**Multiuser Poker Game for Mobile Devices**

December 6, 2018

School of Computer Science

psymw6@nottingham.ac.uk

4276228

Supervisor:

**Milena Radenkovic**

School of Computer Science

pszmvr@nottingham.ac.uk

*Abstract*

In this dissertation we will present a mobile app designed for a game of poker aimed for ease of use for all types of end user. The app will consist of two components, the mobile app itself, designed to be deployed on Android devices, and a standalone server which the mobile app will connect to in order to play with other users. Throughout development we came to understand the difficulties in providing an engaging user experience for users of varying abilities as well as dealing with the concurrency issues that come with developing a multiplayer server designed to handle many games with multiple players at once.

Table of Contents

[**1 Introduction** 1](#_Toc5809351)

[1.1 History of Poker 1](#_Toc5809352)

[1.2 Project Introduction 1](#_Toc5809353)

[**2** **Motivation and Related Work** 3](#_Toc5809354)

[2.1 Motivation 3](#_Toc5809355)

[2.2 Related Work 3](#_Toc5809356)

[**3** **Description of the work** 6](#_Toc5809357)

[3.1 The work 6](#_Toc5809358)

[3.2 Functional Requirements 6](#_Toc5809359)

[3.3 Non-Functional Requirements 7](#_Toc5809360)

[**4** **Methodology** 9](#_Toc5809361)

[4.1 Key Features 9](#_Toc5809362)

[4.2 Development Process 9](#_Toc5809363)

[**5** **Design** 11](#_Toc5809364)

[5.1 Design Overview 11](#_Toc5809365)

[5.2 Prototypes 12](#_Toc5809366)

[5.3 User Testing 14](#_Toc5809367)

[**6** **Implementation** 15](#_Toc5809368)

[6.1 Backend Server 15](#_Toc5809369)

[6.2 Mobile Client 20](#_Toc5809370)

[6.3 Database 26](#_Toc5809371)

[**7** **Evaluation** 26](#_Toc5809372)

[7.1 Testing 26](#_Toc5809373)

[7.2 User Testing 26](#_Toc5809374)

[7.3 Requirements Evaluation 27](#_Toc5809375)

[7.4 Testing Limitations 29](#_Toc5809376)

[**8** **Summary and Reflections** 29](#_Toc5809377)

[8.1 Project Management 29](#_Toc5809378)

[8.2 Project Appraisal 31](#_Toc5809379)

[References 33](#_Toc5809380)

# **1 Introduction**

* 1. History of Poker

The origins of Poker have been debated at length by the historical community. It is reported that “In the year 969, the Chinese Emperor Mu-tsung played a game of “domino cards” with his wife” (Neal, 2011), which many historians have believed was the first ever game of the card game now known as poker. However, in the 1937 edition of Foster’s complete, R.F. Foster wrote “the game of poker […] is undoubtedly the Persian game of As-Nas” (Foster, 1937). Although the exact origin of Poker is disputed, the modern game of Texas Hold’em we know came to prominence in the 1970s with the World Series of Poker popularising it in a tournament setting (Anon., 2007) and it is this method of play that has become wildly popular in the 21st Century, both in-person and online formats being played and enjoyed globally.

* 1. Project Introduction

From the desktop computer to the mobile phone, the world today is becoming increasingly connected, with over “77% of Adults in the US owning a smartphone in 2018” (Pew Research Center, 2018) and studies showing that “91% of people with disabilities in the US own or use at least one of a cell phone, smartphone or tablet” (Morris, et al., 2016).

Despite this, the Play store presents a relatively sparse offering of mobile games with features designed to make them more accessible to users with issues such as sight or hearing impairment, such as TapBeats which was specifically created with the intention of “being accessible for users who are blind or have low-vision” (Joy & Ricaurte, 2011). This limited offering can be seen on this forum thread (Anon., 2015) which has a short list of aggregated games that support accessibility features on Android, mostly focusing on those that support the Android TalkBack feature for screen reading and does not appear to feature any variation of a Poker game at all.

This project aims to build a multiplayer poker application to be deployed on Android devices. It will be capable of connecting the application client to a host server that will allow multiple users to play against each other and include features to improve the game’s accessibility for new and less able users.

In order to provide this we will build a robust and secure server to handle the connections between users and process as well as another unit to process the core game logic and update the user’s client appropriately. This should be handled in real time and provide handling for instances in which clients abruptly or unexpectedly lose connection to the server so as to provide a smooth flow of gameplay for all users. Furthermore the server will implement select algorithms to ensure the fair and random shuffling of cards between hands as well as to efficiently calculate the outcome of each hand.

On the client side of this application we will face several Human Computer Interaction challenges involving creating a suitable interface for the user to interact with that will be clear and understandable for users of all ability, by implementing optional features that will, for example, providing support for the Android TalkBack feature to navigate the application or an alternative UI layout that emphasises the functional components of the app over a more aesthetically focused layout.

1. **Motivation and Related Work**
   1. Motivation

Mobile gaming has fast become one of the most profitable markets for gaming in the world, with projected revenue for 2018 marking the first time that more than half of all game revenues will come from the mobile segment and with revenue from mobile games alone projected to become a “100 billion dollar market by 2021” (Newzoo, 2018), from this we can see the stunning growth in popularity of mobile games and especially “the rise of multiplayer games which accompanied the advent of third generation network (3G) technologies” (Liu & Li, 2011).

It seems essential that in this new age where anyone can open their phone and expect to load up a multiplayer game and have it work almost anywhere that the game servers supporting this should be able to provide a robust and stable experience in order to retain a player base in today's competitive market. Sensor Tower even projects the Apple app store alone to “reach 5 million apps by 2020” (Sensor Tower, 2016).

As highlighted in the introduction, a large majority of people with disabilities use cellular devices (Morris, et al., 2016), however there is a significant disparity in apps designed to cater this audience.

These key points have convinced me of the need to develop more accessible games for Android and will be a key aspect alongside the creation of a robust and enjoyable multiplayer game of Poker capable of handling multiple game sessions connected together to provide an inclusive experience for all users.

* 1. Related Work

1. Zynga Poker



Fig. 1 Main Menu screen of Zynga Poker App

One of the most popular poker apps available on Android, with over 50 million downloads.

Key Features:

* Brief tutorial to introduce the interface
* Friends system which allows importing of friends from Facebook
* Ability to adjust the game speed and table size you would like to play at
* Has a tournament game mode for users to compete in
* Slot machine for earning currency – linked to real money currency purchases

1. World Series of Poker



Fig. 2 Main Menu screen of World Series of Poker App

Named after the annual Las Vegas tournament, this app has over 10 million downloads on the Play Store and offers a robust gameplay experience with an overall clear layout.

Key Features:

* An extensive, narrated tutorial
* A chat system when playing at a Poker table
* Options menu has the option to show current hand strength, as well as disable extra features like the chat
* Friends system with Facebook integration
* A levelling system and VIP system tied to the user’s profile and playtime
* Displays game stats on the user’s profile
* Different levels of tournaments, with increasingly high buy-ins unlocked as the user ‘levels up’
* Can invite friends to join your current table
* Slot machine for earning currency – linked to real money currency purchases

1. World Poker Club



Fig 3. Main Menu Screen of World Poker Club App

Created by CrazyPanda, World Poker Club has a slick layout with labelled buttons for clarity, creating a satisfying and easy to navigate user experience.

Key Features:

* After missing a certain number of turns the user is marked as ‘AFK’ (away from keyboard), and their turn will be automatically skipped, but not removed from the table, until they come back
* Basic gesture controls for gameplay (bet, check, raise, fold etc)
* Scratch card minigame – linked to real money currency purchases
* Customisation of table settings, i.e the number of seats and blind amount
* Manual browsing through a list of tables
* Chat system with prearranged messages

Overall Poker apps tend to share a lot of functionality such as a friends system, user profiles and levelling, and the ability to buy currency to keep playing. With this being the case this project will attempt to build a similarly structured app, but with a focus on the core gameplay – Poker, over pointless minigames and systems designed to entice you into spending money. Furthermore, we will attempt to build a functional interface with a simple and clear layout that is highly learnable for all types of users in order to distinguish from other apps which appear to focus more on aesthetics over a user-friendly interface.

