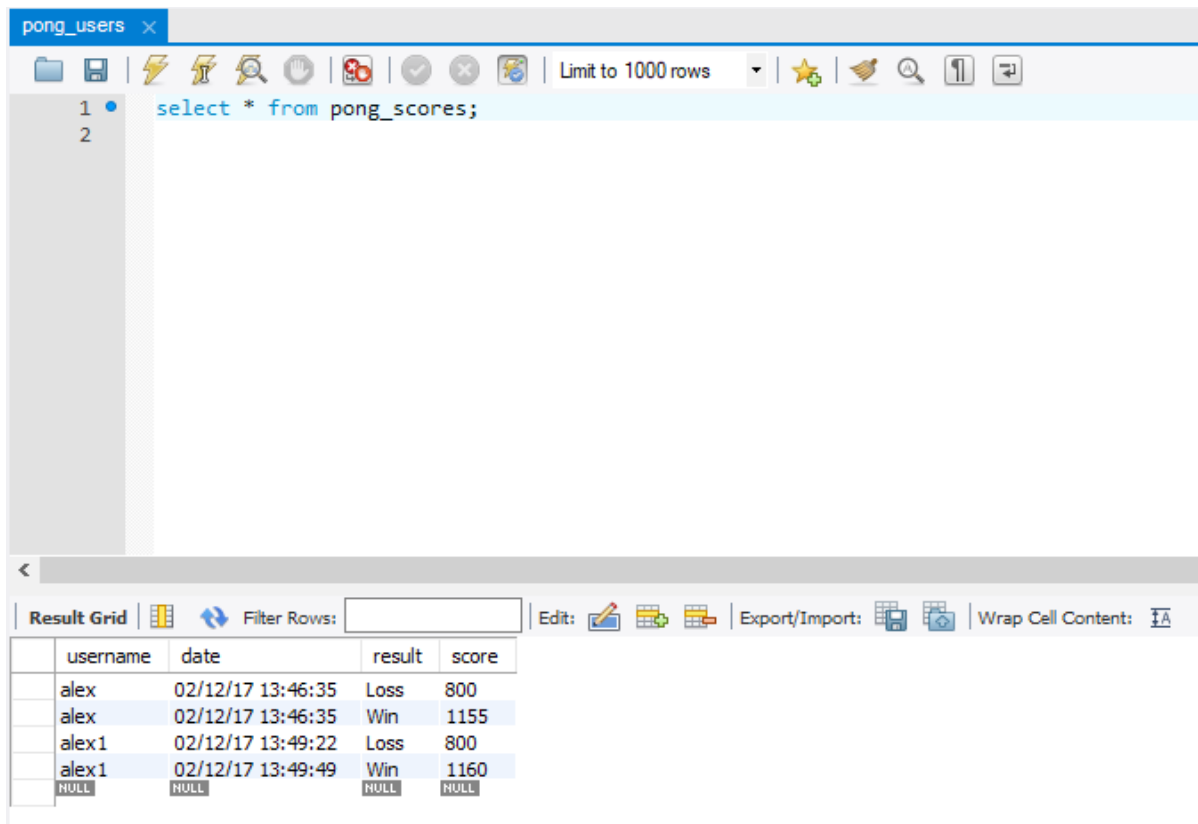
Once the basic game mechanics were working, I moved on to create the scoring system. I added integer variables to represent each players score and then rendered them to the screen. These will be incremented each time the ball collides with the corresponding side of the game. The winner is the first paddle to reach two goals.



Figure 33. The current state of the game. The controls tooltip at the top is in the process of fading out.

I also needed a separate player score to be stored within the database so that I can refer to it for the high score screen later on. This score is increased for each player hit and goal. It is decreased for each computer hit and goal. When inserted to the database, the player score will be accompanied by the username, date and result of the

match. The schema I mentioned above will now include a results column in the pong_scores table. The values will either be “Win” or “Loss”. Below is an image of test entries to the pong_scores table.



The screenshot shows a database application window titled 'pong_users'. The SQL editor contains the query `select * from pong_scores;`. Below the editor, the 'Result Grid' displays the following data:

username	date	result	score
alex	02/12/17 13:46:35	Loss	800
alex	02/12/17 13:46:35	Win	1155
alex1	02/12/17 13:49:22	Loss	800
alex1	02/12/17 13:49:49	Win	1160
NULL	NULL	NULL	NULL

Figure 34. pong_scores table entries test.

I want to briefly mention how easy it was to separate the application into modules for testing. I created the login screen first and would have had to login each time I wanted to test the application if I followed the regular program flow. However, due to the nature of the setScreen method, I was easily able to bypass the login phase to work on the game with more convenience.

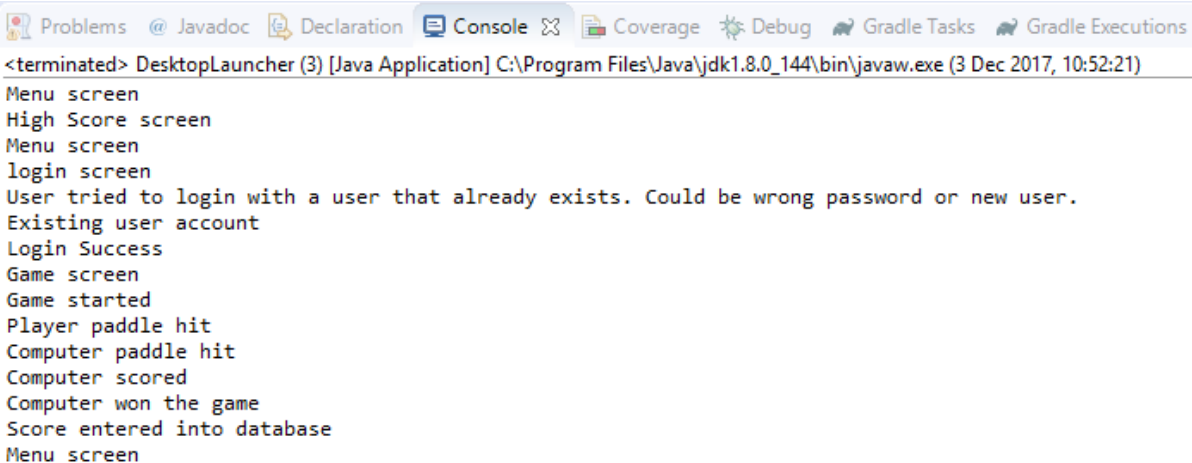
I went on to make some improvements to the general application. I made it possible for a user to login with a different account once they were logged in. I also added code to display the username of the logged in account. After some consideration, I removed the assets class that I mentioned earlier within the class diagram. The game is going to have very few sprites and a separate class is not currently warranted. [27, pages 149-155]

One last check I made was to ensure there was an index for the username column of the pong_users table. Since the column is part of the primary key an index was automatically generated. Database indexes are created for columns in tables that are regularly used for search queries. They speed up the query process by creating a separate tree data structure with that column data to reduce the amount of records that need to be searched. [15.3]

6. Fourth iteration. Building the high scores screen. Adding sound and art to the game.

6.1. Event Logger incorporation.

Before moving onto further development, I wanted to change the way I monitor the game flow. I didn't want to have various print statements with in the code so I built the event logger class for more convenience. This is an abstract class that contains many static methods that are called when a significant event occurs within the game. This made it really easy to see the game life cycle and if there were any anomalies. It also made the code more readable as the method signatures are very descriptive of what event should be happening at that stage. I could also negate or add methods when necessary without messing with the game code. Below is the eventLogger displaying a typical life cycle of the game.



```
<terminated> DesktopLauncher (3) [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (3 Dec 2017, 10:52:21)
Menu screen
High Score screen
Menu screen
login screen
User tried to login with a user that already exists. Could be wrong password or new user.
Existing user account
Login Success
Game screen
Game started
Player paddle hit
Computer paddle hit
Computer scored
Computer won the game
Score entered into database
Menu screen
```

Figure 35. EventLogger in action.

