

Final Year Project:

Chase the Pig

Name: Breanainn Cahill

Student Id: 12157163

Course: LM110

Supervisor: Andrew Ju

**Table of Contents:**

### Contents

**Acknowledgments 3**

**Summery 3**

**1. Chapter 1: Introduction 4**

1.1. General Information..................................................................... 4

1.2. Motivation………........................................................................ 5

1.3. Objectives…………………......................................................... 5

1.4 Report Structure…………………………………………………. 6

**2. Chapter 2: Research 7**

2.1. Existing Products……………...................................................... 7

2.2. Exploratory Programs……........................................................... 7

2.3. Academic Research....................................................................... 8

2.4. Implementation Technologies....................................................... 9

**3. Chapter 3: Development 11**

3.1. Design…………..……………...................................................... 11

3.2. Java Sockets and Multi-Threading................................................. 12

3.3. Application Programming Interface.............................................. 14

3.4. Creating a JavaFX Menu………………………………………... 16

3.5. Using JavaFX Layout Panes to Create the Game..……………… 19

3.6. The Application Thread and the Task Thread….………………... 20

3.7. Game Controls…….……………………………........................... 21

3.8. Player Management....................................................................... 22

3.9. Android Activities and XML Layouts…………………………... 24

3.10. Android Gameplay……………………………………………... 25

3.11. Expose Pop-up Activity…………………………..……………. 26

3.12. Difference Between Android and JavaFX……………………... 27

**4. Chapter 4: Results 28**

4.1. Server Program Structure…………………………………………28

4.2. JavaFx Program Structure……………………………………….. 32

4.3. Android Program Structure……………………………………… 40

**5. Chapter 5: Conclusion 46**

5.1. Future Development……………………………………………... 46

5.2. Final Thoughts…………………………………………………… 46

**Figure Index 47**

**References 49**

**Acknowledgements:**

I would like to first and foremost give a huge thank you to my supervisor Andrew Ju, during my planning and development of my final year project, “Chase the Pig”. I really appreciate him for his help throughout the project even though he was busy working in Dublin, he still found the time to give me his valuable feedback throughout the project. I would also like to thank him for the idea for the project which was an ideal fit for me.

I would also like to thank my friends and family for the support and encouragement I received from them as I progressed through it.

**Summary:**

This project is based on a card game called Gong Zhu. Gong Zhu is a Chinese card game that translates to chase the pig in English. This game involves four players and allows for the user to play against three other users across a network.

The card game involves four players who are dealt thirteen cards each. The aim of the game is to score the least amount of points. Most of the cards are worth zero expect for the hearts and one of the few special cards. Each player must play the suit that is played. If they have the same suit as the first card played and this happens to be the highest valued card of the suit played, this player wins. The player who wins the hand decides what suit to play, and the first turn is decided upon by the player with the two of spades. The game ends when a player reaches the score of minus one thousand and they are then declared the pig. There are different values for different cards, for example the queen of spades is called the pig and is worth minus one hundred points while the jack of diamonds is called the goat and is worth plus one hundred points.

For this project I have decided to make a network chase the pig game. This will involve four separate players playing against each other on a computer or on their android phone. The idea is to create a cross platform game.

The project will be coded in java due to it being the language I am most familiar with and it has great network capabilities. It is also used in the making of android apps. The game will feature a client and a server. The server will let the clients know about changes in the game and the client will present a graphical user interface to the user.

**Chapter 1: Introduction:**

**General Information:**

This project is based on a Chinese card game that translates to “chase the pig” in English. It is a game that involves four players. The aim of the game is to score the least amount of points. Each player is dealt thirteen cards and the player with the two of spades begins the game. The player that goes first has the choice to play whatever card they like. The rest of the players must play a card of the same suit unless they do not have a suit of that kind. The player who has the card that is the highest of the chosen suit wins the hand.

Certain cards are worth a certain amount of points. Most of them are worth zero. The majority of the hearts are worth some negative number apart from two to four. So, if you win these cards your score will decrease. Then there are a few special cards. The Queen of Spades is called the pig and it gives you minus one hundred points. It is a card that no one wants to win. Then there is the Jack of Diamonds or the goat, which gives the player plus one hundred points and the ten of Clubs which multiples the players score. At the beginning of the game the players are allowed to declare if they have a Queen of Spades, Jack of Diamond, ten of Clubs or an Ace of Hearts which then doubles the values of these cards but allows for other players to know who has them. When a score of minus one thousand is reached the game ends, the person who reached this score loses and are then declared the pig.

**Card Score**

Queen of spades (pig) -100

Ace of hearts -50

King of hearts -40

Queen of hearts -30

Jack of hearts -20

10, 9, 8, 7, 6, 5 of hearts -10 each

4, 3, 2 of hearts 0 each

Jack of diamonds (goat) +100

For this project I have decided to recreate this game to play on your pc or android phone. The idea is for people to be able to play against their friends though a network. I have decided to create this game using Java. For this project I will be using a client server model. So, I will be using java socket programming. For the graphical user interface, I will be using javaFx. I decided upon javaFx over swing as it has a few advantages over it. Finally, I will be using Android to develop the final version.

**Motivation:**

I wanted to make this game for several reasons. One of which is the fact I have always enjoyed playing card games as they are quite entertaining. I also thought being able to play friends across different devices as it would be an exciting feature.

I wanted to do this project as I thought it would greatly improve my programming skills. I was also looking forward to working with client and servers since I have been playing online games for many years, I was excited to be given the chance to create my own.

Another motivator for this project was that I would be given the chance to create my own graphical user interface. I was really looking forward to this as I wanted to create my own design that was aesthetically pleasing.

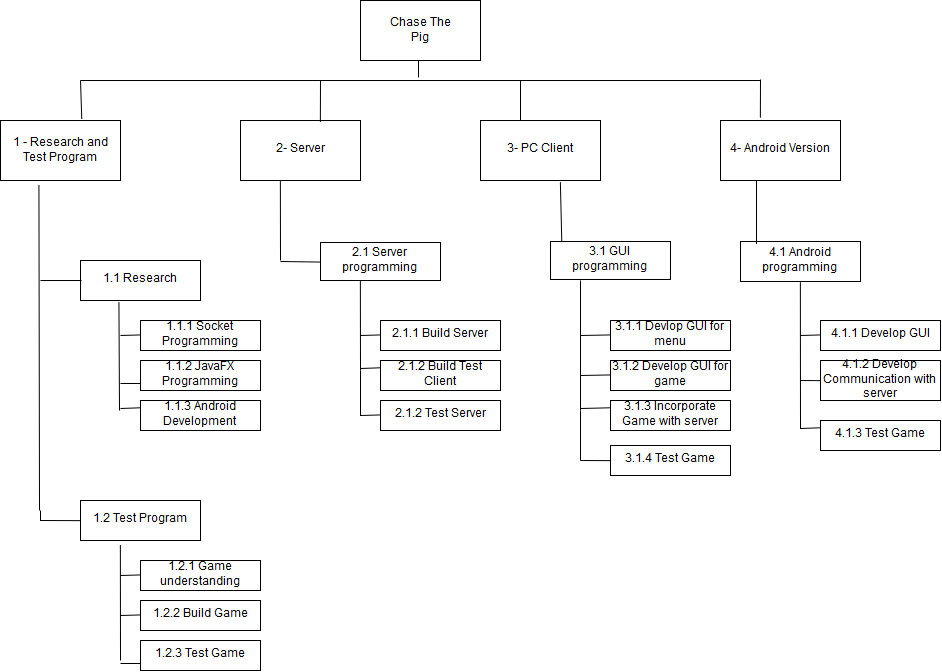
The final factor was getting to create an android application. This as very exciting for me as smart phones are such a huge sector in today’s world and being able to create games and applications for them was always a huge interest of mine.