1. **Description of the work**
   1. The work

This app is designed to function on Android devices and intends to provide a base game experience of Texas Hold ‘em Poker without the extraneous additions of features like real money transactions for currency which can help facilitate gambling addictions and instead creating a simple app the is easy to pick up and understand and be used by all types of users, including those who might have restricted ability such as limited sight or motor capacity.

* 1. Functional Requirements

1. **Platform:** The application will run on all Android devices supporting the minimum API level specified in its creation.
2. **Login:** Users will be presented with a choice of methods to login to the app.
   1. Users will be able to login with a Google account

Users will be able to login with a guest account

* + 1. Users will be able to link their Google account to the guest account

1. **Accessibility:** Users will have the option to enable features to increase the accessibility of the app.

The app will support the Android TalkBack feature to the best of its ability

The app will support an alternative layout mode which focuses on making the key gameplay elements as visible as possible (henceforth referred to as the ‘functional layout’)

The app will support gesture based controls

The app will support voice command controls

1. **Connections:** The app will handle connections from multiple clients in order to facilitate a Poker game between users
   1. The server will match users together from a queue to create virtual “Tables” where the game will be played and removing them from the queue
   2. The app will allow users to leave a table at any time, forfeiting any bet chips and returning to the main menu and disconnecting from the server Table
   3. The server will check users current connection status and:
      1. Remove users who have lost connection to the server from the current match
      2. Update connected users with information on the current game state
2. **Algorithms:** The app will implement algorithms in order to calculate the winner of each hand and to shuffle the cards between hands
   1. Durstenfeld’s improved Fisher-Yates shuffle algorithm will be used in order to shuffle the deck of cards
3. **Data storage:** User data will be stored appropriately by the server
   1. User login data will be stored in a database server side to validate login requests
   2. User game data will also be stored server side for security and stability
4. **Game functions:** The app will provide an interface for the user to make game decisions with that will trigger the relevant actions by the server
   1. Users will be able to take the standard actions permitted within a game of Texas Holdem like Check, Bet, Fold etc with only relevant actions being shown in the screen
   2. The server will receive each user’s actions and apply them to the current game state and update accordingly
   3. User’s will have a common virtual currency, which will be used to bet with in game and will be carried between game sessions
      1. Users will be granted an amount of currency at account creation
      2. Users will be able to win and lose this currency in hands
      3. Users will be provided with currency on a daily based, scaled with login activity
5. **Interface:** The app will provide different menus, implemented as “Activities” in Android that can be navigated between to perform certain actions.
   1. The app will provide a login screen for users when the app launches until they choose a method of login
   2. The app will provide a main menu as a hub to provide access to the options, friend, chat, and game help menus.
   3. The app will provide an options menu to adjust options pertaining to accessibility features, general game features, and connected accounts
   4. The app will provide a friend menu for viewing friend’s profiles and adding friends
   5. The app will provide a chat screen for users to talk in from the main menu and the in game Poker Table screen
   6. The app will provide a game help screen to display the rules of poker for beginners, as well as a reminder of hand values
   7. The app will provide a Poker Table screen where the actual game will take place between users
   8. The app will provide a user stats screen displaying statistics saved in the user’s profile data on the server
   9. Non-Functional Requirements
6. **Usability:** The app should feature introductory tutorials designed to introduce key features of the User Interface, as well as how to use the main accessibility options. Otherwise use should be straightforward even for users with low experience with similar apps using highly learnable buttons that are easily recognisable, and minimising the number of activities displayed to the user
7. **Reliability:** The server side implementation will secure the stored data to prevent unauthorised users accessing it, and will be built to be robust and handle numerous connections at once without failure
8. **Performance:** Communication between the client and server should be completed with minimal delay, and the algorithms implemented in the server for calculating hand value and shuffling cards should not cause any noticeable slowdown when handling requests or sending game data. Additionally the app itself should not present any noticeable lag to the user when performing actions such as switching activities in order to maintain a smooth and enjoyable user experience
9. **Implementation:** The app will be created primarily in Java in order to be deployed on the Android Operating System with a corresponding server built in Java to communicate with
10. **Methodology**
    1. Key Features
11. **Google account** **integration:** A staple of modern mobile gaming, account integration allows a smoother user experience, without the hassle of creating any new accounts and the convenience of being able to import their friends as well
12. **Accessibility**
    1. **Change layout mode:** An optional layout mode that focuses on delivering the main functions of the app, i.e. the buttons and removing the unnecessary menu and game graphics
    2. **Gesture control scheme:** The option to navigate between Activities using gestures and control the game with gestures which will aim to be highly learnable and an effective alternative control scheme for the game
    3. **Android TalkBack support:** A function designed to improve the user experience for users who have low-vision or are blind, TalkBack will be supported as fully as possible to be able to navigate and use the app
    4. **Voice command support:** Another form of control for the app, allowing users to interact with the app utilising voice commands, however is held at a lower priority due to the relative complexity of implementation
13. **Virtual-only currency:** The currency to be used in the game, which will have no real monetary value, and will only be possible to earn by playing the game, without the addition of any ‘micro-transactions’ in order to create a ‘pure’ poker game, without the extra clutter of so called ‘mini-games’ that seem to be prominent in the most popular of games and the enticement to spend real money
    1. Development Process

Although the original intent for this project was to adapt Agile methodology in order to utilise Test Driver Development principles and be flexible in changing requirements due to user testing, we discovered early on into the development process that Test Driven Development would not prove to be practical with the work schedule set out. This was due to the fact that the server and client were being developed separately, and not in tandem and as such in order to test whether each component was working as intended the tests written would need to use a dummy server or client to correspond with the component being tested. Essentially this would mean creating 2 servers and 2 clients in order to properly test functionality which would be impractical given the limited time frame for development and so the decision was made to steer away from Test Driven Development and to simply develop bother core systems in parallel. This led to building the corresponding components alongside each other and using these to essentially perform testing on each other, reducing the need for written tests with functionality being shown to work by correct communication between the client and server components being built.

Tests were still developed for certain components of the server, specifically supporting classes like the HandEvaluator class which is intended for use as part of the game logic of the server to determine the winner of hands played by users. As these components were purely intended for use server- side with no communication with the client the testing was fairly straightforward and helped detect and eliminate defects in relatively complex logic with ease.

The algorithm used within the aforementioned HandEvaluator class in order to actually evaluate poker hands is a simple brute force algorithm. The strength of this approach is the relative ease of implementation with there currently being no need to evaluate millions of hands at a large scale in the app build. The main issue with this approach would be the lack of scalability and would need to be overhauled entirely if it was a key focus. After researching into the use of algorithms such as ‘Cactus Kev’s’ 5 card evaluation algorithm and attempting to adapt similar methods involving the use of lookup tables and bit shifting in order to achieve the fastest possible computation of poker hands we determined there would not be enough development time to completely implement such an algorithm and so we had to settle for simply using brute force.

The Durstenfeld algorithm for shuffling cards in our Deck object is a concise and elegant solution that goes through the list of items and swaps the last entry that has not been chosen with a random entry until there are no entries left that haven’t been swapped. This solution produces an O(n) complexity which is optimal for the problem it solves.

1. **Design**
   1. Design Overview

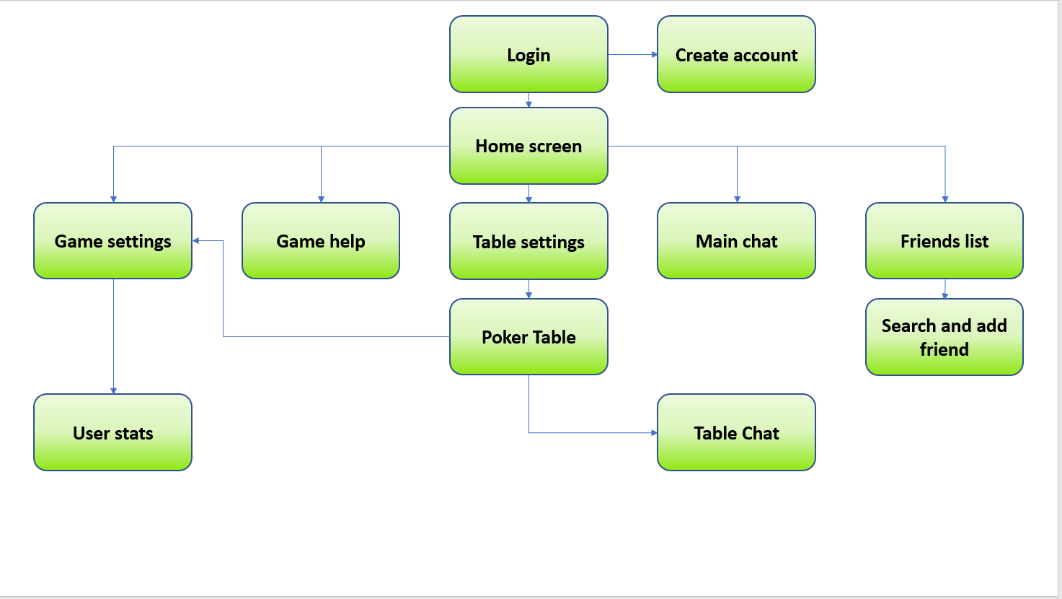


Fig. 4 App screen-relationship diagram

The core design of the project will be in two parts, following a client-server model. The application which will be released on Android devices will act as the client, and interface with the server, which will be developed separately.