6.2. The high score screen.

The high score screen is a significant part of the application. Rather than displaying a static number of scores, I wanted to create a screen that was dynamic and could change depending on player input. I plan on allowing the player to view rankings of their own scores as well as global scores.

I started off by creating a couple of new methods in the database manager to return result sets of high scores for both the logged in player and all players. The highScores

method returns every attribute from the pong_scores table ordered by score and in descending order with a limit of 50. The method uses a preparedStatement which is the best choice since it will likely execute multiple times during a game life cycle. It saves the database from computing an execution plan each time the statement is run. The method is shown below. [26, pages 432-433]

```

194
195 public ResultSet highScores() throws SQLException {
196
197
198     returnScoresQuery = "select * from pong_scores order by score desc limit 50";
199     returnScoresStatement = conn.prepareStatement(returnScoresQuery);
200
201
202
203
204     eventLogger.highScoresLoaded();
205     highScoresRetrieved = true;
206
207
208     return returnScoresStatement.executeQuery();
209
210
211 }
212
213

```

Figure 36. The highScores method.

I then added a method called playerScores which performs a very similar operation however it takes a String parameter for the username and only retrieves scores for the logged in player. This would later be used to give the player the option to view their own high scores. An important point is that this functionality would only be available to a logged in player, otherwise only the global scores would be shown. I could have probably combined the two methods into one using a single result set. It would involve querying the database each time there is player input and I wanted to avoid this for performance reasons.

Then I started to work on the structure of the high score screen. I added a title and a button that lead to the menu screen. This was followed by adding three buttons to the right of the title, top ten, top twenty-five, and top fifty. These will be used by the player to select the number of high score rankings displayed. [27, pages 235-236]

After that, I needed to come up with a way to use the result sets acquired from the database manager to render the scores to the screen. I wanted to create a flexible solution that could be re-used for all the different buttons. I came up with the renderScores method below.

```

318
319 private void renderScores(int numberOfRanks, int yDecrement, BitmapFont font, ResultSet scoreSet) throws SQLException {
320     int yStartHeight = 550;
321     int rank = 1;
322     int iterations = numberOfRanks;
323
324
325     scoreSet.first();
326
327     do {
328
329
330
331
332         font.draw(pongGame.getBatch(), Integer.toString(rank), 100, yStartHeight );
333         font.draw(pongGame.getBatch(), scoreSet.getString(1), 250, yStartHeight );
334         font.draw(pongGame.getBatch(), scoreSet.getString(2), 400, yStartHeight );
335         font.draw(pongGame.getBatch(), scoreSet.getString(3), 600, yStartHeight );
336         font.draw(pongGame.getBatch(), Integer.toString(scoreSet.getInt(4)), 700, yStartHeight );
337
338         yStartHeight -= yDecrement;
339         rank++;
340
341         iterations--;
342
343     }while(iterations>0 && scoreSet.next());
344

```

Figure 37. The renderScores method.

This method performs a lot of work and will be called with in the render method of the class. The parameters include an integer representing the number of ranks to display, another integer which represents the Y axis spacing between ranks, a BitmapFont to designate a font for the scores to be drawn with and finally, a ResultSet to switch between the player and global high scores. The reason behind parameters for the Y axis spacing and font was to fit different numbers of scores on the screen appropriately. Obviously ten scores would be placed differently than fifty scores. The method body includes a do while loop which essentially goes through the result set and draws each attribute to the screen dependent on the parameters. The parameter variables are set within the clickable buttons and this is how the score screen is manipulated. [27, pages 187-188]

Once this stage of development was complete, I added two more buttons to the screen. A button to show the logged in user's score only and another to show all user scores. These were positioned to the left of the title and would only become visible once a user has logged in. Clicking one of the buttons would modify the result set parameter that the renderScores method takes and therefore render the new set of scores to the screen.

An additional feature I wanted to implement was a player win percentage statistic. This would make use of the result attribute within the pong_scores table. I created a method within the databaseManager class called winPercentage. This made use of the following SQL query to return a resultSet with a number equal to the player win percentage.

```
select sum(result = 'win')/count(*) from pong_scores where username = player
```

Figure 38. The SQL query for player win percentage.

Then it was simply a case of obtaining that number in the appropriate percentage format with no decimal places and storing it into a String variable. This variable is made visible once a user has logged in and rendered to the top left of the screen. [7]

Below are a couple of screenshots showcasing the current state of the high score screen. The first is during a scenario where a user is not logged in and the top twenty-five button has been selected. Scores for all users are displayed. I will be looking to change the font so that the spacing between rankings is more appropriate.

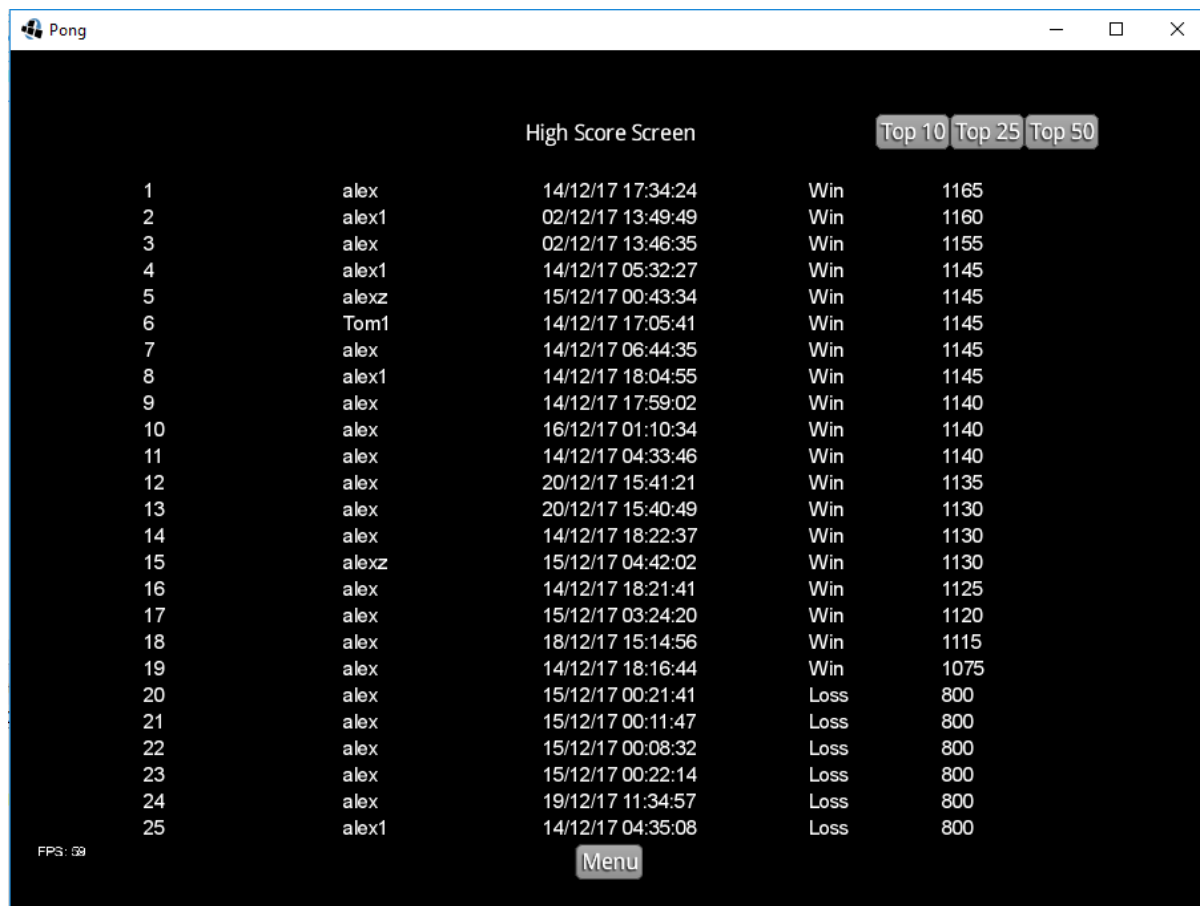


Figure 39. The initial high score screen. This is displayed for a user that is not logged in.