**Objectives:**

My objectives for this project include:

1. **Console Game:** My first objective for this project was to build a console only version of the game that had no network capabilities. This was a great starting point as it allowed me to get a handle on how the game works and build a foundation for the project.
2. **Server:** For this stage I built upon the text version by making a server that had four console-only players. This helped me decide how the client and server would interact and start making the API that the game would ultimately use.
3. **JavaFx Client:** This objective here, was to implement a graphical user interface that the user would be able to interact with to play against each other in groups of four. This lead me to design an interactive GUI that positioned players in the correct order and displayed the game in a format that the players could understand.
4. **Android Client:** The final stage of the project was the development of the android version. For this section, I had a lot of the work done. The main point here was learning the android development process.

The following is a work breakdown structure for the game:



Figure

**Report Structure:**

The report is broken down into a number of sections:

1. **Research:** This section discusses how the areas that were researched for the game.
2. **Development:** This part of the report talks at length about the technologies used and their implementations. It covers the main areas of the project and how certain parts of it were achieved.
3. **Results:** This chapter discusses the program structure and the different classes in each part of the project.
4. **Conclusion:** Covers the outcome of the project.

**Chapter 2: Research:**

**Existing products:**

In beginning my research, I found a game called ‘Chinese hearts’(2-person studio) on the google play store. I initially played this game to better understand the rules of the game and to see how their user interface was laid out. While this game is similar to mine, it uses artificial intelligence for the player to play against rather than play against other people. From this game, I got a few a few ideas on how to lay out the game on the graphical user interface. From this game I decided how I would lay out the cards that have been played for all the players to see clearly. I also got some ideas on how to show the players that they must play cards from the suit that must be played. Overall, I found this application useful to see how this game should be approached and to show me how it would appear on the android version of the game.

**Exploratory programs:**

For the initial stage of this product, I decided to develop a text-based version of the game. While the game itself is unplayable due to the fact that the players can see each other’s hands fairly easily and it is difficult to play from the one computer, I found it to be very helpful.

This program allowed me to flesh the game out into some code and decide what was the best way to do so. It also helped me form a foundation for the rest of the project to build upon. From this program, I made the decision to make a network version of this as to make my first version of the game that would be playable to some extent.

For the next section I created a network version of the previous example. This part of the project greatly improved my abilities of working with sockets and threads. I learnt how to send information across the network in a manner that the client could understand and how to use threads so that the server can run smoothly. So, I created a simple application programming interface was developed for communication between player and server.

**Academic Research:**

For my research, I have read a few books to get a better understanding of the technologies I will be using. These books include:

Davison, A (2009) *Killer Game Programming: Java Gaming and Graphics Programing 4th edition*

This book was helpful as it has some interesting programming examples for games. It had some useful chapters on imaging, sound and some networking basics. Since it is a few years old the book mainly used swing for its graphic user interfaces, but I found it to be a helpful guide.

Harold, E (2013) *Java Network Programming: Developing Networked Applications 4th edition*

This book features an array of useful topics in relation to network programming. Its second chapter talks at length about the different streams used in socket programming, which I’m sure will be of great use for when I’m developing the android version. As to how I should send the data across the network to different systems. It also has great information on threads and has many chapters dedicated to sockets, overall a very informative text.

Liang, Y (2014) *Introduction to Java Programming 10th edition*

This is a book I would recommend to anyone interested in learning java. It is a book I have returned to many times in throughout my time in collage. This edition of the book talks at length about javaFX and overall is a brilliant guide to anyone developing in java. There are so many great code examples and it is very easy to get information out of.

Philips B, Stewart C, Marsicano K, (2017) *Android Programming The Big Nerd Ranch 3rd edition*

This gives you a step by step guide on how to build your different android applications. I thought it was hugely helpful to get my head around the different concepts in android development.

**Implementation Technologies:**

Why Java?

Java has been around since 1995, it is derived from c and c++, it is designed to be able to “write once, run anywhere” (Langley,2002). This means that a developer can write the code once and run it on any machine. This is done by java virtual machine, since the code is compiled into bytecode it can run on java virtual machine regardless of the system used (Liang,2014). Since java has been around for so long it has a large collection of features for developers to rely on.

I decided to choose java because it is the language in which I am most familiar with since I have been using it throughout my course. It also has great network capabilities and is more useful in that regard than a lot of other languages. It is also the main language used in the development of android applications making it extremely suitable for this project.

Why JavaFX?

JavaFX was released in 2008, it is a software platform that is intended to be the standard for graphical user interface library for java.

I decided to use JavaFX in the development of my game for several reasons. JavaFX is intended to be a replacement for swing in the making of graphical user interfaces. While currently both are still in use, I thought it would be more within my interest to develop my application in the newer version since it should be around for a longer time period. It also has a variety of graphical user interface with many different components such as layout panes etc. (Liang, 2014)

Why Android?

Android Inc was founded by Android Inc in 2003 and acquired by google in 2005. The first device running Android was released in September 2008 (FAUguy, 2011).

Androids are the most popular smart phone OS in the world right now (IDC, 2018). With it being this popular it shows that it is an obvious choice for developers. I also own an android phone myself, so I used it to test my game throughout the process.

Why Netbeans?

Netbeans is an integrated development environment for Java. NetBeans started as a student project (originally called Xelfi) in Czechoslovakia, now the Czech Republic, in 1996. It became a part of sun micro systems and then in 2010 it became part of oracle. It is open source and a flagship IDE for java. (Netbeans 2018)

I used Netbeans IDE for the development of my server and javaFX client. Mainly because I had used it before and it also had support for javaFX.

Why Android Studio?

Android Studio is the official IDE for android. It was developed by google and Jetbrains. (Ducrohet 2013)

I decided to use android studio because it is the official development IDE from google. I did find reports that it ran slow on some computers, but I did not have any issues with it, so I decided that it was perfect for me.

**Chapter 3: Development:**

In this chapter I discuss the details of what I did in the project. Mainly the problems I encountered and how I overcame them.

**Design:**

When I started the project, I began by doing some research on areas that I am going to be using throughout my project. Then I started doing some screen mockups to show what the final game will look like.



Figure

As you can see for the first image I showed how the user would be able to see the cards that have been played. For the second image, I showed how the players would differentiate between the cards they can play and the cards they can’t. I then showed the original layout for the main menu and what would happen when a player takes the pig card.

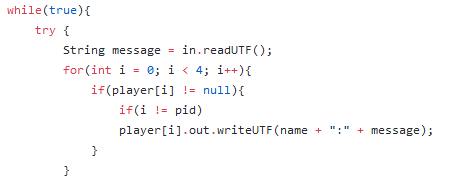
This was interesting to look back on as the game did not turn out looking like this in the finished product. One of the pieces missing from this design were the user’s ability to display their exposed cards. The final product also did not have any animation for the pig card.

Overall, I think this design really helped get some idea of what the final product would look like.

**Java Sockets and Multi-Threading:**

After I finished the console-only version of the game that had no internet functionality, I decided to start with creating a network console version of the game. Since I had no experience working with sockets before this took me quite some time.

I decided the best way to begin was to develop a single threaded server that completed a simple computation and sent it back to the client. This then lead me to build a chat server that was multithreaded and allowed for communication between multiple clients. In this program the user was initially asked for their user name. This would then be recorded so that every message they send after this step, would be edited so that all clients receive a string with the client’s name and message. The server was built so that each time a user was accepted by the server, a thread would begin that had the responsibility of receiving and sending messages to the user.



Figure

As you can see this server relied on there being four users but gave me the starting point to begin developing my game.

For my first task I decided that I would design the server so that when four players joined, the game would begin, and all players would receive their cards. One of the things I was overwhelmed by in this section was the amount of different options available for a programmer to send data across the network. This lead to great confusion and difficulties throughout this section and as I made the game. With so many filter streams to choose from I initially planned to use several different forms of these within my game. (Harold, 2013) This lead me to some problems. I first attempted to build the program so that it sent an array list of cards to each user for them to use as their hand. For this I attempted to use an object output stream. For some unknown reason I could not get the last player to receive their hand. In hindsight, I believe it was related to the fact that I was creating the game object as a sperate thread to players.

