TouchHockey
The Multiplayer Android Air Hockey Game

An Undergraduate Dissertation Project
in the
Department of Computer Science
University of Nottingham
Nottingham, UK

by

Oyedipo Areoye
psyodar@nottingham.ac.uk

Supervised by Chris Greenhalgh
August, 2015
University of Nottingham

Acknowledgements

I am grateful to my supervisor, Chris Greenhalgh, for his continuous support throughout the duration of this project. For being very understanding of my cirmstances and providing the intial idea that made this project possible.

Abstract

This project is wholly a software engineering one and aims to focus on delivering a fully functional mini game to showcase the capabilities of Bluetooth technology present in the majority of modern android smartphones. I also aim to document key aspectes of the the process of the system to provide more understanding to the workings of the Bluetooth API to readers.

The major difficulty with this project comes with the minimal documentation of the Bluetooth API combined with having to properly implement communication channels. Also aiming to provide perfect functionality for the vast range of android devices will prove to be a challenge due to the fragmentation of the android ecosystem.

The project initially began by outlining a very brief overview of the problem. Which essentially was the lack of local multiplayer apps on the market, which I saw as a shame as this meant that although millions people around the globe own smartphones, many will unable to benefit from the joy that comes from multiplayer gameplay due to limited internet connection.

Thanks to Christoper Greenhalgh

Contents

1 Introduction

Smartphones in particular continue to be an integral part of our society. With smartphones becoming more affordable, studies predict that almost 30 percent of the world population will own a smartphone by the end of this year [1]. Due to these trends, mobile game development has also seen a sharp increase with studios traditionally developing software for consoles and PCs shifting their focus to providing mobile entertainment. Users are now spending more time than ever playing mobile games with over 50 percent more time being spent on average playing mobile games in 2014 than in 2012[2].

Interactive multiplayer games have undoubtedly shown extreme popularity across all platforms, however on mobile there is a clear difference in the dynamic. Whilst real time multiplayer gameplay has been widely adopted, this form of gameplay has had trouble penetrating the mobile market. The top ten best-selling console games of 2014 all heavily involve feature rich multi player gameplay[3].

A similar look at the mobile charts on both the Android and iOS[4]platforms indicates that single player games with mild multiplayer social features dominate the app market such as: turn-based games, simultaneous-move games and solo-multiplayer games. However when searching for successful mobile games that feature local multiplayer the results come out very thin. Reasoning for this may be due the needs that local multiplayer games demand such as connectivity and close proximity. For example turn-based games are tailored to the on-the-go lifestyle of the typical smartphone user [5] thus they can be played as a multiplayer but in different locations. Additionally real time online multiplayer games are exceedingly popular amongst the western world which may suggest why developers may lack the commitment to provide local multiplayer features. Moreover developing successful games to compensate for multiple screen sizes, performance capabilities and constantly evolving hardware is proving to be a difficult task[6].

However the lack of local multiplayer games may not be solely due to the lack of developer commitment or unpopularity of local multiplayer for the usual smartphone user; as the success of multiplayer gameplay in general is well documented. This may be due to the fact that more stationary lifestyles such as school children have become very familiar with this style of gameplay on smartphones which are now routinely in the hands of over 80 per cent of secondary school age children[7]. Thus local multiplayer deems perfect for a younger audience as they do not require the same needs as the on-the-go smartphone user. The key components of this paper focus on local multiplayer game play with Bluetooth technology, which deems to be an incremental need in terms of latency. It

will allow for fast data transfer rates when compared to transfer over cellular network [8] and has proven to have a higher amount of documentation and support for developers.

In extensive detail this paper will further discuss the development of Touch Hockey, a local multiplayer game on modern Android platforms.

1.1 Project Description

The intention of this project is to create a 2D real time Air Hockey style multiplayer game, which will be centered on local multiplayer functionality based on wireless technology. Local multiplayer in this case would mean that the players engaged in the gameplay would be in close proximity of one another. Therefore the game will not require any of the players to be connected to the Internet. The application will be developed solely for the android platform using the java programming language with the android SDK. Essentially the main function of the application should allow individuals to search for nearby players running the application and then initiate a request to start a game. The players should then be able to run the game seamlessly and keep track of their game history with all players.

1.2 Motivation

My main motivation originally stemmed from a trip to Nigeria in the summer where I stayed with a large group of my family. During my stay I was thoroughly encouraged to partake in various different entertainment in the form of games. This heavily involved board games, physical sports such as football and table tennis, social games such as charades and then the rare console session on the PlayStation. Due to the unstable electrical supply and a strong reliance on limited generators; the difference on the reliance of electrical components for entertainment with the Nigerian natives was vastly different from what I was used to in the United Kingdom. Of course perfectly understandable, and it was rather refreshing to be able to partake in outdoor activities involving the entire family. Yet on occasion, there was the still the use of smartphones to play mobile applications where I noticed the effects of limited internet in terms of how my Nigerian peers and I could use our smartphones for entertainment purposes alike in the UK.

- Lack of connection to any online leaderboards (Apple Game Centre and Google play Services)
- Lack of integration with social accounts (Facebook / Twitter Login)
- Overall lack of updated apps / apps released within 6 months. ...