This second screenshot was taken after a user had logged in. The upper left buttons and win percentage have become visible. Only the top ten high scores are displayed as that is the default when the high score screen loads. I also selected the logged in user option so that only the scores for the current user are displayed. I am satisfied with the functionality as the buttons that display a certain number of scores work with all user scores and the logged in player's scores. I included the time along with the date because it means that every entry is unique.

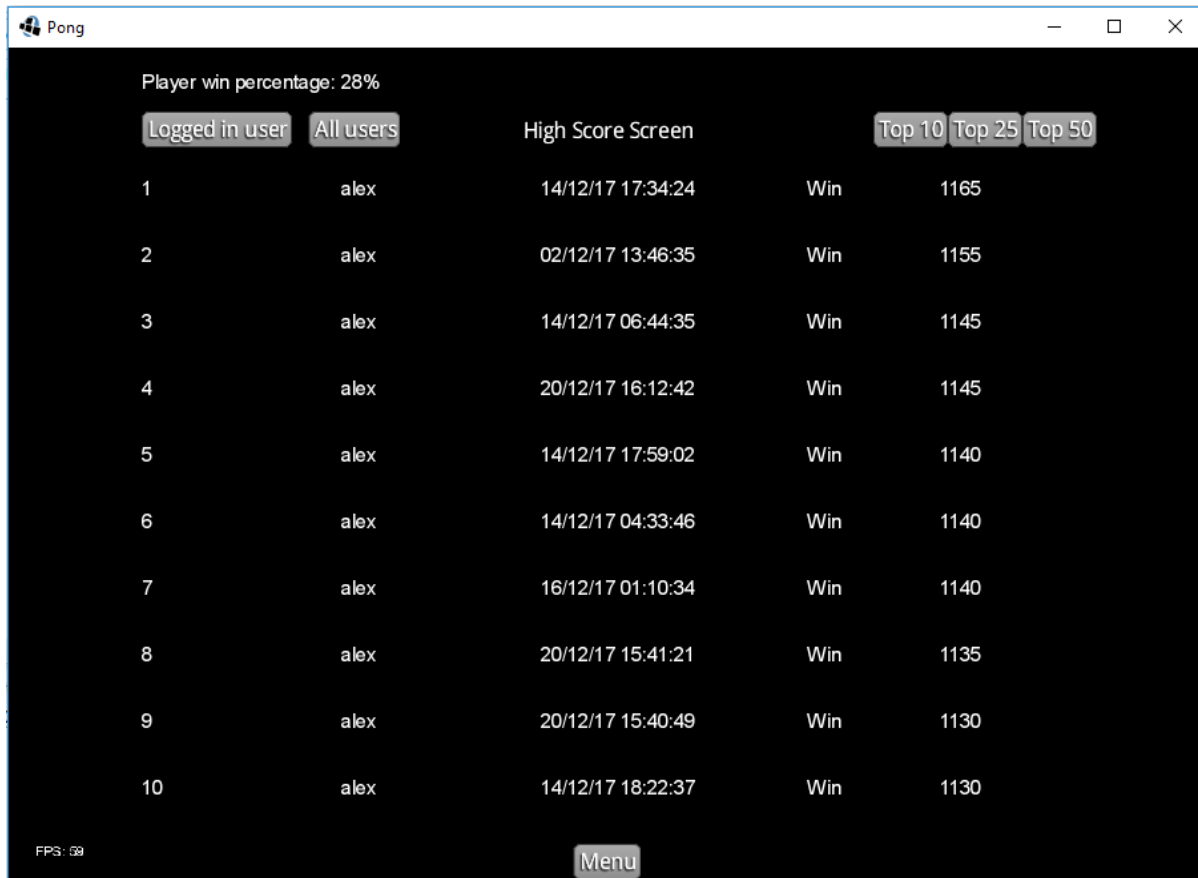


Figure 40. The high score screen displayed for a logged in player. Extra functionality becomes visible in the top left.

At this stage, I went on to add a few additional methods to the eventLogger to test the high score screen synergy with the database. I fixed a small bug where the score of a paddle was being increased multiple times when the side of the paddle was contacted by the ball. I also fixed an obvious mistake where I didn't display the player's score on the game screen once the game had finished.

Later, I went back and made some further improvements to the high score screen. Rendering fifty scores to the screen was troubling and proved to be too many records on the page at once. It was difficult to get the appropriate spacing between scores whilst having a large enough font. I decided to change the top fifty button to a top forty button as this was a much more suitable number of scores to display on the page. I also added titles to each column of data. Lastly, I coloured the font of every other score grey as it allows users to distinguish different scores easier.

6.3. Adding sound and background art to the game

Sound is an important part of every game as it brings the game to life and adds character. For my application I want to include user interface sound effects, game event sounds and music. I started off by researching royalty free sound material so that I didn't need a license to make use of them. [19, 19.1, 19.2, 19.3, 19.4, 19.5, 20, 20.1]

I started off by adding minimalist sounds to every button click within the game. Following that, I altered the game screen so that sound effects played during every event. This included paddle hits, wall touches, goal scores and the end of the game. Finally, I picked two pieces of music to use. I chose a couple of tracks as I wanted a little bit of variety rather than the same music looping. I altered the music in audacity, a free open source audio editor, so that it was more efficient for the game. LibGDX makes it relatively easy to implement sounds with the following piece of code. [27, pages 309-312]

```
private Sound buttonSound = Gdx.audio.newSound(Gdx.files.internal("buttonSound.mp3"));
```

Figure 41. The declaration and initialization of a Sound object.

It assigns the Sound object to the designated asset. Music is implemented in a very similar fashion. There are also lots of useful methods such as play, stop, pause and loop. I was also able to create a global volume float variable that I re-used for all sounds. [10.5, 12.7]

After being happy with the sound, I went on to add background art to the game. Up until now the backgrounds had been solid black and the game was obviously very dull. I wanted to find a few colourful pieces of art to add to the application. I managed to find this fantastic source of artwork usable in non-commercial products. Together with the music, the game started to have a retro-space theme. Below is one of the background pieces I used. [21]