This lead me to re-evaluate the situation and attempt to re-build the program but instead use the runnable interface rather than extending the thread class. During this, I decided that it would be just as easy to send the cards as a string and have the client retrieve the numbers from this.

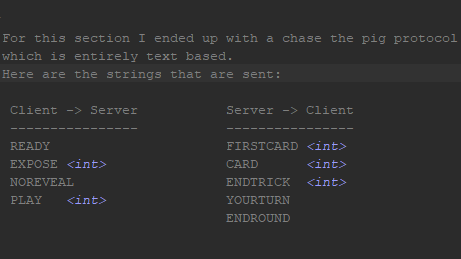
In the initial version I had also attempted to create game as a separate thread to the players. By rewriting the first class I soon realized that doing this was completely unfeasible. It was here that I realized that game should be a shared object between each set of four players. Each player would then be given a chance to interact with the game object during their turn and each player in turn would be updated about changes made in the game.

With my two main difficulties in this task overcame, I had a server that had the ability to deal cards to the players and to create a game for each four players that joined, and the seeds were planted for me to develop my protocol.

**Application Programming Interface:**

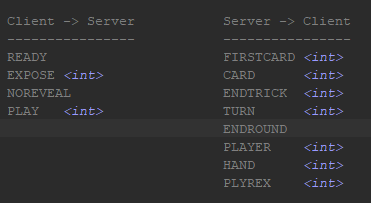
The next stage was to implement the gameplay. So, I needed to decide how my client and server were going to interact. Since I had overcame my last issue by sending strings across the network, I decided this should be the format of all the communication that is sent and received to the server.

For this part I needed to consider what information the client would need to know and what it does not. After much exploratory programming I realized that the client should be able to know what the first card in a trick is. This was the case so that the client can find out what cards they can play. Other necessary information included played cards, their turn and the end of round. This eventually lead me to develop this protocol.



Figure

This initially version left out some crucial details. For example, the player didn’t receive their player numbers and the exposed cards were sent as a long string with details of what was exposed. For the graphical user interface, I made these changes to it.



Figure

The <int> means that a trailing integer is placed onto those strings

The client only has four pieces of information it can send.

1. **READY**  -Tells the server the client is ready to be dealt their cards.
2. **EXPOSE**  -Tells the server what card the user decided to expose.
3. **NOREVEAL** -Tells the server they aren’t exposing their card.
4. **PLAY** -Tells the server what card the client has played.

After the four ready messages and four expose/noreveal messages are sent to the server. The server replies and sends the clients a string of numbers that represent the cards.

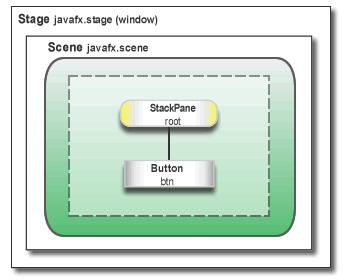
The server has the ability to send these five options.

1. **FIRSTC** -Tells the player a card is played and that it the first card of a trick.
2. **CARD**  -Tells the player a card has been played.
3. **ENDTRICK** -Tells the player the trick is over and who is the winner.
4. **TURN**  -Tells the player the whose turn it is.
5. **ENDROUND** -Tells the player the round is over.
6. **PLAYER** -Tells the player their player id at the start of the game
7. **PLYREX**  -Tells the player the card that is being exposed and the player id
8. **HAND** -Gives the player a string of numbers that represents their hand of cards.

With this information a developer could potentially design an artificial intelligent player that players could play against. All they would need to do is receive the data and send it in the correct format. I currently have message receiver and message sender classes that without much editing could become the basis for a project such as this.

**Creating a JavaFX Main Menu:**

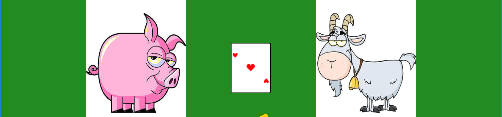
In creating my javaFX GUI, I used a mix of oracles online guides (Oracle, 2016) and the introduction to Java programming book. (Liang, 2014). The first few details that I learnt about JavaFx were the stage, scene and the different types of panes.



Figure

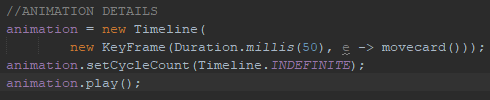
So, as you can see from the diagram above, a Scene is within a Stage and the Panes are within the Scene. The stage represents the window which forms your application. The Scene represents what the user views within this window. Panes then describe the layout of which you can place your buttons etc. For example, this image uses a stackpane which stacks your images or buttons on top of each other.

To gain a better understanding of javaFX I decided to start my development process with the main menu of the game. For the game menu I decided that I would use the four image views and then three options for the user. The first thing I decided to do was to place the three images in a row for the user to see. To do this I placed the image views into a Hbox which is a horizontal layout pane, so the images are placed one after the other. The following image displays the Hbox pane and it images.

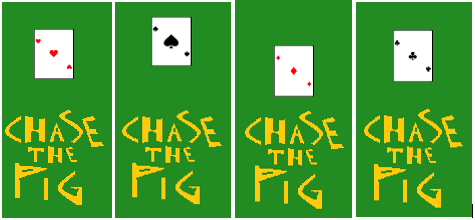


Figure

After this I decided to make the centre icon into an animation. The JavaFx animation import made this quite easy. All I had to do was create a Timeline object with a KeyFrame object specifying how often the animation needed to be called. The method it then called then updated the position of the card object, changing the direction of the card image when it reached a certain height and its image would change each time it went up or down.



Figure



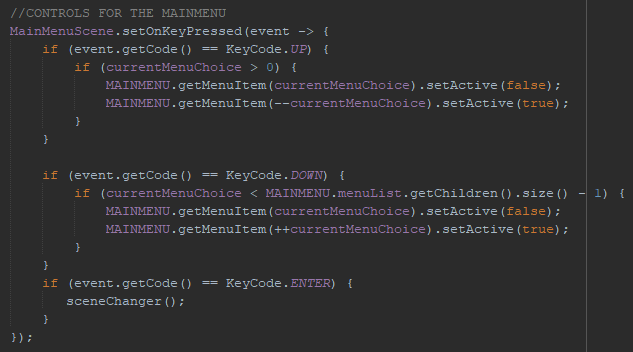
Figure

From here I implemented the main menu options of the game. To this I decided the user would be able use the up and down arrow buttons to scroll through the menu and the option that the user was on would be displayed differently to the other options.



Figure

For this to work I had to tell the main menu scene what needed to happen every time a button was pressed. Each menu item was placed into a Vbox which is the vertical layout pane and the menu item consisted of a Hbox with text and rectangle shapes. An integer named current menu choice stores their choice and each time that menu choice is changed it sets the text colour to white and makes the rectangles visible.



Figure

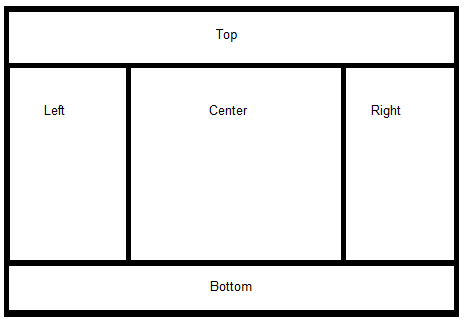


Figure

This lead me to the finished menu. The chase the pig logo was designed by myself. Then the goat and the pig were taken from <http://moziru.com>. The card image was an edited version of the ace cards found in the gameplay section.

**Using JavaFX Layout Panes to Create the Game:**

When making decisions on how I was going to utilize javaFX’s layout panes I looked at an article, I found on tutorialspoint.com. (tutorialspoint, 2018). From this article I decided that the BorderPane layout perfectly fit all the necessary components I needed to build the game.