The server will be created from scratch in order to have full control over the desired functionality of the system. The server will also have an attached mySQL database to handle user authentication for guest accounts and linked Facebook and Google accounts. This data will be stored online as if it is stored in the Android internal storage database provided it will have to be encrypted and could still potentially be tampered with if the encryption is not strong enough.

The server will have multiple responsibilities in communicating with the client. For all users the first thing it will do is add them to a separate handler thread which will authenticate their login ids and retrieve their profile. After this the client will be able to join a queue which will place them in games with other players and handle the relevant game logic in a separate thread during which the original handler thread will be idle. Users will be able to leave their game and communication will be returned to the original handler thread which can then re-add them to the queue or send them an updated user profile if the user wishes to view their profile.

The client consists of multiple Activities presenting information to the user and communicating with the server when necessary. These Activities are presented as screens to the user and will provide links to navigate between them (Fig. 4).

Development of the server will be done in Java as this will provide the easiest method of communication with the client which will be written using the Android Studio Java implementation and the objected-oriented design will facilitate the use of several key design patterns such as the Model-View-Controller pattern and the Command design pattern, the former to be used as part of the client and the latter as part of a communication protocol between both the client and the server.

* 1. Prototypes

Prototyping the User Interface commenced early in the project, in order to promote an iterative, user focussed design methodology. Low Fidelity prototypes were developed to this end and testing was used to gauge the potential learnability and usability of the prototype. The prototyping was software aided in order to add interaction so that users could get a feel for how the app would run and provide more relevant feedback.

The initial prototype demonstrated the overall layout of the main features of the app (Fig. 5 & 6). This prototype had attempted to make use of the space in the screen to only display information that was seen as important, with the links to other screens placed in the corner, out of the way of the main interface. The initial prototype did not have any designs for the optional ‘functional layout’ outlined in the Requirements section, Requirement 3.2, as it had only been conceived as a result of feedback from users.

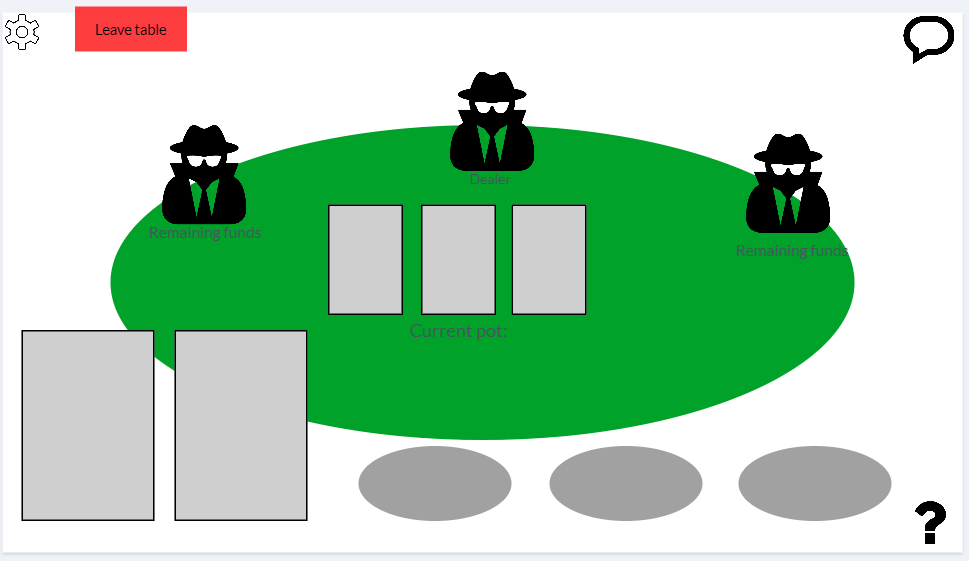


Fig 5. Table screen of initial app prototype

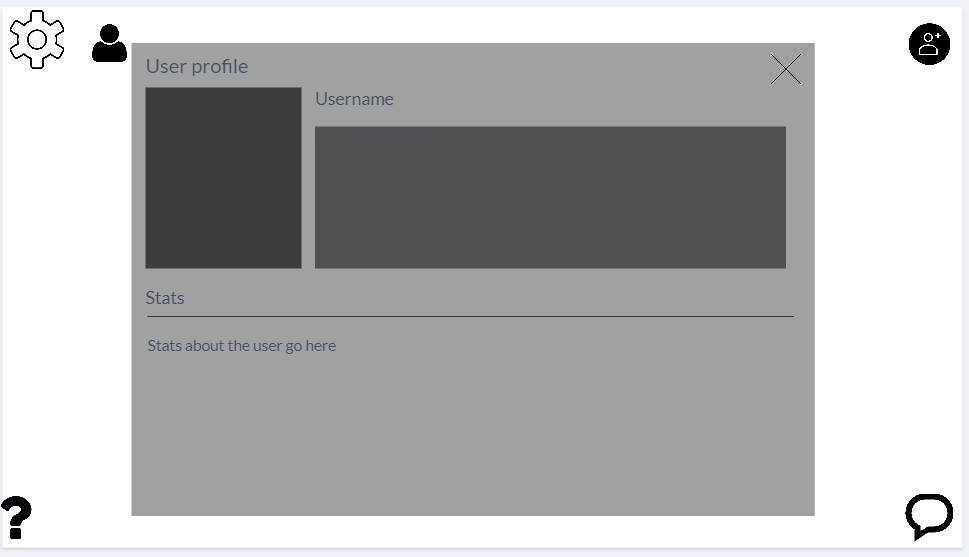


Fig. 6 User profile screen of initial app prototype

This prototype was changed after a round of testing and feedback as a result of the formative evaluation process. The 2nd prototype introduced the aforementioned ‘functional layout’ which had a higher focus on making every functional aspect (buttons) the main focus on the menu screens, and further increase their visibility in the game screen (Fig. 7).

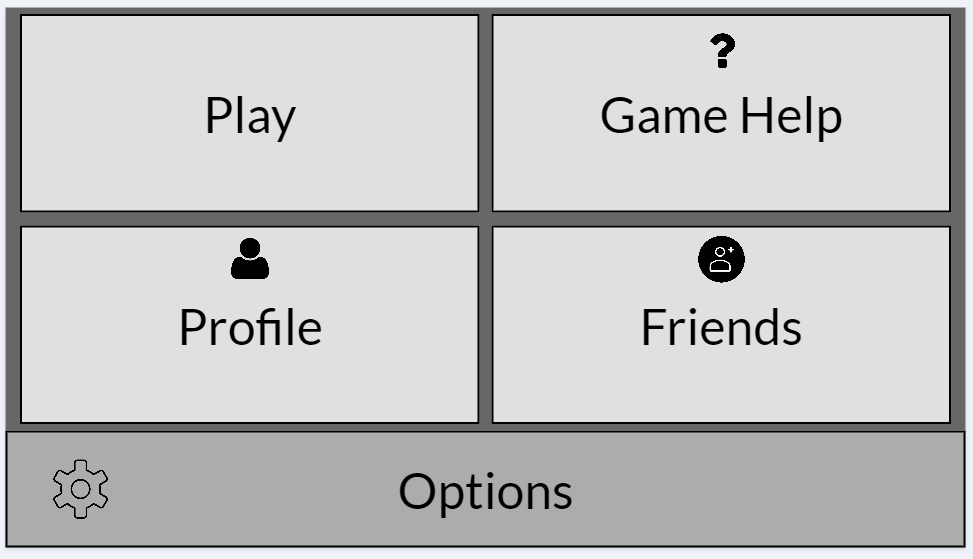


Fig. 7 ‘Functional layout’ Main Menu screen of prototype 2

This prototype was further refined to simply display text, without the associated icons as user’s felt they had no use when the function was already made clear with the text description.