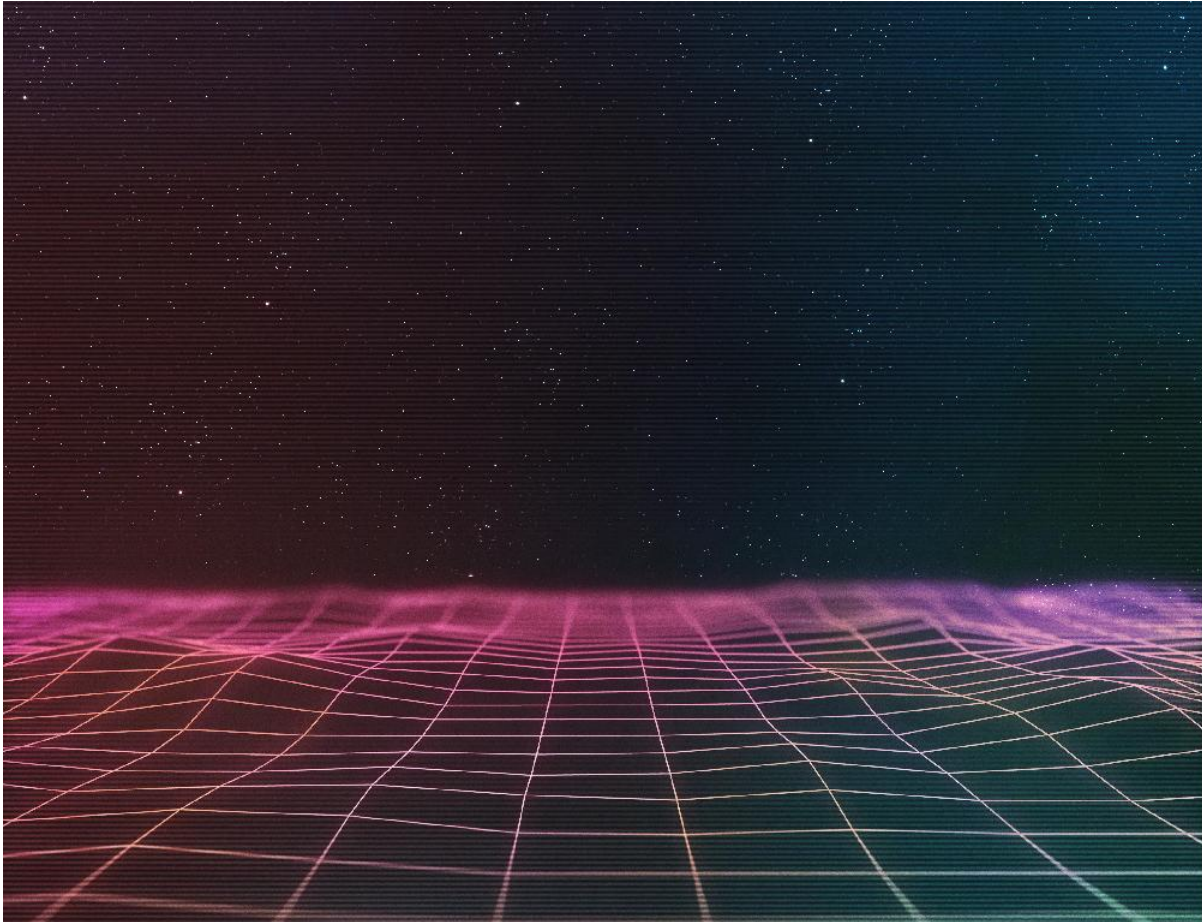


Figure 42. The artwork I used for the log-in background.[21]

I added the JPEG file as a texture so that I could load it into a sprite and set the size according to my application dimensions. This worked really well because my application default resolution is 1024 x 768 pixels and the original image was 3200 x 2560. The result was a very high quality image that scaled appropriately without blurring. As I only included a few images in the game, the added memory required was relatively insignificant. Below is the code for the background sprite creation. [27, pages 94-99]

```
background = new Texture(Gdx.files.internal("loginBackground.jpg"));
backgroundSprite = new Sprite(background);
backgroundSprite.setSize(pongGame.getAppWidth(), pongGame.getAppHeight());
```

Figure 43. The creation of a background Sprite.

Following the addition of background art to each screen, I wanted to alter the font as it was rigid and did not scale well. Up until now, I had been using fnt files which rely on an image file to be drawn and don't scale well during application resizing. After some research I found a solution by using True Type fonts. True Type fonts are made with

straight line segments where as fnt files consist of a pixel matrix. This is the reason for the blurred effect of large scale fonts. In addition True Type font files are relatively small in size making them efficient. [12.6]

During an interjection I removed the Constants class as it was not utilized as I had hoped. I have also read on several occasions that static variables do not function well on the Android platform as memory is limited and reclaiming storage is important. A static variable remains for the entire application execution cycle. If I do deploy a version of my game to Android then this change may save me from issues later. [5.4]

7. Fifth iteration. Adding a settings screen and a simple web service.

7.1. Settings screen development.

Throughout development I have often had ideas for extra functionality within the game. Whether that be added game mechanics, different game modes or new screens. One of the ideas I believe is a priority would be to implement a settings screen. This would give the user flexibility with my application.

Providing a difficulty option was fairly straight forward as I stored the setting within a variable inside the computer paddle class. This meant that I could implement getters and setters for the variable to change it according to the selected difficulty. I added a drop down menu using the LibGDX selectBox class with easy, medium and hard options. These selections change the difficulty float variable to 12.5, 10.5 and 8.5. [27, pages 235 -236]

I then went on to add a slider widget to represent the volume. The bar would have steps of 0.1 from 0 to 1. This would alter the global volume variable which all music, button and game event sounds use. I also added a label next to the slider to show the selected value. [27, page 249]

The next setting was the resolution. I used the selectBox class once again but had the options 1024 x 768, 1440 x 900, 1600 x 1050 and Fullscreen. I chose these options as they are extended graphics array resolutions commonly available on most machines and work well with the game proportions. Wide screen resolutions did not fit quite as well. I used methods from the Gdx graphics class to set the selected option. One of the methods I created is shown below. [27, pages 242 – 244]

```

412- /**
413-  *
414-  * Called to set the graphics resolution dependent on the selected setting
415-  *
416-  * @param res
417-  */
418-
419- private void setResolution(String res) {
420-
421-     if(res.equals("1024 x 768")) {
422-         Gdx.graphics.setWindowedMode(1024, 768);
423-     }
424-     else if(res.equals("1440 x 900")) {
425-         Gdx.graphics.setWindowedMode(1440, 900);
426-     }
427-     else if(res.equals("1600 x 1050")) {
428-         Gdx.graphics.setWindowedMode(1600, 1050);
429-     }
430-     else if(res.equals("Fullscreen")) {
431-         Gdx.graphics.setFullscreenMode(Gdx.graphics.getDisplayMode());
432-     }

```

Figure 44. The method used to change the resolution after a user has made a selection.

I was happy with these settings but I then had the idea to implement one more option which would allow players to set the controls for the game. It would mean they could choose any key for the up and down movements. This proved to be a little more complex as I had to add an input listener to the stage once the user selected the option to change the button. I would have to record the next key input and assign it to the selected movement direction. The input listener would then have to be removed. Another factor was making sure that the up and down movement keys were not assigned to the same input. I added some prevention code and a error label to avoid this. The screen would also render the newly selected key to the controls section. Below is the full settings screen with the default selections.