****

Figure

The figure above represents the BorderPane. As you can see it is split into several different sections. For my game, I decided using this layout I could split the game into five major components. The left, top and right tiles were used to represent the other players in the game. So, each tile would reveal the player number and display the cards that they decided to expose. The center would be where the cards that are played are displayed and the bottom where the user’s cards would be displayed.

**The Application Thread and the Task Thread:**

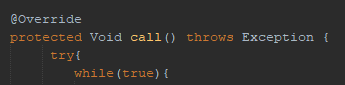
In the development of my server I created the console client without a GUI. These clients they were created with more of an intention of testing the game and were a single threaded player that waited until it received information from the server. When I initially attempted to make the game I first tried to create the game as a single threaded format. I soon realized that this wasn’t going to work as the game kept crashing. Then I decided to use thread like I did in the development of the server but soon came to conclusion this wasn’t going to work either. It wasn’t until I found some information online that I was able to overcome this problem (Oracle, 2012).

When a javaFX program is run the GUI runs on the application thread. So, any changes that are made to the GUI must be done on the application thread for them to appear to the user. Since the application thread is also waiting for events or input from the user it is best to use a background thread for long running tasks so that the program doesn’t become slow or crash.

When you create a background thread that must eventually communicate back to the application thread, you cannot simple create a runnable object and start a new thread. This is because it is error prone. Luckily you can use the javafx.concurrency package. Inside this package you find the task class.

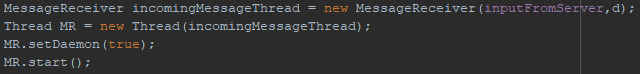


Figure



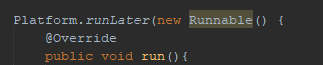
Figure

To create a task, you must extend the thread class. Then you must override the call method which is similar to the run method that is used in a runnable object.



Figure

Then as you can see the task object *MessageReveiver* is created and passed to the thread object and started normally. The set daemon function here lets the thread stop when the application stops running.



Figure

Finally, when the thread wants the application thread to perform something it uses ‘Platform.runLater’ so the application thread can perform the task on the GUI. Once I had figured this out I ready to start receiving data from the server and representing the data on the GUI.

**Game Controls:**

With the layout for my game achieved and my new found understanding of the application thread, I began developing the controls. The first thing that was required was the cards themselves. When four players joined the match, the server would send the cards in string format. The *MessageReceiver* object would then decipher this into just a string of numbers, then the *Dealer* object would take the thirteen different numbers and create a card item object for each. The card item would then use the integer that its passed to determine the card image and it would be added to a hbox pane inside the bottom pane or *Handpane* object.

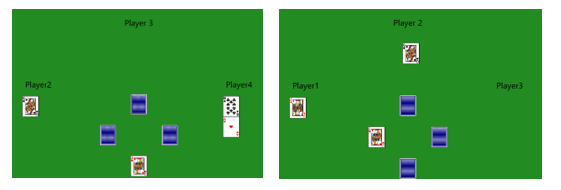


Figure

The cards were named from 1.png to 52.png. Each card item included an arrow above it and the arrow was made visible by the current menu choice like how the main menu functioned. Then certain valued cards such as the pig card had some extra text that questioned the user if they wanted to expose the card. The user could do this by pressing the enter key to expose or space to keep hidden. Once the expose section was finished the user could send the card to the server if it was a valid move and the card would be deleted from the Hbox pane.

**Player Management:**

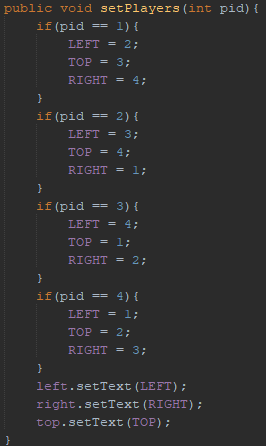
The next thing that needed to be done was the player positioning. Since each player would have a different view of the game, the players needed to have a system where the players cards would appear in front of them for each player. On each player’s screen the cards played by a user would be in a different position.



Figure

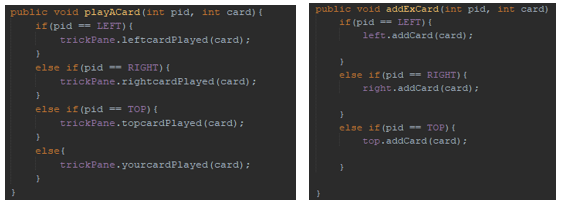
The images above display player ones view and player fours view. As you can see the cards appear relativly to each user. Each players exposed cards appear underneath their player id and the cards that they have played appear in front of them.

When the player joins the match and they receive their player id. This is then told to *Handpane* to display their player id and it is told to the player manager object. When the player manager receives this information, it places the id numbers into three variables named LEFT, RIGHT and TOP. Then it names each of the players their id.



Figure

From this stage on every time a card is received from the server, the dealer passes it to player manager who manages the positioning.



Figure

Once I had finished this I pretty much finished with the javaFX client. From here I added a few details such as the round and score and resetting the game variables for the next round of the game. I also added a text prompt that lets the user know whose turn it is.



Figure

**Android Activities and XML Layouts:**

Since I started the javaFX version with the menu, I decided to start the Android version the same way. Using some of the tips I learnt from the Android programming guide (Philips B, Stewart C, Marsicano K, 2017) I began to work on the menu.

Firstly, an activity is an instance of Activity, a class in the android SDK. It is in charge of managing user interaction with screen of information. A layout defines a set of UI objects and their positions on the screen. The layout is made up of definitions which are written in XML.

I didn’t have too much difficulty with the XML layouts as I have done a fair bit of HTML before, which in my opinion is more difficult to use in some ways. So, all I had to do on the activity was to create the button objects, connect them to the XML layout and set an on click listener to determine their functionality.



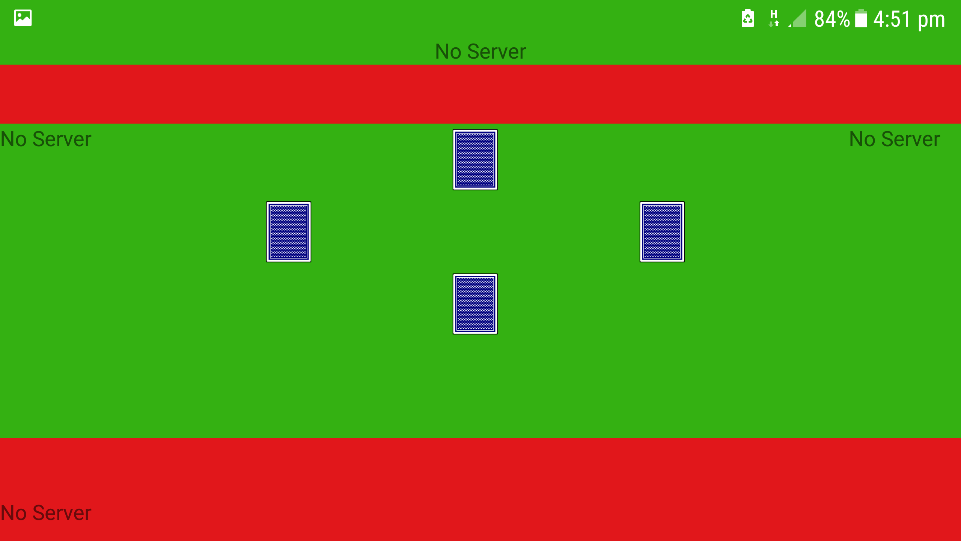
Figure

I also decided to create a similar animation to the menu that was on the javaFX version with the difference of the card moving from left to right instead of up and down. To do this I had to retrieve the screen size of the device and from that I found the centre of the screen. I then create a timer that completed a timer task that called the move icon function. This icon incremented the x position of the card icon to right or left, changing the card image when the card changed direction.