This means that although they have the mobile games installed, they heavily miss out on the engaging social features. Competition (which was central to all the games I partook in) thoroughly increases the longevity of mobile apps and due to the limited ability to download and update new applications increasing applications longevity with offline features is extremely important. Therefore the clear need for more competitive multiplayer games with offline support was my main motivation.

1.3 Aims

The aim of this project encompasses three main tasks. Firstly the technical side of setting up the net- working infrastructure allowing multiple players to engage in real time gameplay independent of an internet connection and gameplay on one device. Secondly the task of designingthe 2D Air Hockey game, which can live up to the current standards on the Google play store providing its users with enjoyable gameplay. Lastly this paper will analyse: the scalability of the networking infrastructure; the testing carried out with the use of questionnaires and; the success of the project identifying potential improvements for future development.

2 Related Work

This section aims to provide a brief background on the hockey style multiplayer games. Also an analysis of the major hockey style games currently featured on the Google play store. All results were found by searching hockey in the search option on the store.

Air Hockey style games have been around for over a decade and take a direct influence from Pong - one of the oldest video games known to man. With over 1.5 million apps on the apps on the Google Play store and around 2 billion downloads per month[10]. It would be highly worrying if this arcade classic did not have a representation on one of the worlds largest applications stores. As expected this is far from the case as air hockey style games are in full force.

Interestingly even with the significant popularity of the Air Hockey game within the arcade genre, there does not seem to be much variety from app to app. Most of the popular applications seem to take an extremely similar approach only varying on its aesthetics in terms of art design. This is highly expected as due to the nature of arcade games they are relatively easier to clone and do not offer as much depth in terms of gameplay mechanics, alike other genres such as adventure or strategy. However one way in which these games can offer differentiation is in terms of extended functionality such as multiplayer. As expected all the popular air hockey games do offer some type of multiplayer. The following section will briefly evaluate the most popular applications analyzing both positive and negative qualities of each game. These evaluations can also be used to scope the design in my project.

Glow Hockey The first application to be explored is considerably the most successful. The initial Glow Hockey instalment managed to achieve 100 million downloads to date with the second instalment bringing in a tenth of that number with minimal changes as a separate release. Although being on the app store for just under 5 years it has managed to stand the test of time and is still the most frequently downloaded air hockey game to date.

The app presents the expected features for a game of this nature. It focuses on rich visuals and animations with its neon glow theme. The game follows the standard gameplay mechanics, offers the player a single and 2 player mode. The single player mode places the player against an AI controlled opponent in which the player has a choice between 5 different difficulty levels. The local multiplayer takes place on the same device using the smartphones multi touch technology allowing two users to control separate mallets ,

Finally the game offers the user the option of customisability allowing the user to change the appearance of the game board, paddles and puck.

However whilst the app offers spectacular aesthetic features, alongside minimal customisation and smooth physics. Its major drawback is definitely down to the lack of replay value. Although the single player mode offers varying AI difficulty, the AI can still be easily defeated using repetitive tactics.

Air Hockey Deluxe The second game is highly similar visually and identical in gameplay mechanics to Glow Hockey. In offers similar visuals, physics and identical single and multiplayer modes. Therefore it gains the same positives from Glow Hockey and the same drawbacks. The one additional feature presented in this app however is the ability to set timer for a round resulting in faster paced gameplay which adds does add variety but still does not have any impact on the game mechanics nor address the issue of lack of replay value.

Air Hockey Speed This game once again follows the highly successful formula of its predecessors, and does not differentiate itself with any outstanding features. Like the others it suffers from similar drawbacks with minimal replay value.

Evaluation The sheer popularity of these games can provide validation for the potential success for an air hockey game. Analysing that these games do little to differentiate from each over and still receive positive review and high download numbers indicates that there is a clear market for the genre. However in regards to the motivation for this project, there is lot more room for improvement in the experience that can be provided to players especially in the multiplayer category. Although all three games make an effort to provide multiplayer gameplay on the same device this solution is not always optimal due to limited screen sizes and the difficulty involved in balancing a phone whilst manoeuvring the mallet for two players on the same screen. For this project to be a success it will be important to take the positives from these examples such as smooth physics and visual appeal, whilst attempting to make an improvement on the drawbacks in regards to adding replay vale through some sort of leader boards where players can tract the progress alongside a more robust multiplayer system as naturally Air Hockey was designed and is best played in a multiplayer environment.

3 Platforms and Tools

This chapter aims to discuss the possible tools that could have been used to develop the project alongside a look at potential platforms for the application whilst evaluating the positives and negatives for each. The project itself is separated into two sections; the game engine itself and the networking framework on top of a wireless technology. To reach the project goal, the decision of the mobile platform was to develop the application for had to be made. Various wireless technologies where considered to handle the connection between the devices alongside the possible use networking helper libraries. A variety of game engines were also considered to handle the rendering and physics of the actual game. Finally some research into how local proximity can alter the dynamic of gameplay was done with a specific look into games that make use of dual or multi screens. The final decisions made on which tools will be used for this project are detailed in Chapter 6.