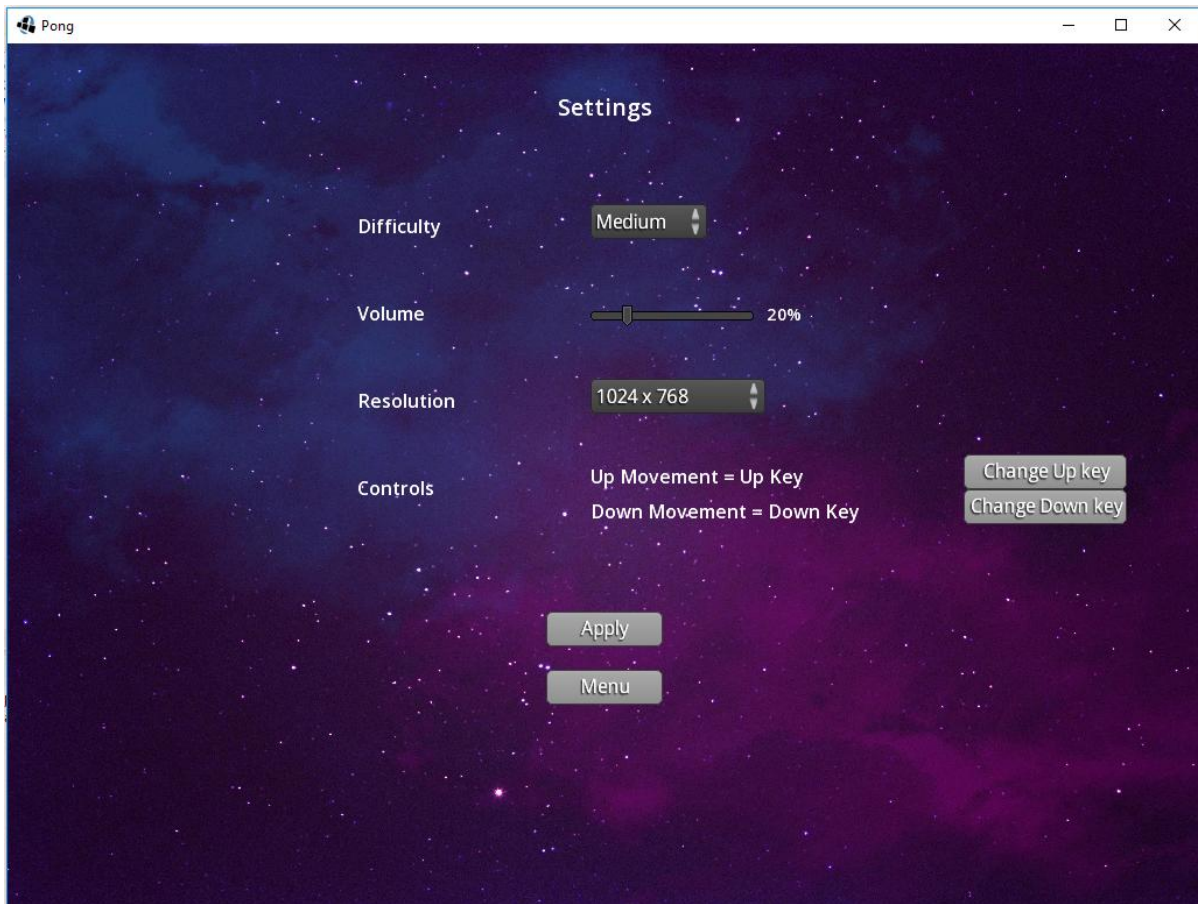


Figure 45. The complete settings screen.

I included an apply button below the settings which has to be clicked to confirm the changes to the difficulty, volume or resolution. Otherwise the value reverts back to the previously selected option when the screen changes. On the other hand, the controls are changed as soon as the user successfully presses a valid key upon clicking the change control button. This is because it was better to keep a smaller scope and it reduced a lot of boiler plate code.

I was really happy with the resulting screen as it provides players with the flexibility I intended. I then made a couple of changes to other parts of the game. I altered the game score algorithm so that it compensated for difficulty. Harder difficulties reward more points.

The current sprites for both paddles were exactly same so I changed the computer sprite to include a CP so that it is more clear to the player. Below are the paddle sprites I use.

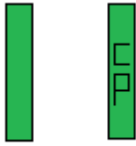


Figure 46. The paddle Sprites I have created. “CP” signifies the computer paddle.

7.2. Web service for database password.

I was not happy with the lack of security regarding the database credentials. Therefore, I created a very simple PHP web service on the Birkbeck Student web space to retrieve the database password. The web service displays nothing unless a single correct hash-coded value is provided as a \$_GET parameter. My PHP script then generates a random key as one of the two keys used for AES(Advanced Encryption Standard) encryption/decryption. The script then displays a unique cipher text string which is the result of a base64 encoded and encrypted password. The first randomly generated key is encoded and concatenated as part of the cipher text; the other is stored inside the game application. [28]

The URL for the web service is:

<http://titan.dcs.bbk.ac.uk/~apearc03/projectphp/projectPHP.php>

The resulting cipher text is random for each key generation and resembles the below example.

THNaVXNIcFRUZ0R6R3d1Zw==l8WRfEFF8a9nJv4URnudqLjHKTqxxzf8L2Ej
QHV5+pg=

My game application decodes and then decrypts the string to get the database password. Of course, it is still possible for an attacker to obtain the database credentials but this is definitely an added layer of obfuscation and it means that the database password is not hardcoded inside my application. The downside is that the web service is now an additional dependency for the game but I felt the trade off was worth it. In the future I would design my game following the client/server model instead.

Lastly, I added a simple loading screen to the game. This is shown whilst assets are loaded and a database connection is made.

8. Deploying the game.

8.1. Finished game diagrams.

Earlier on in the setup chapter I added a class diagram representing the game structure at that point in time. A lot has changed since then so I wanted to include a final class diagram displaying the finished organization of the application.

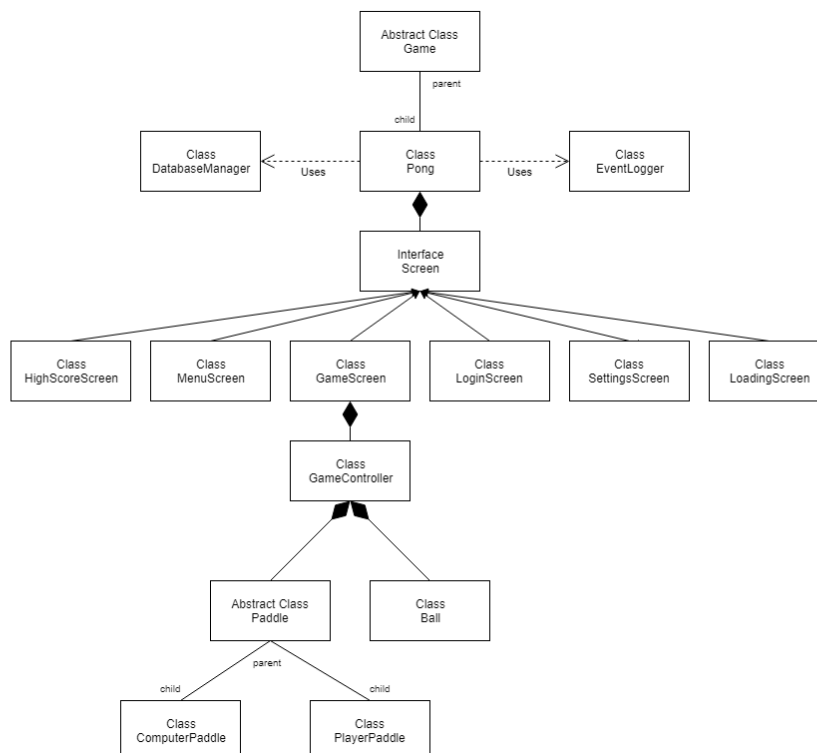


Figure 47. The updated class diagram.

To complement this, I also created an activity diagram for the game. This displays the potential application life cycle.