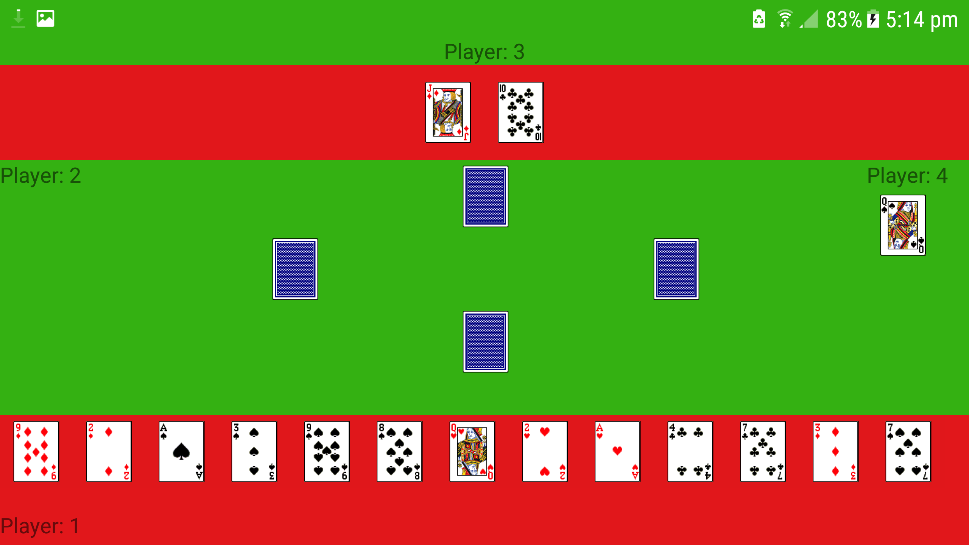
The buttons then just created a new activity apart from the quit button which just ended the game.

**Android Gameplay:**

To create the game play activity of the game, I used XML layouts to create the look of the gameplay.



Figure



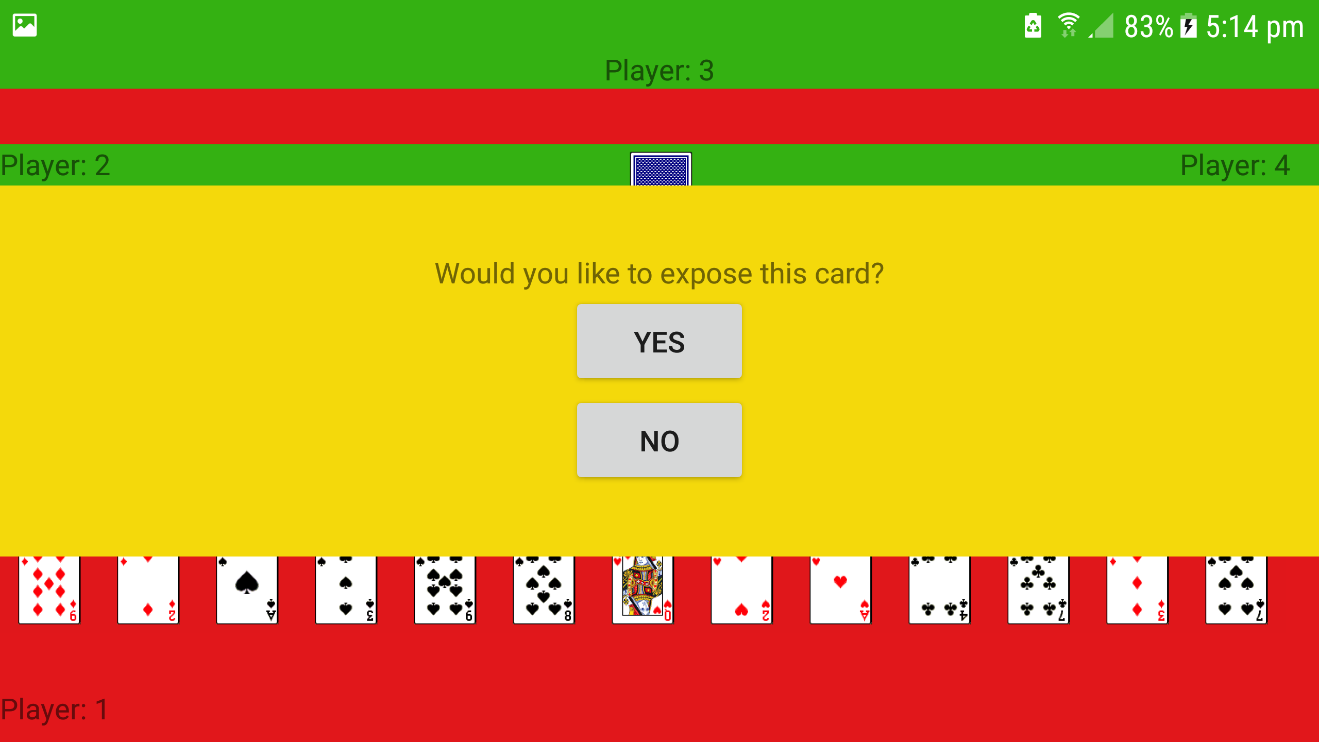
Figure

From javaFX version, there was a lot of code that I could use from it. A lot of the classes such as the *messageReciever* and *messageSender* objects needed very little change. The biggest change from the javaFX version was that the game was now touch screen and not key operated.

So, when the card was being added to the screen a button image was created with its card value and a series of if statements decided on what would happen when the button was pressed.

**Expose Pop-up Activity:**

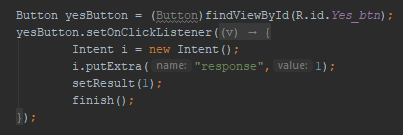
With the buttons now being touch instead of key input, I had the problem of how I was going to get the users decision on whether or not, they wanted to expose the card. To do this I decided to use a pop-up activity that didn’t cover the whole screen and asked the user what their decision was. I found information to do this in the android development guide. (Android, 2018)



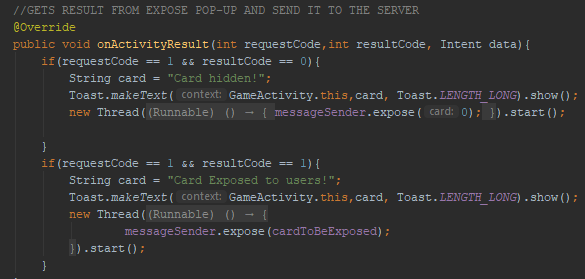
Figure

So now when the user was at the start of the game and the card that they could expose were highlighted yellow. When the user selects the card a pop-up activity like the one pictured above gave them a choice.

To do this I used an intent that would return information from the pop-activity to the game- activity when the activity finished. The activity only finished when one of the buttons were pressed.



Figure

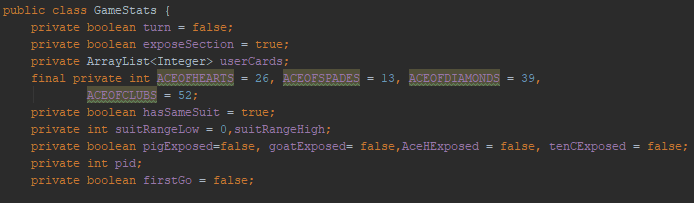


Figure

Then when the information is returned the message sender object sends a string to the server based on the user’s selection.

**Difference Between Android and JavaFX:**

As I said before a lot of the code just had to be refactored for the Android version. While doing that I did make a change for the android version that I thought were a bit of an improvement on the javaFX version. In the javaFX version I used two objects to make sure a move was valid these were *gamplaylogic* and *legalcommand*. The improvement I made on the Android version was that I made an object call *gamestats* that could be queried easily to find out game information. It also made it a lot easier to reset the game for the since all the game stats were found in the one place.