3.1 UI Design and Prototyping tools

3.2 Wireless Technologies

3.2.1 Bluetooth

The Bluetooth wireless technology is a standard for transferring data over short distances. It is used in data transfer for both stationary and mobile devices which it first made an appreance in 2000 with the Sony Ericsson T36. The Bluetooth technology has matured into a smart simple and secure hardware enabling it to be found in million devices across the globe and in all smartphones today. Due to the technologies prevalence and adoption from developers it has become the standard for mobile communication in close proximity. Data transfer on Bluetooth technology can be transmitted at around 100 meters[1] which will support local multiplayer games and another benefit is due to Bluetooth low power consumption on devices making it extremely suitable for transferring data in real time. Improvements to the Bluetooth technology are ongoing and the Bluetooth Special Interest Group (SIG) introduced Bluetooth Low Energy (BLE) in 2010, which is predicted to be provided in 90 percent of smartphones by 2019[14]. One major benefit of BLE is it can display up to 100 times less power consumption that classic Bluetooth technology[9].The

[1] `http://www.bluetooth.com/Pages/Basics.aspx`

Bluetooth technology is heavily supported with both the iOS[2] and Android[3] operating systems with well officially documented APIs to communicate between devices using the technology.

How Bluetooth Works The Bluetooth protocol transmits data at a frequency of 2.4GHz[4] like other communication protocols[15]. What differentiates it is a standard set of rules and specification which is defined and trademarked by the Bluetooth SIG.

Bluetooth devices operate in a network commonly refereed to as a piconet[5], which can involve a maximum of 8 devices. In this network one device is assigned to be the master whilst all the other devices are slaves. In a piconet a slave can only directly connect to a master whilst a master can directly connect to up to 7 slaves.

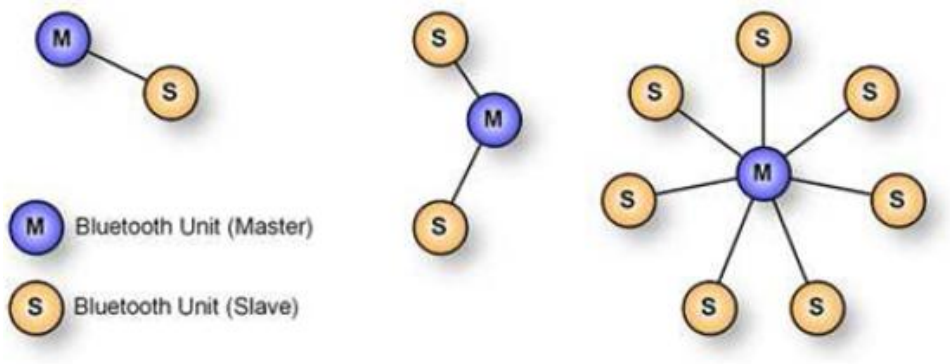


Figure 3.1: Slave/Master model in a piconet network

This setup ensures that data transfer can only happen between a slave and a master, this means that in the case that a slave device wants to send data to another slave device within the same piconet then it must transmit this data through the master device. The Bluetooth technology is not only just a hardware radio based system but also involves software stacks that provides links between the various layers which can be likened to the OSI model. This means that the Bluetooth technology protocol stack defines well-structured hardware and software layers providing robust guidelines for optimal interoperability and compatibility between devices.

[2] https://developer.apple.com/bluetooth/
[3] http://developer.android.com/guide/topics/connectivity/bluetooth.html
[4] http://www.bluetooth.com/Pages/Basics.aspx
[5] http://ntrg.cs.tcd.ie/undergrad/4ba2.01/group3/terminology.html

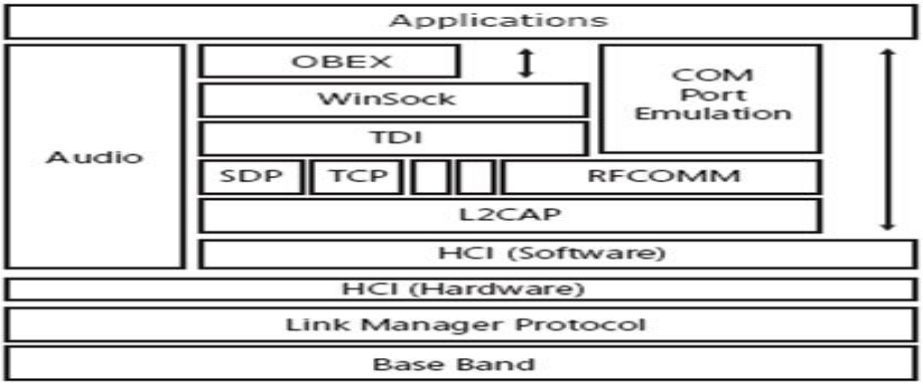


Figure 3.2: Bluetooth network stack

3.2.2 Wi-Fi Direct

Wi-Fi Direct is a standard allowing Wi-Fi devices to connect to each other wirelessly. Although based upon the same technology as the standard Wi-Fi radio present in all smartphones but it is only supported in a certain number of devices (Android 4.0 and up). Nether less Wi-Fi Direct does offer communication over a longer range than Bluetooth with a supported range of 200 meters[6]. Although the power consumption using Wi-Fi like all communication technologies can be attributed to many different factors. When compared to Bluetooth, Wi-Fi technology does not offer the benefits in low energy consumption , seen with BLE as Wi-Fi chips consume 3 percent more power to transfer the same amount of data[11]. Google provides an official API[7] for the technology allowing android devices to discover request and connect to nearby Wi-Fi Direct enabled devices. However there is minimal documentation on establishing communication channels between devices to transfer data in real time. Wi-Fi Direct is not supported by the iOS pllatform and has no such API.