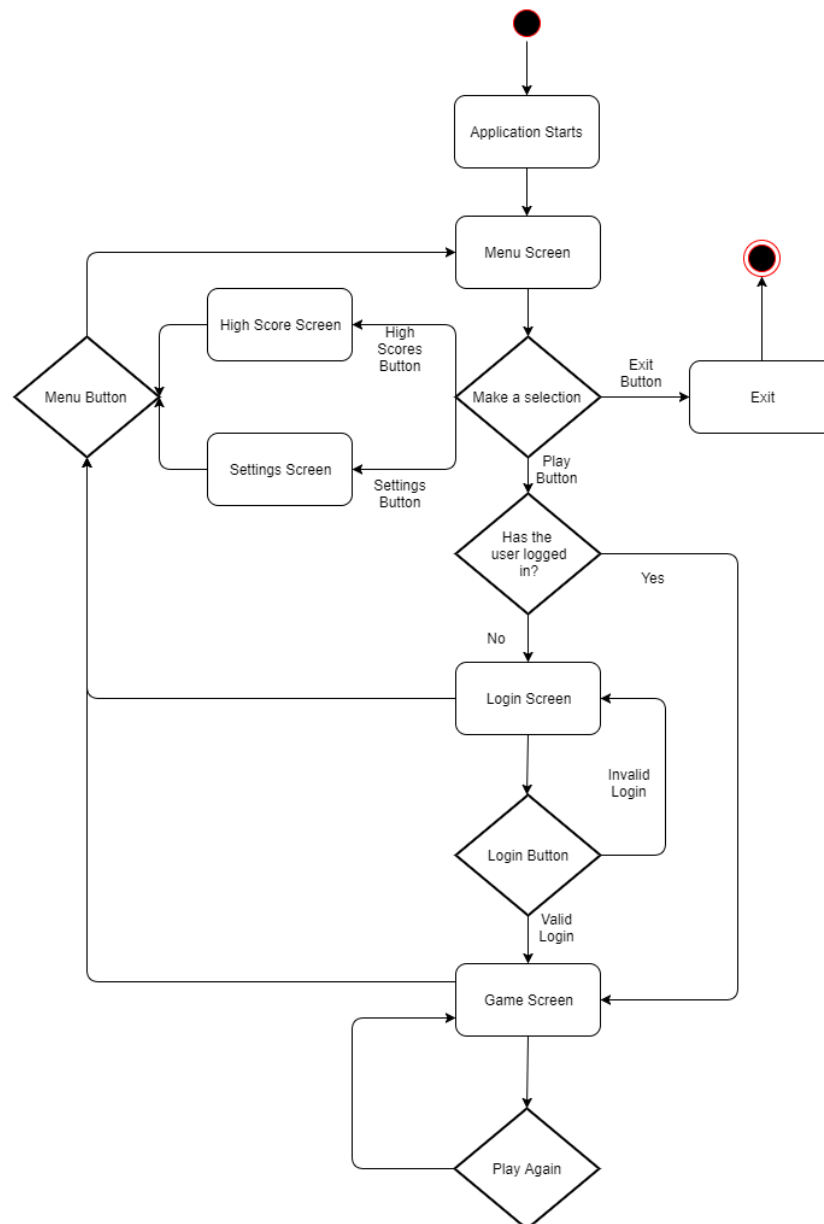


Figure 48. The activity diagram representing the game life cycle.

8.2. Deploying the game to Desktop.

To deploy the game on a desktop environment I had to perform a few steps. Firstly, I stripped the source of any unnecessary comments and testing code. This included the eventLogger class. Next, I remade a LibGDX project with only the required launcher, libraries and tools. Before that, I was working with a project that included all platform launchers for testing. From there, I exported the project as a runnable jar file. Eclipse makes this process very easy. [11.5]

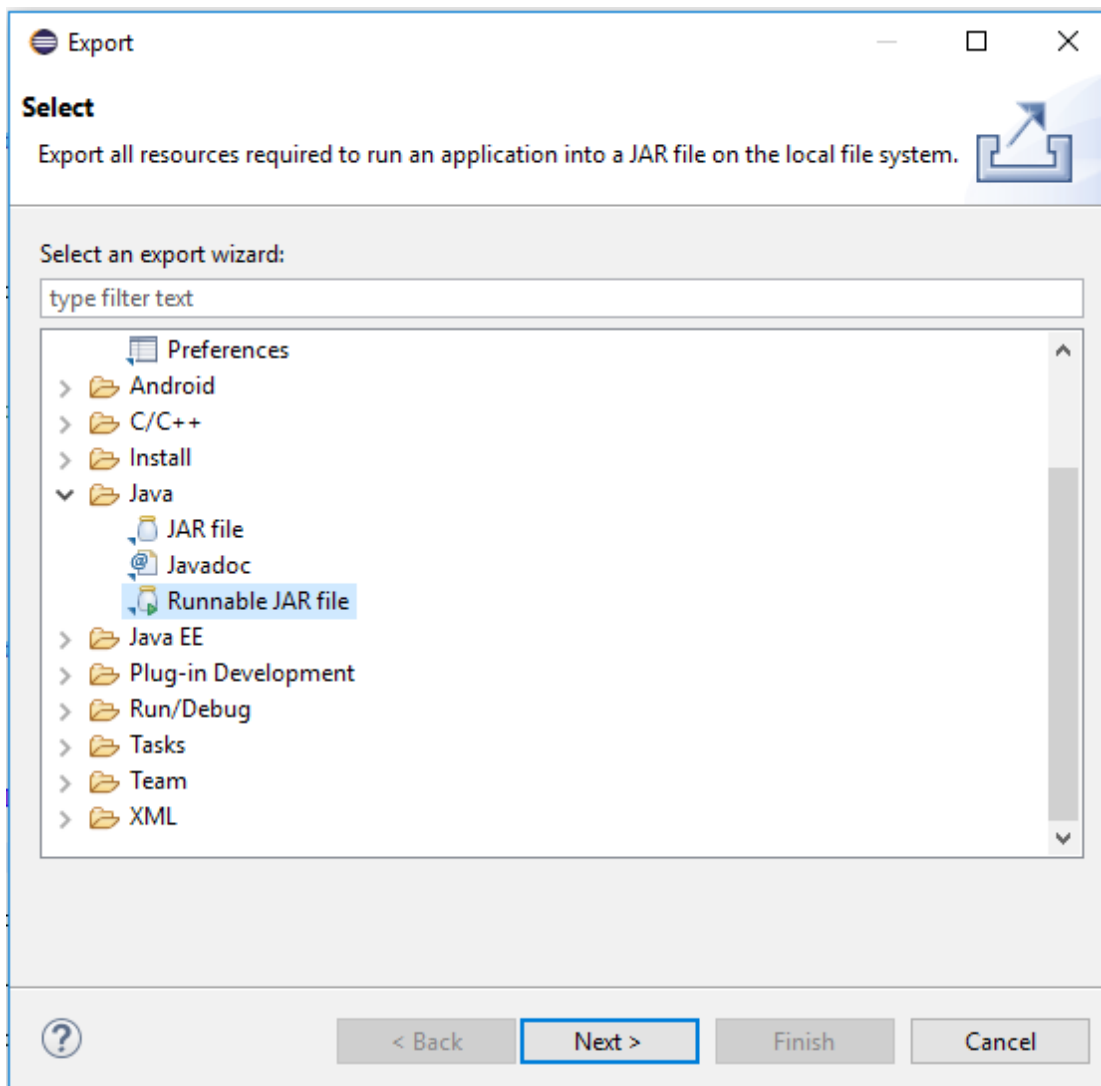


Figure 49. The process of exporting a Runnable JAR file.

This process produced a file that would run on any desktop environment. I tested the file a few times and it worked as intended.