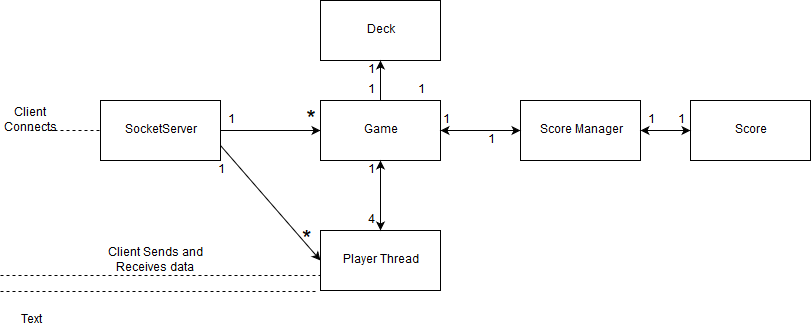


Figure

Other than that, the games work similarly, receiving and sending the same information to the server. I did initially have some difficultly connect my android device to the server but once I had changed my firewall settings and add a permission to the android manifest I overcame this issue.

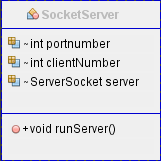
**Chapter 4: Results:**

**Server Program Structure:**



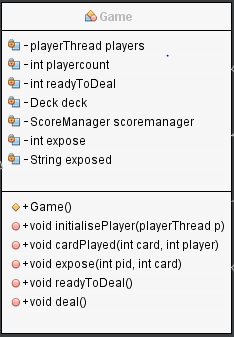
Figure

As you can see the server is made up of five different classes. The first of which is the *SocketServer.* This class is constantly waiting for players to join and when they do, it creates a *PlayerThread* for further communication and adds the *PlayerThread* to an array in the *Game* object.



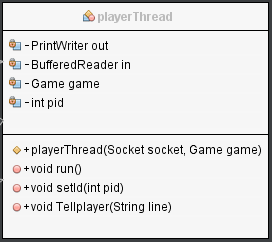
Figure

The next major class is the *Game* class. The *Game* object has a total of four players. The *Game* class job is to take information from the players and inform players about changes to the *Game*.



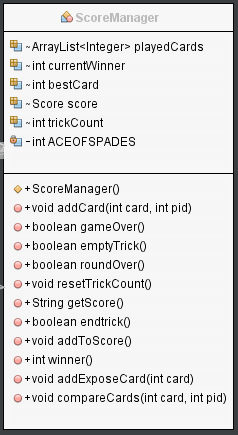
Figure

The *PlayerThread* then, oversees the receiving the data from the client. From the data it receiving it calls a function in *Game*. *Game* then uses the *Tellplayer* method to send information to the user.



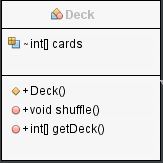
Figure

*ScoreManager* does what it says on the tin. It keeps account of the current trick and then when the trick ends it tells *Score* the winner and the cards that they won.



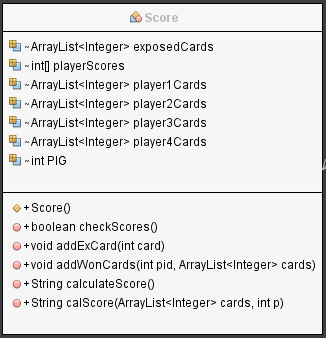
Figure

*Deck* is in charge of creating a random list of integers from one to fifty-two to represent our card deck.



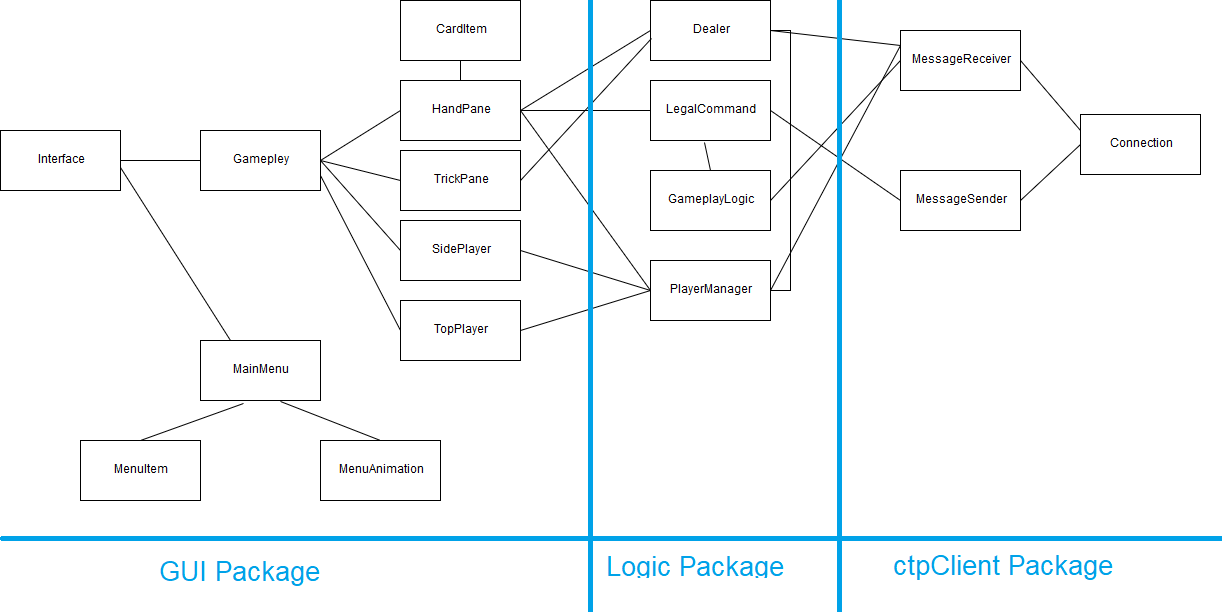
Figure

*Score* keeps account of all the cards that a player has won during this round so that it can calculate the score at the end of a round. It also keeps track of the players accumulative score from the previous round.



Figure

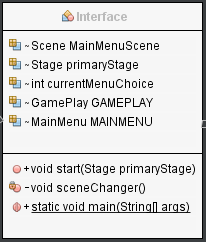
**JavaFX Program Structure:**



Figure

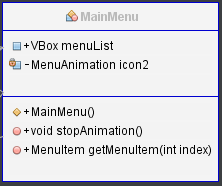
As you can see the javaFx Program featured a lot more classes. I’m going talk about them in package order from *GUI* to *ctpClient.*

The first class we’ll talk about is the *Interface*. The *Interface* sets up the stage when the game starts it also contains the buttons and has the two scenes in the game which are *GamePlay* and *MainMenu* objects.

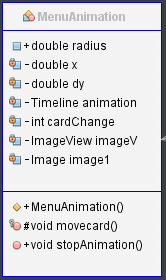


Figure

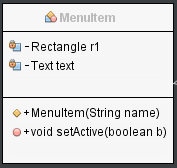
The *MainMenu* object is the first scene the user sees. It gives the users the option of playing a game and quitting, it also has two objects, one for the menu animation and the menu item that the user selects.



Figure

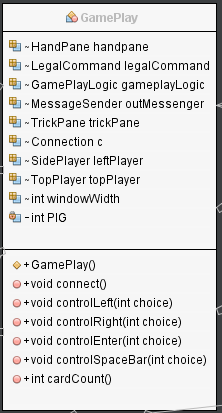


Figure



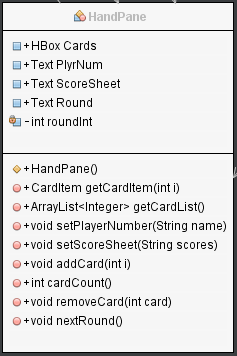
Figure

*Gameplay* is the Scene for the main gameplay of the game. It also starts the separate thread for the message *Reciever* object. The Gameplay Pane is also a border pane so it has five panes in that single pane. These panes are the objects *HandPane*, *GamePlayLogic*, *SidePlayer* and *TopPlayer*. With there being two *SidePlayer* objects.