3.2.3 Data formats For Communication

One challenge in wireless communication with mobile phones is that data must to be sent in byte format. Language objects therefore must be serialized before sent to another device, which means that the bytes must then be parsed and built into desired class objects on the receiving end. With this in mind a decision needs to be made on how to setup up the networking infrastructure and the potential methods of storing and sending the data had to be analysed.

[6] http://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel
[7] http://developer.android.com/guide/topics/connectivity/wifip2p.html

String Sending messages in the simple string format is highly viable for short messages with pre-determined content. The String object will have to follow a consistent format as to enable seamless parsing from the receiving device. One advantage of this method is the simple setup most standard libraries offer simple conversion from String to byte arrays making string easy to send over a wireless connection.

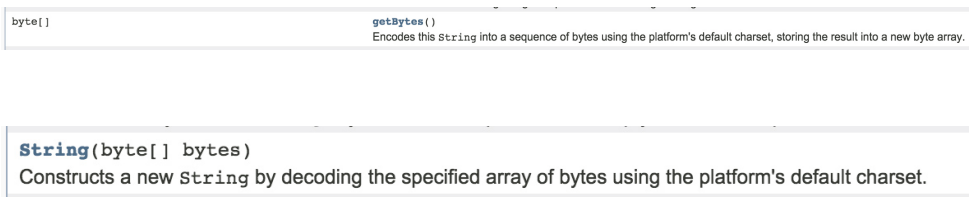


Figure 3.3: String construction and deconstruction

Sending messages in the simple string format is highly viable for short messages with pre-determined content. The String object will have to follow a consistent format as to enable seamless parsing from the receiving device. One advantage of this method is the simple implementation and, most standard libraries offer simple conversion from String to byte arrays making string easy to send over a wireless connection. Despite the initial simplicity, parsing Strings can prove to be difficult and is by no means a scalable solution. When parsing large amounts of data parsing strings can begin to require a lot work to ensure type correctness. For the networking portion of this project to remain easily extendable Strings will possess a great challenge. One issue is that before serialisation strings must be given a stop marker to signal the end of the message. Storing complex information in string will also heavily reduce the human readability due to the lack of a defined structure.

XML The Extensible Mark-up Language known as XML was created as a language to define a set of rules to encode documents in a both human and machine-readable manner. According to the W3C its also plays a huge role in the exchange of data on the Web and elsewhere. XML is a widely supported format across a majority of platforms and complete functionality to create parse and serialise XML data is provided through the JAXB library included in the Java SDK. Although XML is powerful tool in document mark-up it was not purposefully designed to define the structure of class objects.

JSON JSON (JavaScript Object Notation) is a lightweight data-interchange format, which offers great human readability. JSON was built with data exchange in mind and is supported by all modern programming languages. Due to its properties, objects in JSON

are able to follow a clear defined structure making parsing JSON data and converting it to application objects extremely simplified.

Google Protocol Buffer Although both XML and JSON both offer well documented methods to structure data objects for the transfer across applications, one of the major factors for deciding which technology to adopt for this project is how efficiently data can be serialised into its transportable byte format and thus equally deserialized on the receiving end. This is where the Google protocol buffer shines. A protocol buffer is a defined method of serializing structured data, designed to be smaller than XML with equal simplicity. This protocol buffer was built with serialisation in mind and offers the greed human readability and simplicity.

3.3 Game Engine

A game engine is a software framework essentially created for the development of video games. Its core functionality aims to handle the rendering of the visuals but it also handles the task of calculating game physics, user input and animations allowing developers to create fully functional video games. Game engines, although platform dependent usually follow a standard set of concepts and practices when dealing with the production of a video game. Game engines can usually come with cross platform functionality and can be differen- tiated by whether they involved 2D or 3D graphics rendering. Game engines come with unprecedented benefits as they allow the developer to work on game content and design. Most of the complex tasks are memory management, asset management and lighting which greatly simplified.

Unity Unity is a cross platform engine developed and released in 2005, it is one of the most popular game engines to date, as it is the most common choice for developers creating games for the mobile platform[10].

One of Unitys major advantages as previously mentioned is due to its popularity, it has been around for almost a decade and thus has a lot of experienced supporters and therefore its usage is well documented. Its cross platform feature allows games to be ported seamlessly. The Unity SDK comes with its own IDE, with a core focus on video game creation therefore providing developers with professional tools specific to video game development, which also makes it easier for beginners. One of Unitys drawbacks however is more specific to this project; games built in Unity cannot easily be integrated with any software built outside the platform. This means that the networking architecture

must also be built inside the unity framework and will thus have no compatibility with non-Unity games.