8.3. Deploying the game to Android.

The android process was a little more involved as I had to make alterations to the source code. Obviously an android device does not have the same peripheral and resolution options. I also mentioned earlier that my JDBC-SQL implementation would not work on the Android platform so I had to exclude database functionality. I began by creating a separate Android LibGDX project. My current paddle controls involve using a physical keyboard so I had to change this. I added two on-screen buttons to move the paddle up and down; this worked well after testing it with an Android emulator. Further on, I changed the positioning and size of many screen elements to fit on a smaller screen. I also fixed a bug where changing the difficulty on Android would reduce the fade out time of the “play” animation on the game screen. Below is a screenshot of my game running on an Android emulator within Android Studio. As you can see, the extra control buttons are visible and the screen elements have been resized.

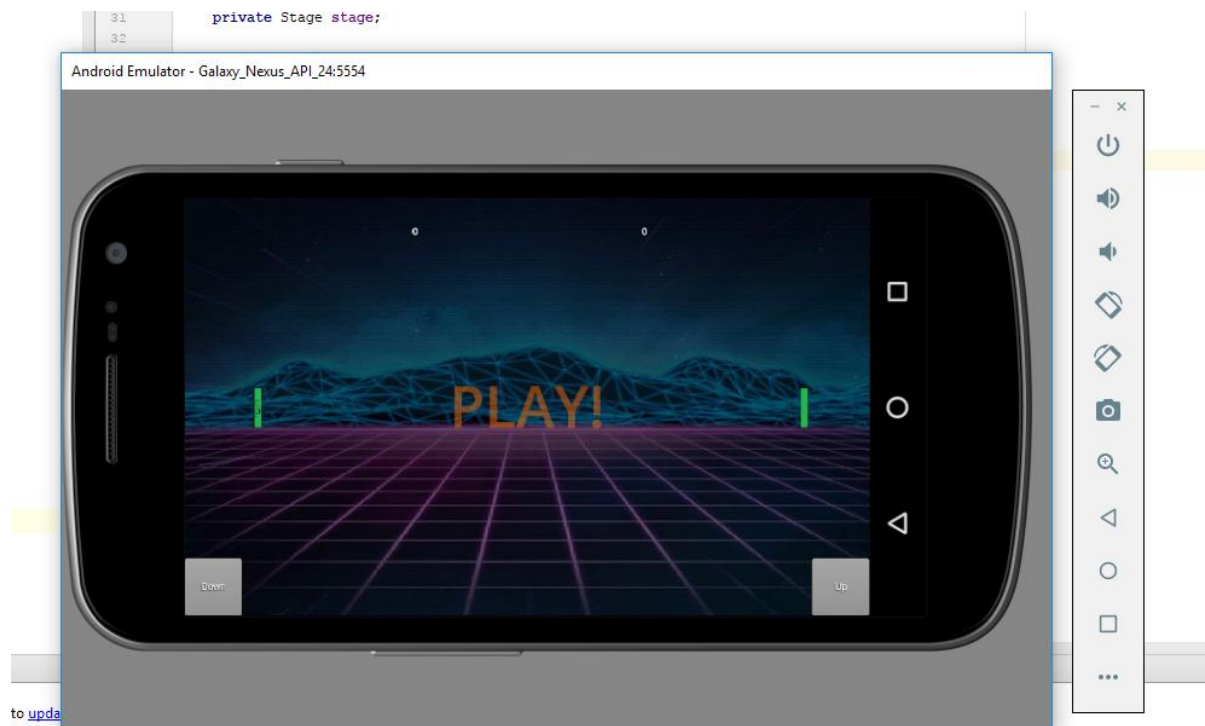


Figure 50. Android Emulation of the application. The screenshot was taken when the game was just about to begin and the “play” animation was finishing.

After further code refactoring to ready the application, I used Android Studio to build and export the game as an APK(Android Package Kit) file. This file could then be installed on any Android device. [23]

9. Conclusion.

9.1. My experience.

Overall I am definitely happy with my project choice and I managed to learn a lot. I like that it incorporated many different technologies. I had previous experience with included technology such as core Java, Git and MySQL. I was able to reinforce and gain new knowledge in these areas. Some of the new equipment I used was the LibGDX framework, Gradle and Amazon Web services. It's always interesting to have exposure to new areas of technology as computer science is such a versatile field. Implementing all of these together proved to be a challenge but the end result was very rewarding. I have never worked with a framework as heavily as I did with LibGDX but I really enjoyed making use of the tools it provided.

Additionally, I was satisfied by the level of difficulty. Building the game to function well with many screens and database connectivity was a challenge. I had to prioritize object orientated principles such as polymorphism, abstraction, inheritance and encapsulation. I also made a big effort to re-use objects wherever possible. Development of the actual Pong game required mathematics and physics to build working paddles and a ball. That combined with rendering the frames approximately 58 times per minute took careful planning. Another factor of the project was including adequate art and fonts that provided clarity and scaled well. I also had to spend ample time testing all inputs and potential outputs of the game. A last point of difficulty was including meaningful messages and errors to follow good human-computer interaction principles.

9.2. Improvements that could have been made.

The game is in a state that I am content with, however there are a number of improvements that could have been made. Games have so many different aspects to them so there is always room for enhancements.

One important improvement would have been a more effective testing platform. I started off the project with the hopes of using JUnit but I had problems implementing it with the launch configurations of LibGDX games. There were ways of working around the problems I was experiencing but I was impatient and wanted to move on with the project. If I had persisted and researched the issues further then I would have had much better testing capabilities. Alternatively, I made use of an event logger class which wrote to the console during any major game event. I also went through lots of manual testing by separating screens, loading them and testing them independently. This did prove to be effective but my original goal was to create automated tests that could be built whenever there were changes to the code. If the game was on a larger scale with hundreds of classes then my manual testing would not be suitable and way too time consuming.

Leading on from that, the application start up time is very quick when there is a database connection. However, without one, the application hangs for about 3-4 seconds before showing the menu screen. I tried to fix this on several occasions by altering the DriverManager properties and attempting to use a SqlDataSource object instead. The defect is not game breaking but it is not convenient for the user. I was unsuccessful in fixing it.

The next potential fault within the game is that the database credentials are stored inside the application. This is very bad practice as it will allow attackers to find them if they look hard enough. After that, any type of malicious database attack could take place. I did obfuscate the situation by adding a remote web service but this is not a concrete solution. To fix this, I could have created my application using a client/server architecture. The server would be responsible for database operations whilst the client would take care of the game logic. This would improve both scalability and security.

In terms of game play. There could be an improvement with the ball movement as it is not as smooth as I envisioned. I believe it was either the result of not using the delta time in the correct way or my ball sprite was not adequate. Delta time is the time between each frame. I did attempt to include this number but it seemed to make no difference as I designated a static fps rate. My ball sprite was extremely simplistic and might not have had suitable dimensions.

9.3. Parts of the project that went well.

Overall, I was very happy with most aspects of the game. I really like the smooth transitions between screens and I think the navigation inside the application is very user friendly. This was accomplished with consistent button positioning and well ordered screens. Any messages or errors inside the application are meaningful. The artwork and music theme has a futuristic techno vibe which is what I was intending. The game elements also scale well during screen resizing.

In terms of game play, there are many positives. The pacing is great and the game is winnable whilst still providing a challenge. The ball X velocity increases over time which means that rounds don't last too long. The event sounds are a nice addition and make the game interactive. Also, the movement of the paddles and ball is very responsive.

I think the high score system adds a nice dimension to the game. It allows players to compare a variety of records and encourages them to strive for higher results. The score sets shown are flexible which was very important to me as it improves the user experience.

I'm really happy with the settings screen. Before the screen existed, the game came across as very static because it had a single control set, resolution, difficulty and volume. Afterwards, players were presented with various options to choose their play style.