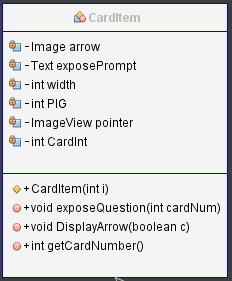


Figure

*Handpane* is where the user’s hand of cards is positioned. This is the part of the GUI that reacts to the user. (i.e. scroll through cards and select them). It also contains info on the players score and round. The *CardItem* then is what is used to display a single card. It contains an arrow and some text for the special cards. It allows the users to scroll though the *CardItem*’s by using the arrow keys and letting the user know which card they are on by making the arrow visible.

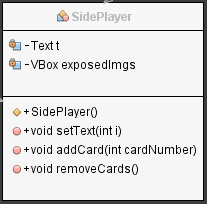


Figure

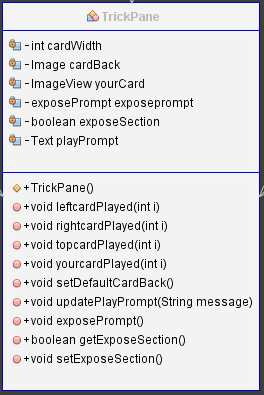
**

Figure

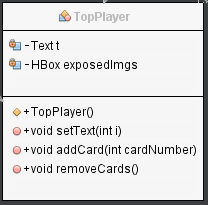
SidePlayer, TrickPane and TopPlayer are all GUI sections that are managed by playerManager. They display the users cards and their positions.



Figure

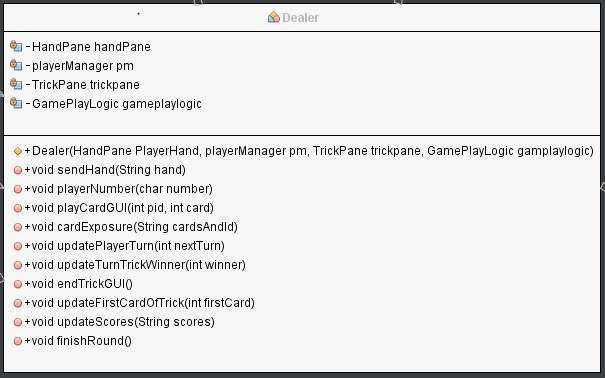


Figure



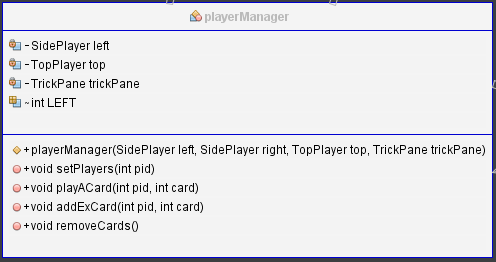
Figure

This is the *Dealer* class, it’s job is to take data from *messageReciever* which takes the data from the server and does what’s necessary for it to appear on the GUI. Weather it’s been dealt cards or passed on a played card to *PlayerManager*.



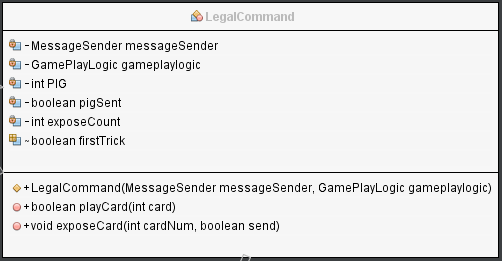
Figure

*PlayerManager* is concerned with the players positions, for example it takes the players id and card they’ve played and places it into the correct position.

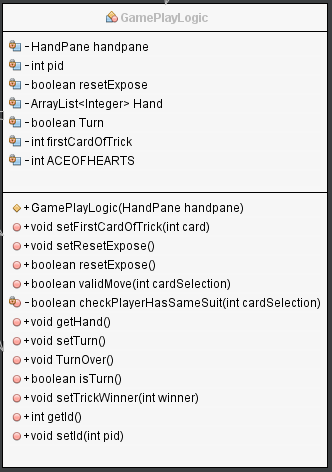


Figure

These two classes are concerned with the player playing the right card. For example, when a card of some suit is played *gamePlayLogic* is told about it and then *LegalCommand* checks with *gamePlayLogic* to see if a move is allowed.

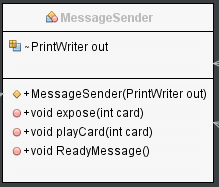


Figure

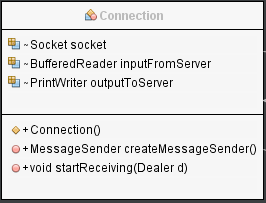


Figure

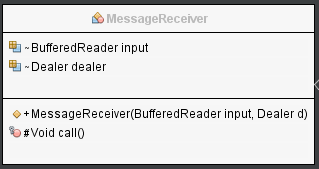
Finally, these objects are concerned with the connection to the server. *MessageReceiv*er takes a string from the server and takes out the necessary information so it can be passed on to dealer. *MessageSender* then formats messages to be sent to the server.



Figure

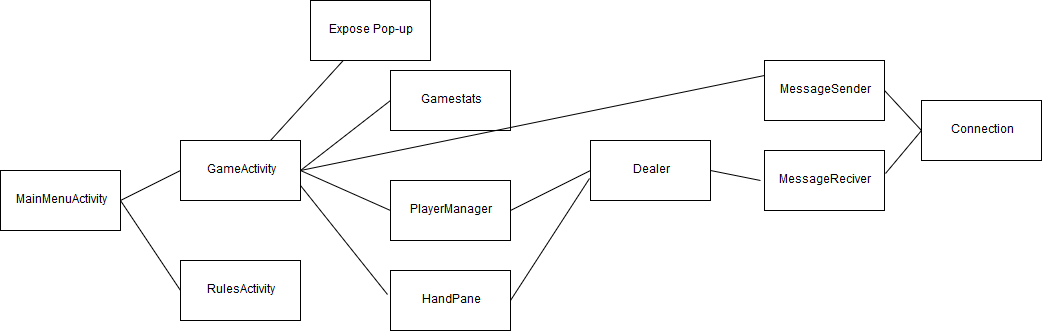


Figure



Figure

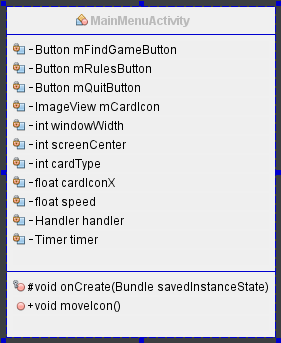
**Android Program Structure:**



Figure

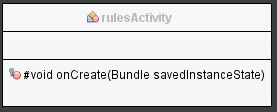
The android version turned out to have a few less classes then the JavaFX version.

The first class of interest is the *MainMenuActivity*. This class reads the layout of the XML file. It features a simple animation and gives the user the option of starting two different activities.



Figure

The *rulesActivity* just displays the rules to the user.



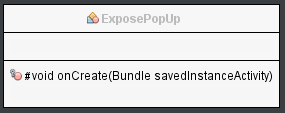
Figure

The *GameActivity* reads in the layout of the XML file and sets up the buttons that appear in the game. It also retrieves the result of the expose pop-up and makes changes to the GUI on the application thread.



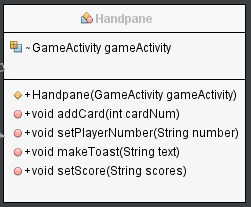
Figure

The *ExposePopUp* lets the player choose whether to expose their card or not.



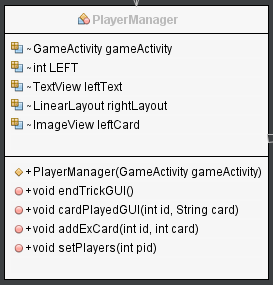
Figure

The *Handpane* object manages the users hand and sets the players id and their scores.