An issue raised in the evaluation of this prototype was the lack of chat option in the functional layout, however we decided that the type of user that would elect to use the more visibility-centric layout is likely to either be visually-impaired or prefer to just focus on the actual game experience and as such the lack of chat visibility was not an issue.

Another major change in this prototype occurred in the normal layout of the main menu screen, with the icons that were initially in the corners representing links to different activities being moved to a navigation bar at the bottom of the screen, with a brief text description reminiscent of the design seen in Fig.3 in Section 2, which in particular was evaluated as being much clearer and easier to learn for new users who might be unfamiliar with similar technologies. Accompanying this was similar changes to the actual game screen, which also favoured larger buttons and clearer presentation whilst sacrificing aesthetic layout (Fig. 8).

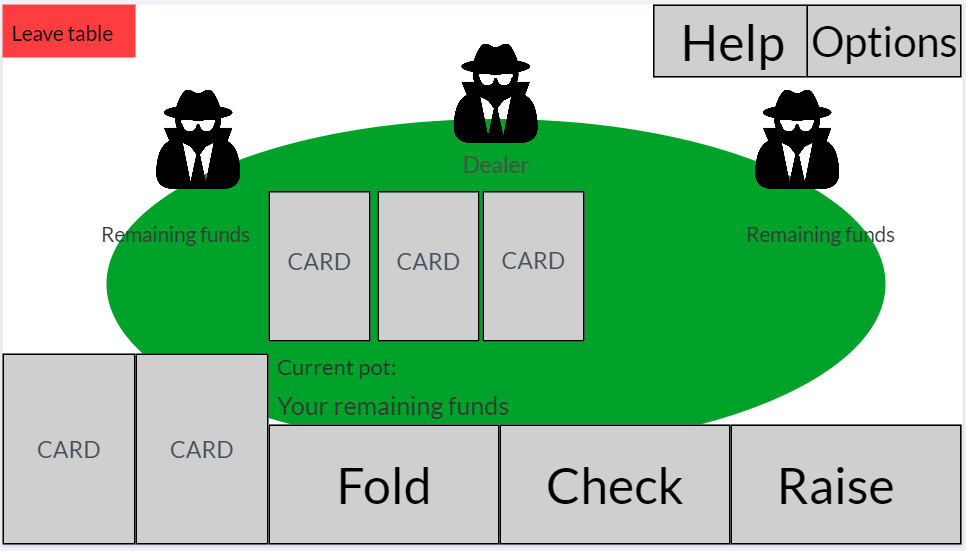


Fig. 8 ‘Functional layout’ Table screen of prototype 2

* 1. User Testing

Initial testing was conducted with a small user group of 3 people, in order to keep the feedback manageable and be able to review it quickly to implement changes to the prototype.

The test group comprised of 3 different types of user, the inexperienced, moderately experienced and experienced smartphone user. Testing proceeded in the form of a cooperative evaluation, and users were encouraged to share their thoughts as they navigated the interface, following a ‘think aloud’ protocol, from which notes were made. The advantage of this sort of testing is that the tester can see the user’s first reactions to the prototype and better understand the concerns and problems that might arise than if the user simply filled in a questionnaire.

The testing process involved a set of 3 tasks for each user, designed to allow them to interact with all the interfaces present in the prototype. The time taken to complete each task was noted and compared between users to understand how usable the system is. The user’s comments as they were performing the tasks were also noted to understand what users think during use. After the tasks were completed users were asked to give 2 ratings, the first for how obvious the steps that needed to be taken to complete each task was, and the second an overall rating on how usable the felt the system was.

Further testing will be conducted later in development of the front end in order to ensure that it is built to be as user friendly and bug free as possible. This testing will be both user testing as well as functional testing using JUnit as part of the Test Driven Development methodology.

1. **Implementation**
   1. Backend Server

**Language and dependencies:** The Backend Server was written using Java SE 11, being the latest release of the powerful Java platform intended for long term support by Oracle which provides security and bug fixing updates for years into the future which will be useful to reduce the need for refactoring with new Java releases such as the jump between Java 9 and 10 and will be useful for maintaining the project over time.

Application dependencies include the Google API for Java Clients (Google, 2019) which facilitates the usage of Google Account integration using tokens that can be verified through their API to retrieve user’s details and securely log them into their accounts. The other key dependency is the SQL Java Database Connection (JDBC) package (Microsoft, 2019) which allows for Java code to connect to MySQL databases, crucial for integrating the SQL database that is used for storing user accounts and their associated details.

**Database connection:** The server contains 2 classes encapsulating the core logic of accessing the database using the JDBC package. The base class, SQLDatabaseConnection consists of basic methods to connect and disconnect to the database server (Fig. 9), as well as a method to create SQL statements to be executed on the database.

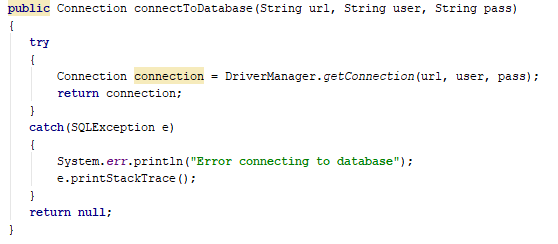


Fig. 9 Example method from the base database connection class

The second class uses inheritance in order to implement these basic connection functions and extends the functionality to create database queries that are specific to the server’s needs. This class serves as the interface object for server threads to access the database. This object allows for the abstraction of database communication in the main server logic as all the queries are handled within the class and all the handler need do is call the appropriate method, this is used every time a client connects to the server to retrieve their user details for example (Fig. 10). A separate query object is maintained for each handler thread and only connects to the database when necessary in order to reduce overhead making such calls.

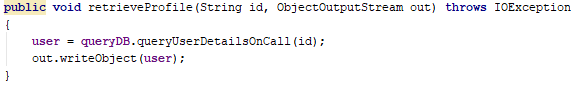
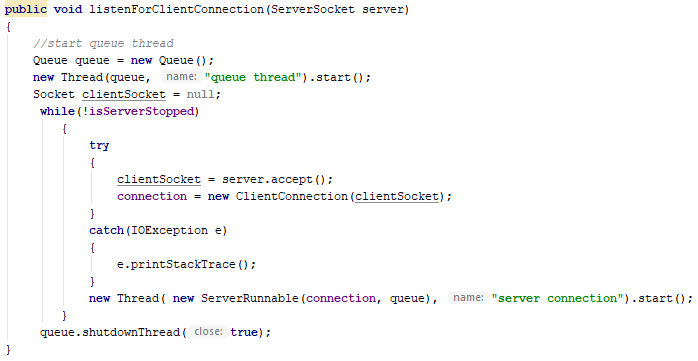


Fig. 10 Example method using the database connection object to retrieve a user’s profile details

**Handler thread:** The server handles new connections by continuously accepting new connections in a main thread loop and for each connection a new ClientConnection object is created which contains the socket and associated output and input streams that will be used anytime communication between the server and client is required for the lifetime of the connection and then adding them to their own thread handler by passing the ClientConnection object into a separate thread which will exists as long as the connection is not closed (Fig. 11)

Fig. 11 Accepting new connections to the server and passing them to their own handler thread

The handler thread is key in providing the client with requests outside of the actual game, these requests include retrieving the user’s details on login (Fig. 12), obtaining an updated user profile when viewing their profile, linking their guest account to a google account, and adding them into the queue to join a game. Since each thread operates independently for each user there are no concurrency issues with this solution however a drawback of this is that we were unable to prevent users from being logged in with the same account on multiple devices.

The handler thread maintains a copy of the user object throughout the lifetime of the user’s connection in order to reduce the amount of database queries needed to be made as they are much more costly than simply referencing an object for details, which means that the only time the database needs to be queried after the initial login is to retrieve new details in the case of the user requesting a profile update.