Cocos2d-x Cocos2d-x is another open source framework supporting 2d game development. It is also highly popular and a search for cocos2d in stack overflow will return 15,000 results. It is written in C++ and supports game development for all major mobile platforms. It is an extensively documented engine and games based on this engine also dominate the mobile app stores. Like unity Cocos plays a heavy focus on providing cross platform tools to enable games to be published anywhere. This however means that games have to be written in C++ , Lua or Javascript. One issue is this is that the major IDE for android development does not natively support C++ and thus to develop games using Cocos the user must either rely on an unsupported port or develop Cocos and native elements of the application to be compiled separately which leads to difficulty in integration. To build android games users must have the java NDK (Native Development Kit) installed, which is not currently supported by Android Studio.

AndEngine Andengine is a 2d game engine developed by Nicolas Gramlich. Its main advantage is that it was developed solely for the android platform and thus takes advantage of must of its native features. It features a robust API for the handling game mechanics and asset rendering. Its also extremely lightweight when compared to Unity and Cocos and can easily be integrated with any existing android projects. One major drawback of is that Andengine does not have any official documentation but it does however have numerous examples on the web detailing the various aspects of the framework. A search for Andengine in stack overflow returns 1,400 answers since January 2014, which is fairly considerable as its an engine designed for a solely for the android platform.

Box2d Box2D is an open source physics engine used to calculate game physics involving colllision detection and determine game properties such as gravity and friction. It was created in 2007 by Erin Catto, It is the most used physics engine by game developers and has been used in billion pound games such as the famous angry birds. Many modern game engines including AndEngine and cocos2d feature box2d wrappers to offer physics rendering functionality in their engines.

4 System Specification

4.1 Functional Requirements

This section will enumerate the functional requirements of the project; these requirements will thus be used to evaluate the project.

1. Users can host a game.
    (a) On attempt to host a game the user should be notified on his/hers Bluetooth state.
        i. If Bluetooth is disabled the user can enable for a pre defined period.
    (b) The host should be notified when another user (client) joins the game.
        i. The host should be notified of the clients Bluetooth name.
        ii. The host can accept or decline the connection.
    (c) Users can search for hosts to join games on doing so;
        i. The users should be presented with a list of current hosts within their Bluetooth range.
        ii. Upon selecting a host to join the user should send a request to pair with device if unpaired.
        iii. Once the connection is complete both users should be taken to the main game screen.
    (d) On the launch of the main game screen the users should be presented with the game assets.
        i. The user should see his/her Mallet in its starting position
        ii. The puck should spawn at the centre of the hosts screen.
        iii. A countdown should initiate indicating the game proceeding
    (e) During the gameplay:
        i. The user should be able to pause the game.
            A. The game should automatically pause for both players if a user exits the application.
        ii. The user should be able to control his/her mallet hosts screen.

iii. Upon the user bringing the mallet into contact with the puck, the puck should move with appropriate velocity.

iv. Upon the puck crossing the top boundary and thus exiting a players screen it should then appear with the same motion and relative position on the opponents screen.

(f) Players should be awarded a point if the puck crosses the opponents goal.

i. Upon scoring a goal the pucks position should then be reset on the conceders screen.

ii. Both players should be notified of any changes to the score.

(g) A victory should be declared when a player attains a score of 7 points.

i. Upon a player reaching a victory both players should be notified and the game over state should ensue on both screens.

A. On this screen the host can choose to restart or quit the game.

B. Upon game restart the game should return to requirement 4.

C. Upon quitting both users should be returned to the home screen.

(h) A user should be able to view an overall summary of their game history.

i. This should include the amount wins, losses and the win/loss ratio.

4.2 Non-Functional Requirements

This section aims to detail the non-functional requirements of the application.

Accessibility As previously mentioned the due to the sheer fragmentation of the android ecosystem. One has to ensure that any software being developed for this platform is highly accessible and works as expected on the multiple devices available for the user. The fragmentation results in various screen sizes, but also varying Operating System versions (API levels) and device specifications, which can result in varying performances of software on each device. Thus the system must ensure be accessible for all the devices it is created and available for.

In regards to the Bluetooth framework it must also be accessible for other developers to easily view implement and extend. Thus the framework must be available to download and must work with the various major IDEs (Android Studio, Eclipse) used to develop android applications.

Maintainability One major goal for this project is that the underlying network architecture can be portable and modularized enough to work with any game or front end. This would greatly increase the usability of the project giving the possibility for future developers to save substantial time on the development of local networking based applications. To achieve this the code must have a high readability with necessary comments and follow general conventions in regards to the language its written in. The overall networking architecture should be well separated to allow easy customizability for the user and simplify the debugging and testing process.

The game itself should be easily maintainable for the purpose if any future updates need to be made.

Performance The game should showcase an appropriate response time, and frame rate (minimum of 20 frames per second) under normal circumstances (within appropriate Bluetooth range).

Recoverability To ensure the system remains entirely recoverable the project will be backed up using a GitHub repository. The game itself upon release will be available on the relevant app store ensuring users can re download it anytime if connected to the Internet. User data will be cached on its local devices allowing recovery if the application is somehow lost.

5 System Design

Creating a high quality mobile game is a challenging problem due to the fact that it is really hard to judge the reaction of its end users as the when project is developed solely. It would be difficult for me to judge the success of the game without extensive user testing or input.

5.1 Rapid Prototyping with User Centered Design

The design process that was used for this project was based on rapid prototyping with a focus on user-centered design. Rapid prototyping simply involves continuously iterating a three-step process, which consists of developing a prototype, reviewing it and then refining the prototype based on feedback. Below is an overview of each process:

Prototype From the requirements devise a series of mock ups aiming to portray functionality which focusing on user experience and following.