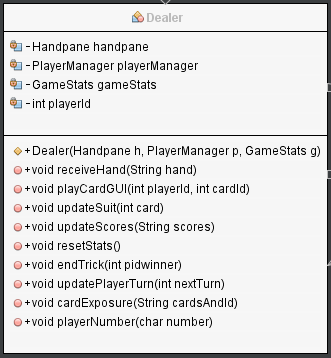
Figure

*PlayerManager* controls the where the cards are positioned on the GUI and it works very similar to the JavaFX version.



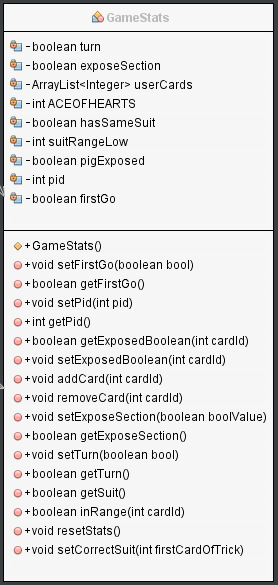
Figure

The dealer receives the information from *messageReceiver* and informs parts of the GUI that need to be updated and tells *gameSt*ats about changes in the game.



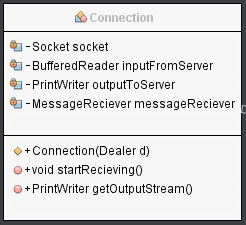
Figure

This class record the details of the game for the client, this allows the user to have select the right type of card and play by the game rules. Most of the variables are reset before a new round.

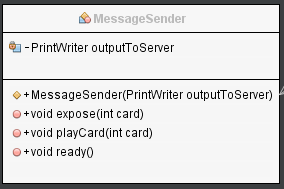


Figure

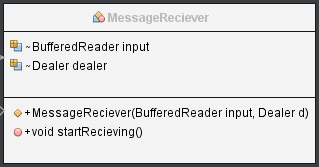
These classes pretty much work the same as the JavaFx version, taking and sending information in the correct format to the server.



Figure



Figure



Figure

**Chapter 5: Conclusion:**

**Future Development:**

I was quite happy with the finished product for this game. I feel that I achieved everything that I wanted. If I was to expand on this game, I would probably try to develop my own artificial intelligent player. I think this would be a very interesting route to take as I have the groundwork done with a playable game and a basic application programming interface ready to work with. I believe this would be a fascinating direction to take the project as nowadays A.I players are better than humans in games were both players know all the details of the game. Expect card games are an area in which machines are still not quite up to scratch. So, I’m sure I would learn a lot from continuing in that direction.

**Final Thoughts:**

When I started this project, I wasn’t that confidant in my programming skills but by working to achieve my set goal in this final year project, I feel like I’ve learnt a huge amount. By coming across so many challenges throughout the development, I’ve realized I have a lot more to learn, but I’ve also learnt that with enough hard work, practice and patience, I can get there. I am now looking forward to my next project and the many challenges that come with it.

**Figure Index:**

1. **–** Work Break Down Structure
2. **–** High Fidelity Design
3. **–** Socket Code
4. **–** First Protocol
5. **–** Final Protocol
6. **–** Stage Diagram
7. **–** HBox Diagram
8. **–** Animation Code
9. **–** Animation GUI
10. **–** Menu Items GUI
11. **–** setOnKeyPressed Code
12. **–** Main Menu GUI
13. **–** BorderPane Diagram
14. **–** Message Reciever Code
15. **–** call() Code
16. **–** New Thread Code
17. **–** Platform.runLater Code
18. **–** Game Controls GUI
19. **–** Player Management GUI
20. **–** setPlayers Code
21. **–** Player Management Code
22. **–** JavaFX Game GUI
23. **–** Android Menu GUI
24. **–** XML Layout GUI 1
25. **–** XML Layout GUI 2
26. **–** Pop-up Activity
27. **–** onClickListener Code
28. **–** On Activity Result Code
29. **–** GameStats code
30. **–** Server Diagram
31. **–** SocketServer Class Diagram
32. **–** Game Class Diagram
33. **–** playerThread Class Diagram
34. **–** ScoreManager Class Diagram
35. **–** Deck Class Diagram
36. **–** Score Class Diagram
37. **–** JavaFx Diagram
38. **–** Interface Class Diagram
39. **–** MainMenu Class Diagram
40. **–** MenuAnimation Class Diagram
41. **–** MenuItem Class Diagram
42. **–** Gameplay Class Diagram
43. **–** HandPane Class Diagram
44. – CardItem Class Diagram
45. – SidePlayer Class Diagram
46. – TrickPane Class Diagram
47. – TopPlayer Class Diagram
48. – Dealer Class Diagram
49. – PlayerManager Class Diagram
50. – LegalCommand Class Diagram
51. – GameplayLogic Class Diagram
52. – MessageSender Class Diagram
53. – Connection Class Diagram
54. – MessageReciever Class Diagram
55. – Android Diagram
56. – MainMenuActivity Class Diagram
57. – rulesActivity Class Diagram
58. – GameActivity Class Diagram
59. – Expose Pop-up Class Diagram
60. – HandPane Class Diagram
61. – PlayerManager Class Diagram
62. – Dealer Class Diagram
63. – GameStats Class Diagram
64. – Connection Class Diagram
65. – MessageSender Class Diagram
66. – MassadeReciever Class Diagram

**References:**

Davison (2009) *Killer Game Programming: Java Gaming and Graphics Programing* *4th edition*

Harold (2013) *Java Network Programming: Developing Networked Applications 4th edition*

Liang (2014) *Introduction to Java Programming 10th edition*

Philips B, Stewart C, Marsicano K, (2017) *Android Programming The Big Nerd Ranch 3rd edition*

2-person studio *Chinese hearts[online] Play Store*  <[*https://play.google.com/store/apps/details?id=com.twopersonstudio.games.gongzhu*](https://play.google.com/store/apps/details?id=com.twopersonstudio.games.gongzhu)*>*

*[Accessed 17 Oct 2017]*

Langley (2002) *Write once, run anywhere? [online] -ComputerWeekly.com Available at: <*[*https://www.computerweekly.com/feature/Write-once-run-anywhere*](https://www.computerweekly.com/feature/Write-once-run-anywhere)*>[Accessed 20 Nov 2017]*

FAUguy (2011) *Google’s Android OS: Past, Present and Future [online]* [*https://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future\_id21273*](https://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future_id21273) *[Accessed 25 Feb 2018]*

*IDC (2018) Smartphone OS [online]- IDC.com* [*https://www.idc.com/promo/smartphone-market-share/os*](https://www.idc.com/promo/smartphone-market-share/os) *[Accessed 25 Feb 2018]*

Netbeans (2018) *A brief history of Netbeans [online] -netbeans.org* [*https://netbeans.org/about/history.html*](https://netbeans.org/about/history.html) *[Accessed 25 Feb 2018]*

Ducrohet (2013) *Android studio: An IDE built for Android [online] – android-developers.googleblog.com*

[*https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html*](https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html) *[Accessed 25 Feb 2018]*

# Oracle (2016*) Java Platform, Standard Edition (Java SE) 8- [online] -oracle.com*

[*https://docs.oracle.com/javase/8/javase-clienttechnologies.htm*](https://docs.oracle.com/javase/8/javase-clienttechnologies.htm) *[Accessed 26 Feb]*

tutorialspoint (2018) *JavaFX-Layout Panes(Containers) – [online] – tutorialspoint.com* <https://www.tutorialspoint.com/javafx/javafx_layout_panes.htm> *[Accessed 5 March]*

Oracle (2012) *Concurrency in JavaFX – [online] -docs.oracle.com* [*https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm*](https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm) *[Accessed 5 March]*

Android (2018) *Android Development Guide* *– [online] – developer.android.com* [*https://developer.android.com/guide/index.html*](https://developer.android.com/guide/index.html) *[Accessed 20 March]*