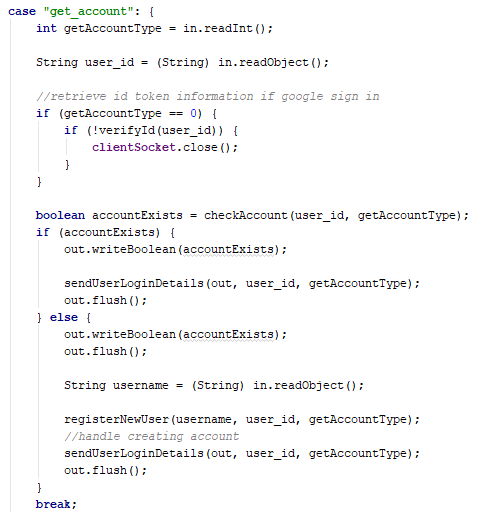


Fig. 12 Example of handler thread responding to a client request to retrieve their user account

The largest issue encountered with implementing this handler thread was to be able to smoothly remove and restore control to it when a user joined and exited a game. The issue that existed initially was the handler thread taking priority in using the client’s input and output streams even after they had been passed to the appropriate game thread. In order to fix this we stopped the handler thread from iterating over its main loop reading input from the client whilst the client was considered ‘in-game’. This produced a further issue that meant that after the client left the game the handler thread did not resume communication as it had no way of telling when the client was no longer ‘in-game’. The fix to this was to introduce callbacks from the game thread that notified the handler thread of the client leaving allowing it to resume the main loop for client communication.

**Game objects:** We created a number of support objects to be used by the game threads on the server that will serve for functions needed to play a game of poker. These serve to decouple things such as data structures and calculations from the core gameplay loop presented in the threads to improve readability and abstract functionality away from the game thread.

There are four core objects used server side to help implement game logic:

* Deck is a collection of 52 cards that can be shuffled and drawn from to simulate a real deck
* Hand Evaluator is responsible for calculating what hand a player holds and determining which hand is the winner from a given set. The main issue for testing this was to ensure that all types of hands were covered, an important bug that was uncovered due to testing involved the interaction of the evaluator with an ace low straight as the ace card was considered as value 13 in the Card object definition, and so it was not properly evaluating the A-2-3-4-5 hand combination as a straight we fixed this by implementing a check specific to this combination as it was the only outlier found that wasn’t caught by the initial algorithm.
* Game Player List is a data structure responsible for holding a list of players in the game at the time and provides methods for retrieving players meeting different criteria e.g. getting the player who is currently the dealer
* Hand is an enum type that denotes all possible hands that can be mode in poker to be used by the Hand Evaluator, and implements a comparator in order to rank the hands accordingly.

**Core game:** There are 3 main classes that comprise the main game logic, the Queue, Table and Game. Each of these run in their separate threads and have their specific functionality in handling the necessary game functions.

The queue is a thread that is started with the server and runs for the lifetime of the entire server, whenever a user wants to join a game they are first added into a thread safe priority queue and will be added to the first table with open space in the list (Fig. 13). The list of open tables is also maintained as a priority queue and whenever a table reaches its max capacity it is removed from the queue. If there are no tables available then the queue will start a new one provided there are enough users to add to one.

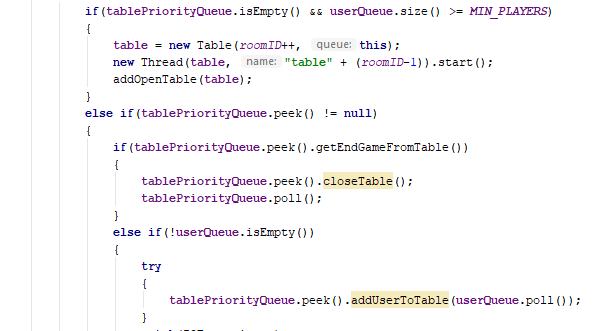


Fig. 13 Queue main loop for adding users to tables

The table has the responsibility of managing the players in the game, which includes keeping the game updated on which players are in the game, and handling the communication protocol with the players while they are in game.

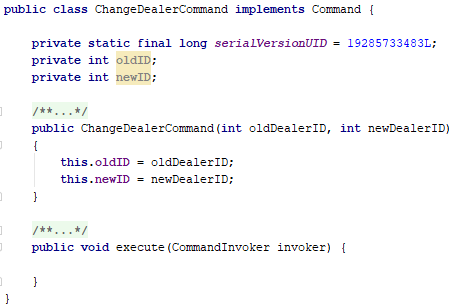
 Communication between the server and client is implemented using the Command design pattern (Fig. 14) which is created by implementing a specified Command interface which is serializable and giving it a method called execute which is implemented client side. The table creates the required command object with the given game data server side, sends the object either to a specific client or to all clients and waits for a response from the client if necessary. In this way the server doesn’t need to know how the commands are executed and the client doesn’t need to know the specific command sent in the object it simply calls the execute method common to all commands.

Fig. 14 Example Command object

The game has the responsibility of running individual hands for each table. This combines the uses of the game objects and the function of the table to communicate with the client in a main thread that follows a game flow in 6 stages:

* Pre-hand, where the game is setup by clearing the previous game state (if any), resetting the game variables, shuffling the deck of cards, changing the dealer and dealing each player their hand.
* Pre-flop, where the blinds are set and the first round of betting commences
* Flop, where the flop cards are sent to all clients and another round of betting occurs
* Turn, where the turn card is sent to all clients and a round of betting occurs
* River, where the river card is sent to all clients and a round of betting occurs
* End of hand, where the players remaining in the hand have their hands evaluated and each players statistics are updated appropriately

Within the game the table is notified to remove players under two different circumstances, the first is if the client notifies the server of a exit move, which can occur by the player pressing the leave game button in their game, or if any unexpected exceptions is caught that is not a SocketTimeoutException the table considers that as an exit move and will remove the player. The second circumstance is after a set amount of missed turns by a player, if the player’s ‘inactivity counter’ reaches the MAX\_TURNS\_INACTIVE they are removed from the table to make room for active players. When a player is removed from the table the database is updated with the player’s new statistics i.e. their new currency amount and other statistics saved in the details table discussed in section 6.3 .

The main issues encountered with implementation of the server were focused around concurrency which produced a number of bugs that were difficult to reproduce due to the number of threads that run at once when the server is up. This lead to an increased development time than planned as debugging took up a large portion of the time dedicated to creating other features which resulted in a change of design and having to remove the chat system and friends list functions from the planned implementation of the server. As the queue, game and table threads frequently interacted with each other and ran asynchronously extra care needed to be taken to ensure no thread was modifying data in another thread in a non thread safe manner. This was exemplified in a bug that occurred between the game thread and the table thread wherein the table would receive a player to add to the game but it would not be appropriately registered and the game would proceed as if the player was not in it, despite being added to the table.

* 1. Mobile Client

**Language and dependencies:** The client was developed in Android Studio and Java SE 11 in order to be consistent with the server’s development environment. The only dependency used client side is the google API package used to retrieving google accounts for signing in users and validating details.

**Client login:** The login activity is the first screen a user will encounter when launching the app on their mobile phone. The activity’s display will change based on whether the user has previously logged in or not, attempting to login the user if they have previously chosen a login method or presenting options for them to login with if they have not. The user has a choice to log in with either a Google account which will allow them to log in to that same account on any device with the app installed, or a guest account which is tied to the current installation of the game on that specific device. In either case the user is simply prompted to choose a username for their account if it is their first login and the account creation process is handled server-side with the account details saved to the database automatically. During the process of either creating a new account or retrieving the details of an already created account the user is kept informed of the current status by an updating text fragment notifying them of their current sign in status which is important to have a dynamic element when logging in so users understand that the app is working to log them in and isn’t just frozen.

Implementation of Google account login is made possible through the Google api which is used to retrieve an id token in the client which can then be sent to the server for back-end authentication and verification. For guest accounts every new installation of the app will automatically generate a random UUID, a unique identifier for that installation which is what will be used to identify an account when sent to the server to retrieve the user’s details.