Review With user feedback evaluate the prototype assessing its success in meeting user needs and expectations.

Refine Based upon the user feedback, identify key areas of improvement for the prototype.

Rapid Prototyping is a highly beneficial process as it allows users a tangible demonstration of the system. It also forces the development to be based around the user needs, which is why it fits the User Centric design process. Rapid prototyping promotes swift development and the final design is constantly being improved if reduces the need for major changes at later stages in the project which is heavily favorable considering the time constraints placed upon the delivery of the software.

User Centered Design User Centered Design is a method standardized by the International Organization for Standardization (ISO). It is a project approach that essentially places the intended user of the software at the centre of both its design and development. This is achieved by continuous communication between both the user and developer. The standard identified as ISO 9241-210 describes user-centered design with 6 key principles:

- The design is based upon an explicit understanding of users, tasks and environments.
- Users are involved throughout design and development.
- The design is driven and refined by user-centred evaluation.
- The process is iterative.
- The design addresses the whole user experience.
- The design team includes multidisciplinary skills and perspectives. ...

For the project to be a successful one the final game produced must be wholly playable and must be able to match the standards of those on the respective app store. This can only be determined from the end user of the game. The benefit of designing in relation to the experience delivered to the user is thus very beneficial.

5.1.1 First Prototype

The first step of the design process was to determine the screen flow of the game. This will then aid the process of designing low fidelity wireframes to visually present the core functionalities of the game. Rather that focusing on the game mechanics or aesthetics. The purpose of the wireframes is to detail the functionality of each screen and this will help to visualize how the user will move between each of the game states. The screen flow of the game can be designed purely of the functional requirements. Although the design of the visual appearance of the game is highly likely to change during the design change the general screen flow detailing the games core functionality will remain unchanged.

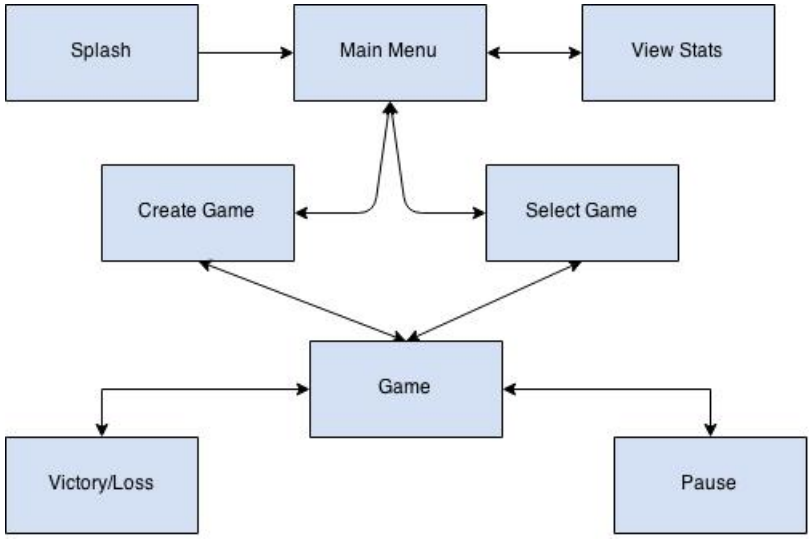


Figure 5.1: Expected flow of the Game

Although they seem simple, the screen flow aids the process of ensuring that the game flow makes sense and also helps to analyze how all the various game states connect to to each other. Note however that this is a simplified flow as within the actual game uses will be able to go back to the main menu from the pause screen etc. The purpose of the screen flow is to visualize the core path of the gameplay from start to finish. With the screen flowcombined with the requirements specification the functionality of each of the game screens can be determined. The following wireframes where created with Balsamiq.