I believe the coding aspect of the development went fairly well. The underlying structure of the application is quite effective as it made it very easy to add new screens and test existing ones by isolating them.

As a final positive, the game is built around being connected to the database. However, if no connection is made then I made a point of designing the game to function regardless. The application manipulates it's presentation to work offline and does so quite effectively.

Github link

<https://github.com/apearc03/pongProject>

References

Web URLS and date referenced

General

1. <http://www.computinghistory.org.uk/det/4007/Atari-PONG/> [November 2nd 2017]
2. <https://www.theverge.com/2012/11/30/3709758/history-of-pong> [November 2nd 2017]
3. <https://aws.amazon.com/rds/mysql/> - Amazon RDS for MySQL [November 23rd 2017]
4. <https://www.agilealliance.org/agile101/> [November 3rd 2017]
 - 4.1. <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/> [November 3rd 2017]
5. <https://stackoverflow.com/> [December 7th 2017]
 - 5.1. <https://stackoverflow.com/questions/26566258/rectangle-overlaps-method-libgdx> [December 7th 2017]
 - 5.2. <https://stackoverflow.com/questions/14779104/how-to-allow-remote-connection-to-mysql> [November 23rd 2017]
 - 5.3. <https://stackoverflow.com/questions/34479099/what-is-a-delta-time-in-libgdx> [November 10th 2017]
 - 5.4. <https://stackoverflow.com/questions/19774075/is-it-bad-practice-to-keep-data-in-staticvariables> [December 21st 2017]
6. <https://gamedev.stackexchange.com/> [November 17th 2017]
 - 6.1. <https://gamedev.stackexchange.com/questions/90223/mysql-database-reading-and-writing> [November 17th 2017]

6.2. <https://gamedev.stackexchange.com/questions/4253/in-pong-how-do-you-calculate-the-balls-direction-when-it-bounces-off-the-paddl> [December 10th 2017]

LibGDX/Java

7. <https://docs.oracle.com/javase/7/docs/api/> [November 3rd 2017]

8. <https://libgdx.badlogicgames.com/nightlies/docs/api/> [December 7th 2017]

9. <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> [November 23rd 2017]

10. <http://www.gamefromscratch.com/page/LibGDX-Tutorial-series.aspx>. [November 10th 2017]

10.1. <http://www.gamefromscratch.com/post/2013/09/26/LibGDX-Tutorial-2-Hello-World.aspx> [November 10th 2017]

10.2. <http://www.gamefromscratch.com/post/2013/10/02/LibGDX-Tutorial-3-Basic-graphics.aspx> [November 10th 2017]

10.3. <http://www.gamefromscratch.com/post/2013/10/15/LibGDX-Tutorial-4-Handling-the-mouse-and-keyboard.aspx> [November 13th 2017]

10.4. <http://www.gamefromscratch.com/post/2013/11/06/LibGDX-Tutorial-7-Camera-basics.aspx> [November 13th 2017]

10.5. <http://www.gamefromscratch.com/post/2013/11/19/LibGDX-Tutorial-8-Audio.aspx> [December 21st 2017]

11. <https://libgdx.badlogicgames.com/documentation/> [November 2nd 2017]

11.1. <https://libgdx.badlogicgames.com/documentation/gettingstarted/Setting%20Up.html> [November 2nd 2017]

11.2. <https://libgdx.badlogicgames.com/documentation/gettingstarted/Creating%20Projects.html> [November 2nd 2017]

11.3. <https://libgdx.badlogicgames.com/documentation/gettingstarted/Importing%20into%20IDE.html> [November 10th 2017]

11.4. <https://libgdx.badlogicgames.com/documentation/gettingstarted/Running%20and%20Debugging.html> [November 10th 2017]

11.5. <https://libgdx.badlogicgames.com/documentation/gettingstarted/Packaging.html> [February 14th 2018]

12. <https://github.com/libgdx/libgdx/wiki> [November 2nd 2017]

12.1. <https://github.com/libgdx/libgdx/wiki/A-simple-game> [November 10th 2017]

12.2. <https://github.com/libgdx/libgdx/wiki/Spritebatch%2C-Textureregions%2C-and-Sprites> [November 10th 2017]

12.3. <https://github.com/libgdx/libgdx/wiki/Orthographic-camera> [November 13th 2017]

12.4. <https://github.com/libgdx/libgdx/wiki/Mouse%2C-Touch-%26amp%3B-Keybaord> [November 13th 2017]

12.5. <https://github.com/libgdx/libgdx/wiki/Hiero> [November 30th 2017]

12.6. <https://github.com/libgdx/libgdx/wiki/Gdx-freetype> [December 21st 2017]

12.7. <https://github.com/libgdx/libgdx/wiki/Sound-effects> [December 21st 2017]

12.8. <https://github.com/libgdx/libgdx/wiki/Scene2d> [November 13th 2017]

12.9. <https://github.com/libgdx/libgdx/wiki/Skin> [November 13th 2017]

MySQL/Databases

13. <https://dev.mysql.com/doc/refman/5.7/en/> [November 3rd 2017]

14. <https://dev.mysql.com/doc/workbench/en/> [November 27th 2017]

15. <https://www.tutorialspoint.com/mysql/index.htm> [November 23rd 2017]

15.1. <https://www.tutorialspoint.com/mysql/mysql-connection.htm> [November 23rd 2017]

15.2 <https://www.tutorialspoint.com/mysql/mysql-create-database.htm> [November 23rd 2017]

15.3. <https://www.tutorialspoint.com/mysql/mysql-indexes.htm> [December 10th 2017]

GIT/GitHub

16. <https://git-scm.com/doc> [November 10th 2017]

16.1. <https://git-scm.com/docs/git-add> [November 10th 2017]

16.2. <https://git-scm.com/docs/git-pull> [November 10th 2017]

16.3. <https://git-scm.com/docs/git-push> [November 10th 2017]

16.4. <https://git-scm.com/docs/git-remote> [November 10th 2017]

16.5. <https://git-scm.com/docs/git-status> [November 10th 2017]

17. <https://guides.github.com/introduction/flow/> [November 3rd 2017]

18. <http://product.hubspot.com/blog/git-and-github-tutorial-for-beginners> [November 10th 2017]

Music, Art and Sounds

19. <http://soundimage.org/> [December 21st 2017]

19.1. <http://soundimage.org/sci-fi/> [December 21st 2017]

19.2. <http://soundimage.org/sci-fi-2/> [December 21st 2017]

19.3 <http://soundimage.org/sci-fi-3/> [December 21st 2017]

19.4. <http://soundimage.org/sci-fi-4/> [December 21st 2017]

19.5. <http://soundimage.org/sfx-alerts/> [December 21st 2017]

20. <https://www.zapsplat.com/> [December 21st 2017]

20.1. <https://www.zapsplat.com/sound-effect-category/science-fiction/> [December 21st 2017]

21. <https://natewren.com/rad-pack-80s-themed-hd-wallpapers/> [December 21st 2017]

Tools

22. <https://www.eclipse.org/> [December 21st 2017]

23. <https://developer.android.com/studio/index.html> [January 17th 2018]

24. <https://www.getpaint.net/> [December 7th 2017]

25. <https://www.draw.io/> [November 3rd 2017]

Books

JDBC/SQL

26. Oracle Database Programming Using Java and Web Services by Kuassi Mensah (2006)

LibGDX/Java

27. Learning LibGDX Game Development by Suryakumar Balakrishnan Nair and Andreas Oehlke - Second Edition (January 2015)

Course material

28. Building web applications using MySQL and PHP module lesson slides. Session 7 - Web Services. [February 10th 2018]