**Connection service:** This is a background service which acts as the main hub for communication with the server done outside of the game and runs for the lifetime of the app. Whenever an activity needs to perform a network action it makes a call to the service through either binding directly to the service or sending a local broadcast which is received by the service. The service initially connects to the server when it is started and will spawn worker threads to communicate with the server when the appropriate calls are made. These calls correspond to the server’s handler thread functions as mentioned earlier, retrieving a user’s account details, joining the game queue, retrieving their profile and linking a Google account to their guest account. When these worker threads are called in order to avoid bugs that occur when attempting to make a call to the server after leaving a game the threads will automatically discard all objects that are not the correct response type expected as we had an issue involving the server still sending Commands after the client had requested to leave the game due to the server not processing the leave command until the user’s next turn, and potentially causing a ClassCastException when the service attempts to request data from the server’s handler thread.

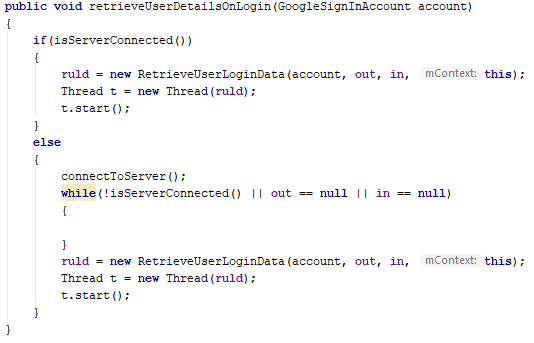


Fig. 15 Example service method to retrieve a user’s account details on login

**Main menu:** The main menu is fairly simple and corresponds closely to the initial prototyped design with 5 buttons arranged in a grid that acts as a hub for the other activities in the game (Fig. 16). Unfortunately since the friends system was not implemented on the server side the button on the menu does not have any functionality, and there are currently no options that are changeable in the options activity.

**Game objects:** Creating dynamic objects in the game activity of the app is handled with 2 main objects, the ClientCard and the ClientPlayer. These objects represent views of cards and views of each player’s status in the game. Initial implementation of these objects intended to make use of the canvas surface methods in Android studio by implementing a custom view surface and drawing the objects on that view. To achieve this we created a base GameObject class which our 2 game objects inherited from which would handle functions to retrieve and draw the bitmaps dynamically when they were needed onto the surface, which was supported by a worker thread to handle the object updates i.e. changing which hand cards the player held or updating the list of players. This design proved to be quite resource intensive however especially when attempting to load bitmaps onto the surface, and so the design was changed to hold simple bitmap image views on the activity, instead of a custom view. The game objects were kept as they provided useful methods for drawing bitmaps from a grid of arranged images.

The client side card object is drawn from a custom set of card images we made in order to make them as simple and visible as possible. This is changed from a traditional card design by showing the card rank and the suit in the centre of the card as large as possible, without any superfluous details so you can tell the card easily at a glance even with some sight impairment.

The client side player object is used as a baseline for drawing the user stats and turn animation onto. This is achieved by setting a base blank image as a background for the player and using a canvas object to draw text directly onto the image view it has been loaded on to, this draws the player’s name, their current currency amount, and their current amount bet in the ongoing round of betting. Depending on whose turn it is the player display frame will also have a red border that will disappear a section at a time in intervals of 5 seconds until the turn is over.



Fig. 16 Example card image

**Game model:** This class contains data structures which hold the data necessary to construct the game view on the client side. The class has 2 inner classes with the responsibilities of managing the game bets and the user’s own player details respectively, with the main class handling the other components of the game such as the current game state and the list of players connected to the game. This is where the game information that the server sends to updates clients gets stored and it is tied to the lifetime of the game view activity, meaning the data is not persisted any longer than the user is in the current game for which the data is provided.

**Game view:** This is the game activity that users will see when they join a game. This view visually represents the data held within the model and presents the user interactive options to make game moves when they are required. The design of the activity was chosen to maximise visibility of the key elements of the game and be as intuitive as possible. The activity is composed of fragments that are populated when necessary as well as layouts containing the constant on screen elements.

When it is not the player’s turn the user can see whose current turn it is, and will be notified by on screen text when they make a move but cannot make a move themselves. When it becomes the player’s turn the user is notified by a countdown timer starting on the left side of the screen and a visual animation will play as a form of visual cue to let them know it is their turn and how much time they have left to make a move. This is compounded by the appearance of buttons and a slider with which they can choose their move from as a result of showing the appropriate fragments. For users with smaller screens there is a function on tapping any of the cards displayed on the screen which will enlarge them on the screen to better be able to see what they are.



Fig. 17 Game view activity example

**Game controller:** The controller links both the model’s data and the activity view together, by implementing the observer pattern it receives updates when the model data is changed which will prompt changes in the user interface of the game view. For example when the client receives an update on the player’s hand it is updated in the model, and this change is observed by the controller which will call the appropriate method in the game view itself to display the new cards in the player’s hand.

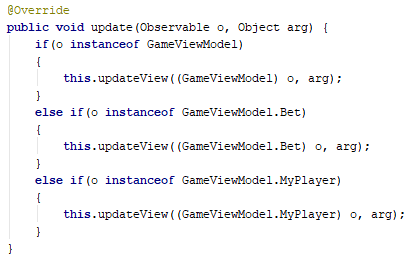


Fig. 18 Implementing observable interface within the game controller

**Game threads:** There are 2 key threads that support the game activity, the GameListener thread, and the CommandInvoker thread. There are a number of worker threads that function to update text on screen dynamically such as the turn countdown timer and the current pot value.

The GameListener thread is used in order to receive game commands from the server whilst the client is connected. Whilst this thread is running it will continually try to read in Command objects from the input stream and add them to a queue of commands which is synchronized to avoid concurrency issues between adding and removing commands to the queue.

The counterpart to the listener is the invoker, which takes commands out of the queue and calls their execute method, which highlights the strength of this design pattern, as the invoker does not need to know the specific command that was received by the client and added to the queue, it simply calls the same method on all objects queued and the appropriate data is modified.

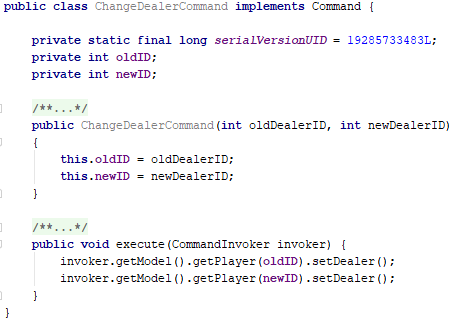


Fig. 19 Client side implementation of Command pattern with the execute method

**Game profile:** When switching to the profile activity the app uses the running service to request an updated user profile from the server, and updates the client side stored profile with the new data and these changes are reflected in the profile activity which shows a list of all the user statistics saved in the database. These details are stored in a shared preferences system for user details as being able to access and modify this data does not affect the integrity of the app as a fresh copy is retrieved from the server every log in and as such the lack of security is not a concern.

**User interface:** The interface throughout the apps activities uses the same splash background for consistency with a green theme reminiscent of a poker table. Changes to the design of the game view were made when implementing it as we realised the design of a table with players arranged around it was not making the best use of space in making the interface visible and easily learnable, as such this was changed and is reflected in fig. 17 above. Text on the green background is generally displayed in white with a slight shadow to make it as visible as possible. An additional element of the user interface is ensuring that all visual elements had appropriate content descriptions to correctly support the Android Talkback feature as mentioned in the design requirements, which includes dynamically changing the descriptions for elements that change during the lifetime of the activity, for example the cards displayed as the user’s hand needs to have their description updated whenever a new hand is dealt for user’s who use the feature to know what cards they have.

**Limitations:** Due to changes in the project schedule as a result of the back-end server development overrunning in time cuts to features in the client side app were made from the initial design. The biggest change in the implementation is the lack of an alternative app layout. In the initial design there were plans for a normal layout that all users would see on start-up and an optional ‘functional’ layout that focused on making key screen elements as obvious and clear as possible for all users, in the final implementation there is only one layout which is the ‘functional’ layout as there was not enough time to create the visual assets that would be required to support 2 separate layouts. Voice commands were initially intended to be usable to navigate the app and perform in game actions however when attempting this implementation we discovered that the Google provided speech recogniser was not accurate enough to be consistent for use in the app, we considered attempting to correct for this using a form of fuzzy string matching however this was not implemented due to time constraints imposed by other coursework leading to a lack of development time. With regards to the non functional requirements outlined earlier the design of the app was changed away from introductory tutorials to the app in favour of simply making the app more learnable and easy to pick up initially, the results of which will be discussed in section 7.