5.1.2 Splash and Menu Screens

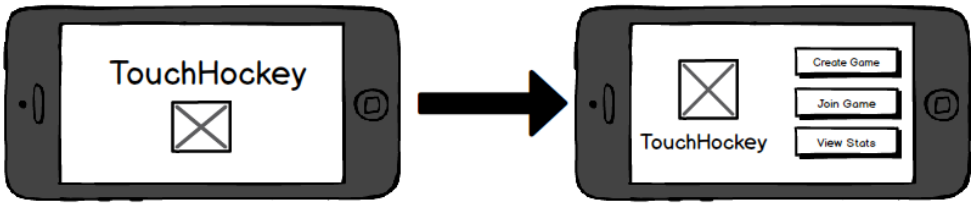


Figure 5.2: Left : Splash Screen , Right: Menu Screen

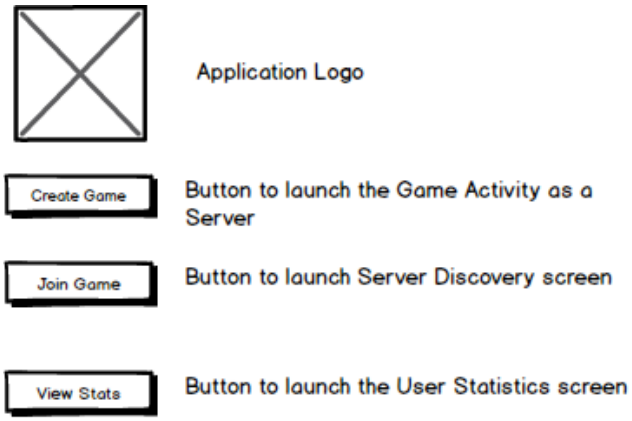


Figure 5.3: Details of icons and buttons

5.1.3 Statistics Screen

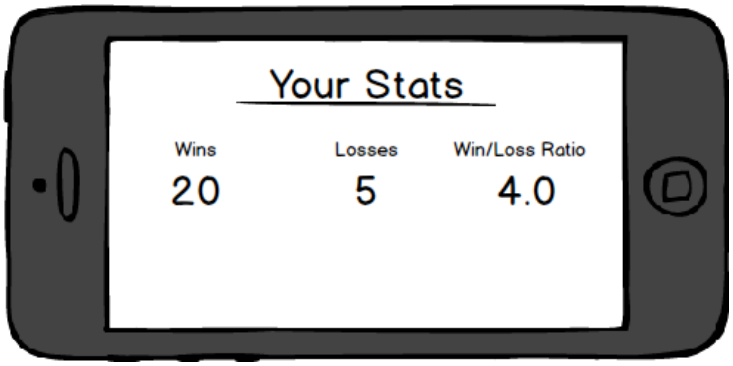


Figure 5.4: User Statistics Screen

5.1.4 Discover Game Screen

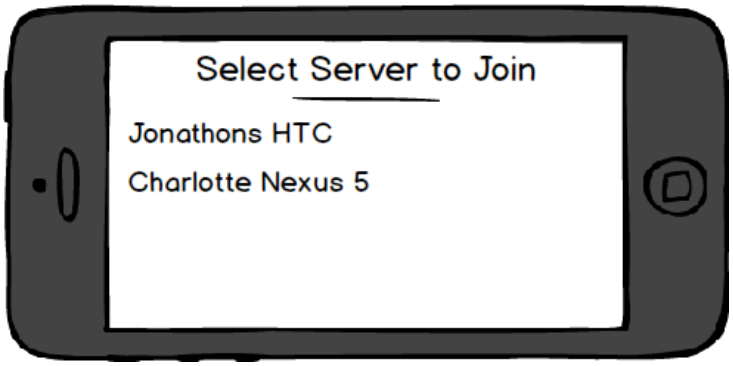


Figure 5.5: The screen to discover nearby servers. Servers are identified by their bluetooth name

5.1.5 Game Activity Screens

Figure 5.6: The Game Activty from servers point of view. Conection has not been made.

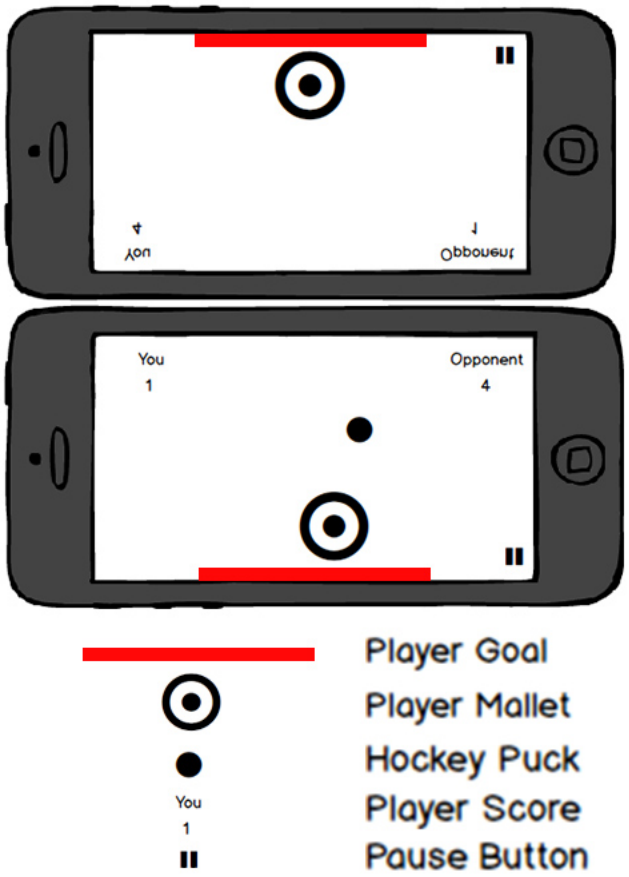


Figure 5.7: The Game Activty showing both the server and the client side by side.

5.1.6 Review Method

After the initial prototype stage in which the mockups where created, the review process was able to proceed. The review process generally focused on usability testing methods in which the user was presented with the mockups and a test was conducted to analyze the ease at which the game interface could be traversed.

The purpose of the review was to answer the following questions.

Can the user effectively create a game, and join existing game?