* 1. Database

The database structure consists of 2 tables; the ‘users’ table and the ‘details’ table. These store all the relevant account data required for users to play the game and have their stats persist between sessions. It also provides a layer of security to prevent exploits as it is all stored on a server and accessible by the main backend server, so that users are not able to edit their own details or spoof them in some manner to exploit the server.

The ‘users’ table holds the google user id, guest user id, username, last login date, and currency amounts for each user, these are essential fields for each user with at least one of google or guest user id needing to be provided for each user to be able to retrieve their credentials and details.

The ‘details’ table holds non essential fields that correspond to certain statistics measured as each user plays the game. These include the amount of hands played and win rate, among others and are not necessary to play the game hence are stored in a separate table.

1. **Evaluation**
   1. Testing

Unit testing was conducted server-side on the EvaluateHand class due to the complexity of the code being written there was a need to test cases of specific hands to ensure the correct evaluations were being made, as this would not prove practical to test in a live game environment as for example the statistical probability of a royal flush hand being dealt completely randomly is a tiny 0.00015% (1/649,740). As mentioned in section 6.1 this testing allowed us to find bugs with hands being incorrectly evaluated and ties being incorrectly broken in a trivial manner which helped to save time in what was a lengthy development process on the server side.

Further unit testing was planned for the server however there was not enough development time to create tests for all aspects and the focus was switched on performing live integration testing through evaluating expected use cases of the app during user testing in order to determine if there were any bugs or issues with the server side implementation.

Unit testing was not used on the client side as interface testing was conducted through user evaluation and interaction with the server was tested with a live build of the app instead of using a dummy server as described in section 4.2 .

* 1. User Testing

User testing was conducted throughout development of the client in order to receive quick feedback regarding the design and usability of the app. These tests occurred in a casual setting and would usually only last for a few minutes with the purpose of learning a user’s first thoughts upon using the app and where improvements could be made in order to make the overall experience more enjoyable and easy to use. These evaluations allowed for a more dynamic iteration over different user interface designs and interactions and to explore and detect bugs that otherwise might have been overlooked by simple static unit testing. Improvements to the design of the app occurred as a direct result of this testing, these include changing of text colour from an initial black to white as users reported the lack of contrast between the background making it hard to read the text, the inclusion of a text turn countdown timer for a more precise measurement of how long the user has left alongside the animation that plays around the player display. Smaller changes such as spacing and sizing were made throughout development in response to user feedback also. After a round of user testing during development we noticed a bug that caused the user’s profile statistics to not be updated when viewing the profile activity after leaving a game,

More structured user testing occurred at the end of main development of the client in much the same way as documented in section 5.3, with a test group engaging in cooperative evaluation and giving their thoughts during it, with a short questionnaire to answer immediately afterwards. The questionnaire’s were designed to present as neutral and unbiased questions as possible by using scales of measurement and avoiding using leading questions to receive the most accurate feedback possible from users, as well as providing a short section for extra written feedback to be documented. This proved useful in understanding common user actions and issues that might occur in practical use of the app. From this evaluation we learned that for the most part all levels of experience in users were able to interact with the app and found the navigation and gameplay to be intuitive and simple to understand. The most common criticism being regarding the styling of the buttons, of which the grey button colour was not popular and was planned to be changed to custom button images more fitting of the app’s overall theme however there was not enough development time to create the assets required and as such the buttons have not been changed.

The results of these questionnaires are included in Appendix B however there is no available transcript for the think aloud portion of the evaluation due to it not being recorded at the time according to participants wishes.

* 1. Requirements Evaluation

Functional:

* Platform: The app will run on Android devices running at least API level 15
* Login: Users are able to login with their Google account or a guest account, the option to log in with a Facebook account was dropped from the requirements not due to technical limitations but from a more moral perspective in regards to the recent news and information that came to light concerning Facebook’s practices with data and privacy.
* Accessibility: The app supports Talkback as much as possible, there is only one layout, the functional layout designed for visibility. The app does not support gesture controls as the control system using buttons was deemed intuitive and simple enough that gestures would in most use cases be more difficult or require more effort than simple button presses. Voice commands were not implemented due to limitations with the technology proving impractical.
* Connections: Through the use of 3 separate threaded classes players are able to join a queue to be added to tables in order to play a game of poker, and players can leave the game whenever they choose, and are alternatively removed from the game if they lose connection to the server.
* Algorithms: Shuffling of cards in a deck is implemented optimally within the server using Durstenfeld’s algorithm, poker hand evaluation is implemented using a non optimal brute force algorithm however the loss in speed should not cause any noticeable difference when used on a smaller scale, although a re-implementation will be necessary to be used at scale i.e. in the case that hundreds of thousands of hands need to be evaluated possibly simultaneously but this should not practically be a concern for use of the app.
* Data storage: The database server storing user data is only accessed through specific methods from the server after authenticating a user’s identity and all modification and updating of user game data is operated entirely on the server side to prevent fraudulent manipulation of stats by users.
* Game functions: The app provides within its game view a fully functional game of poker that can be played with up to 4 other players and have a common virtual currency they can bet and win in games implemented. Extra currency is given every new day logged in, and if the user has logged in for multiple consecutive days the amount of currency given is scaled up.
* Interface: The app currently has a login activity, main menu, game activity and a profile screen, features such as the options, friends and help menu are not currently in the app build due to development schedule limitations and the chat screen was not intended as a feature for the functional layout which is currently the only layout implemented and as such is also not included.

Non-functional:

* Usability: During our evaluations with users they found the app to be easy to understand and use, with the interface being clear and understandable with little need for explanation or learning, outside of the game interface player display which typically took users a round or two to understand what each line of text represented.
* Reliability: The server design means that user’s data is secured in a database with no identifying data stored that would affect privacy on the off chance the database is breached and data is leaked. As discussed in the next section testing the reliability of the server at scale was not possible however limited connections to the server using multiple clients was used and server performance was unchanged.
* Performance: On the tested hardware the performance of the app was smooth, with activities switching quickly and no noticeable or reported slowdowns of the app.
* Implementation: The app was created in Android Studio using Java as described earlier, the same as the server and so this requirement was met.
  1. Testing Limitations

A key weakness in the testing of our app as a whole is the inability to test at scale. User testing was conducted as part of a small group and independent testing only manages testing 1 to 3 active clients at a time, with no real way of simulating user interaction with the app and server at a larger scale. Ideally we would run stress tests in order to detect anomalous behaviour with the server under load to see how it would perform but unfortunately to be able to conduct such tests would require the creation of a separate testing client that could be run and reproduced in large amounts in an automated fashion, which was not realised until too late in the development cycle to be practical to implement. This limitation affects our ability to test the overall reliability and stability of the server, part of the non functional requirements outlined previously.

Additionally we did not have the required hardware to test our app on different configurations and as such were unable to design and test different layouts outside of the 1080x1920 phone resolution and performance. The hardware used to test the app was the Nox player emulator running on 4 cores and 3gb of RAM.

1. **Summary and Reflections**
   1. Project Management

In the initial plan for this project a list of tests were to be produced before beginning development however upon reflection we decided this was not feasible due to the sheer amount of tests that would need to be created and the agile development methodology meaning that requirements could very well change throughout development of the application, which we saw happen with the introduction of the ‘functional layout’ requirement 3.2. This project goal was adjusted to better fit with the principle of Test Driven Development, by creating tests as appropriate before beginning development on specific features of the project. However as noted previously we were not able to fully meet this goal over the course of the project as it became clear that the work required to meet the schedule was severely underestimated and could not be feasibly completed.

Further revision to the initial project plan was made to switch the schedule for when front and back end development would take place. This was made as we decided it would be naïve to leave back end development last as there would be little time to deal with potential bugs or issues that may occur. As such we revised the schedule to create only the basic, necessary components of the user interface such as the main menu and navigation first, followed by developing the back end functionality fully and then completing the front end functionality.

In following this project schedule throughout its course we discovered that after initially creating the prerequisite front end features, development on the back end would take a lot longer than expected. This was due to my own inexperience with server programming and lack of understanding of the specific intricacies of server-client communication and concurrency issues that occur as a result causing me to underestimate the amount of work required to complete the goals set out. This can be attributed to not being specific enough with my objectives for completion, and going forward I understand that setting a goal such as ‘create back end’ is much too vague and ambiguous, not conducive to a structured work flow. Going forward I will strive to break down my goals into more specific, achievable objectives in order to better understand the amount of work required to achieve my goals. Another key issue that arose with not being specific with my objectives was a lack of direction during development. Whilst this was not as much an issue with the client side as I laid out clearly in the requirements what was expected to be developed, with the server there was no clear structure to follow on what needed to be implemented and when by, this lead to a lot of time wasted attempting to figure out what had to be done to progress in development, and struggling to keep motivation towards the project.

As a result of this, even with starting implementation earlier than planned within the Christmas break, development of the back end took twice the time that had been scheduled within the project plan, leaving much less time to complete the features planned to be completed after the server.

Overall I believe that the initial goals I had set out to complete for this project were too optimistic given my level of experience with this type of software development, and this is reflected in the features that were dropped from the development goals as a result of progressing through implementation at a much slower pace than anticipated. This has in turn been a major learning experience for myself and I now believe that if I were to begin work on a similar project from scratch in the same timeframe I would be able to complete such goals using the knowledge and understanding I have gained in the creation of this project.

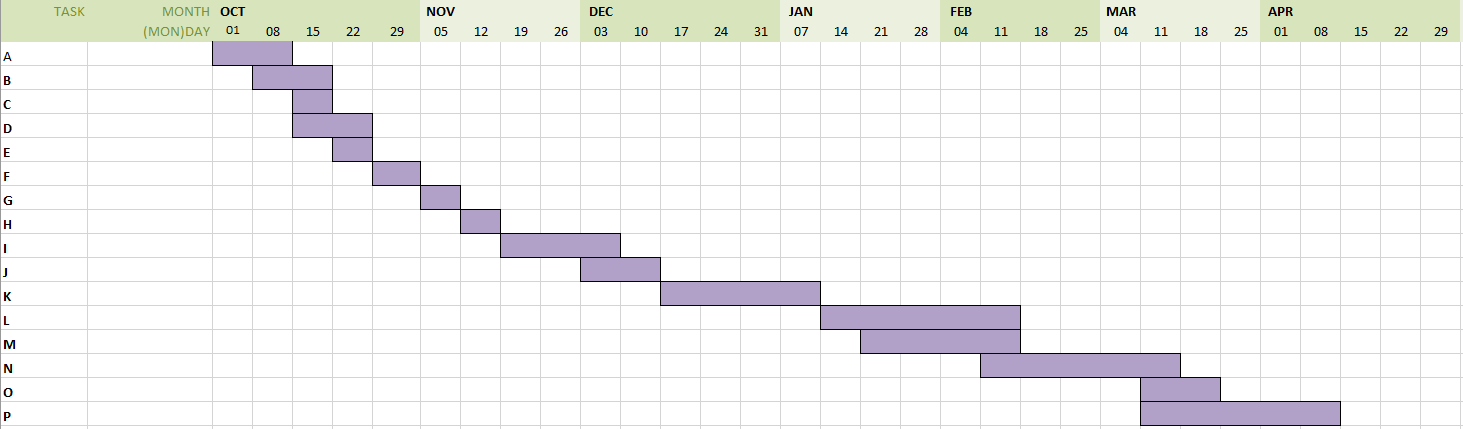


Fig. 20 Original Gantt Chart for project plan, see Appendix A for Task key

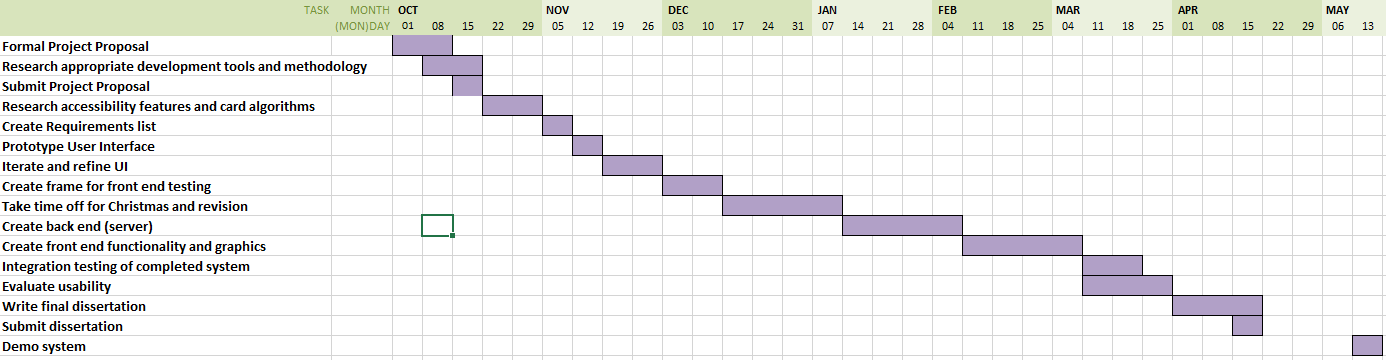


Fig. 21 Revised Gantt Chart for project plan

* 1. Project Appraisal

The challenges of designing and implementing a robust communication protocol between the client and the server have helped me understand and experience a new side of programming that, prior to this project, I did not have more than a rudimentary understanding of.

In particular the challenge of designing my own methods of communication between the client and server without relying on external libraries has helped me understand more underlying Java principles with regards to networking, for example understanding the concept of serialization and the optimisations Java makes to get the fast transfer speeds of objects, specifically the fact that Java will cache certain serializable objects in order to appear as if they a:re being sent quickly, which lead to a bug involving a lack of updating player statistics in a game as Java did not consider the serializable player object that was being sent from the server to the client as different enough so it would simply reload the cached version with the same statistics as before, instead of sending the updated version. Additionally this project has allowed me to gain experience applying different design patterns within my code and understanding their practical applications.

I believe that over the course of this project the learning experience has allowed me to better understand my own ability to self manage my time and plan a project efficiently. Through this I have recognised areas in which I have been lacking such as estimation of time taken to complete goals and approaches to solving concurrency problems in code.

**Summary:** Overall, I am pleased with my work on this project, despite the issues faced during development. Users appear to be satisfied with the system produced, although I would like to finish development on the incomplete features and create the complete system originally envisioned in a polished manner.

# 

# References

Anon., 2007. *World Series of Poker Retrospective: Horseshoe History.* [Online]   
Available at: https://gaming.unlv.edu/WSOP/history.html  
[Accessed 14 November 2018].

Anon., 2015. *AudioGames.net Forum.* [Online]   
Available at: http://forum.audiogames.net/topic/13872/android-accessible-games-please/page/2/  
[Accessed 10 November 2018].

Foster, R. F., 1937. *Foster's Complete Hoyle: An Encyclopedia of Games.* New York: Frederick A. Stokes.

Google, 2019. *Google API Java Client.* [Online]   
Available at: https://developers.google.com/api-client-library/java/  
[Accessed January 2019].

Joy, K. & Ricaurte, J., 2011. TapBeats: Accessible and mobile casual gaming. p. 2.

Liu, Y. & Li, H., 2011. Exploring the Impact of Use Context on Mobile Hedonic Services Adoption: An Empirical Study on Mobile Gaming in China. *Computers in Human Behavior,* 27(2), p. 9.

Morris, J. T., Sweatman, W. M. & Jones, M. L., 2016. Smartphone Use and Activities by People with Disabilities: User Survey 2016. *The Journal on Technology and People with Disabilities,* p. 18.

Neal, A., 2011. *The Evolution of Poker, From Origins to Texas Holdem,* s.l.: Ezine Articles, http://ezinearticles.com/?The-­Evolution-­of-­Poker,-­From-­Origins-­to-­Texas-­Holdem&id=6544601.

Newzoo, 2018. *Global Games Market Report,* s.l.: Newzoo.

Pew Research Center, 2018. *pewinternet.org.* [Online]   
Available at: http://www.pewinternet.org/fact-sheet/mobile/  
[Accessed 15 November 2018].

Sensor Tower, 2016. *Apple's App Store Will Hit 5 Million Apps By 2020, More Than Doubling Its Current Size.* [Online]   
Available at: https://sensortower.com/blog/app-store-growth-forecast-2020  
[Accessed 12 November 2018].