The Usability test consisted of a tester being asked to traverse through the set of mockups placed on the phone. The in app button images where set to be hyper links which then resulted in an action in the app, such as pressing the create game button would transition the page to the desired screen. This method was very effective in getting realistic feedback on the flow of the application without actually any implementation. The feedback of the study was very positive with users having a clear understanding of the applications user interface. All 6 users could effectively distinguish between creating

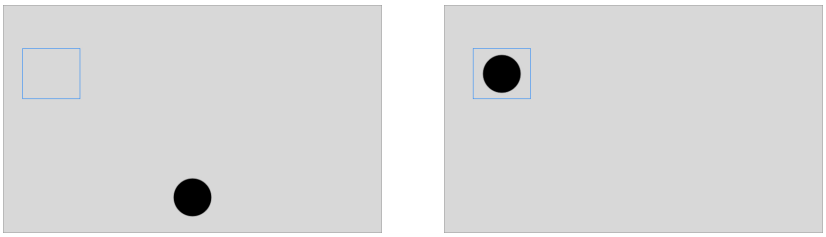
and joining game, however one user suggested a slight change to be made to the button names such as host game instead of join. A User also suggested that the back button be removed, as it possesses the same functionality as the android device back button. Moving forward based on the presented screen flow a second prototype was created with actual interactivity to observe user behavior and determine user controls.

5.1.7 User Controls

Modern day smartphones are presented with limitations in regards to game controls. With the touchscreen being the entity that provides the user with information whilst also detecting user input, designing interactive games often involves carefully managing screen space. With the TouchHockey project there were various methods of receiving input form the user in regards to the movement of the Mallet:

Touch , Drag and Drop The Drag and Drop method is the most popular method in the genre and is the gesture most realistic to the real Air Hockey game. With this input method the user will control the mallet by touching and holding within its boundaries whilst effectively dragging it across the screen in any desired direction. The mallet essentially follows the exact path as the holding finger.

Tilt The accelerometer within the smartphone can be used to retrieve orientation feedback about the device. This allows the player to potentially tilt the device to move the mallet. This method also offers 360 control to accelerate and decelerate the onscreen mallet but will require extra motion from the user and is not as intuitive as the drag and drop method. This method of input has not previously been applied in any air hockey games but has been featured in various other applications such as DUAL! Mentioned above. A major benefit with this method is that it offers the maximum screen space, as the user does not have to touch the screen during gameplay.



(a) Start Position (b) Clearing The Square

Figure 5.8: The second prototype

5.1.8 Second Prototype : Screen Controls

In the second iteration, an interactive prototype was designed to figure out the best way to implement the user control system of the game. This prototype was a simple implementation, requiring the user to control an on screen circle representing the mallet and attempt to move it using the control methods outlined above. Each user is give both control methods to try and then their score (equal to the number of squares cleared) was recorded for the purpose of the study.

In this task the user was given 1 minute to see how many squares they could clear. The mini game would start with the users Circle and the target square positioned as seen above. Once the user moves the circle the timer starts (hidden to the user) and they must move the circle into the desired boundary of the square. Once a square is cleared it disappears and simultaneously another square appears in a random location on the screen.

The study was carried out with 6 participants of various degrees of mobile game experience and the results are as published.

From these results a clear assumption can be made that the first control scheme leads to better results and as this game tested reflex, response times and general accuracy it can be assumed that the first control scheme grants the users higher performance in regards to those skills. Quick response times and reflexes are key skills in air hockey so the first control scheme (touch and drag) would be the better-suited method to provide users with an improved control experience.

However upon observing users conducting the touch and drag study it was noted that users had difficulty with controlling the spite on certain screen regions. It was noted that users began using two thumbs to move the mallet from side of the screen to another. This posed a serious limitation on the control scheme as it switching thumb to control the mallet posed a significant delay, which will cause discomfort in the real time game. Unfortunately the user playing with a single thumb cannot solve this issue, as it will lead to the user experiencing unreachable regions on the opposite side of the screen.

(a) Left Handed Player (b) Right Handed Player

Figure 5.9: Thumb Heat Map

With a game of this nature there are not many ways to overcome the limitation of major unreachable regions. One quick solution would be to redesign the game to be portrait only. As the design process was still in its early stages, this did not prove difficult and second study could be repeated with the same prototype. The second study essentially confirmed the greater accuracy of the touch and drag control method and that also proposed the problem with limitations of movement across the screen with a sing thumb.

The study was then repeated in the portrait orientation under the same conditions as the landscape orientation study. As seen below it showed a slight improvement on the results (might be due to the groups familiarity with the game). This time the study was only conducted with the touch and drag control method as it has been proven superior.

The unreachable region problem however was not entirely solved, the thumb map below details the screen limitations when the phone is held in portrait mode, however with regards to the air hockey game this is minimalized where the player will be able to defend their goal as the unreachable regions are nearer to the top of the screen (a problem will exist in a real air hockey game) the player ultimately will have to stretch to move closer to the halfway line.

With a game of this nature there are not many ways to overcome the limitation of major unreachable regions. One quick solution would be to redesign the game to be portrait only. As the design process was still in its early stages, this did not prove difficult and second study could be repeated with the same prototype. The second study essentially confirmed the greater accuracy of the touch and drag control method and that also proposed the problem with limitations of movement across the screen with a sing thumb.

The study was then repeated in the portrait orientation under the same conditions as the landscape orientation study. As seen below it showed a slight improvement on the results (might be due to the groups familiarity with the game). This time the study was only conducted with the touch and drag control method as it has been proven superior.

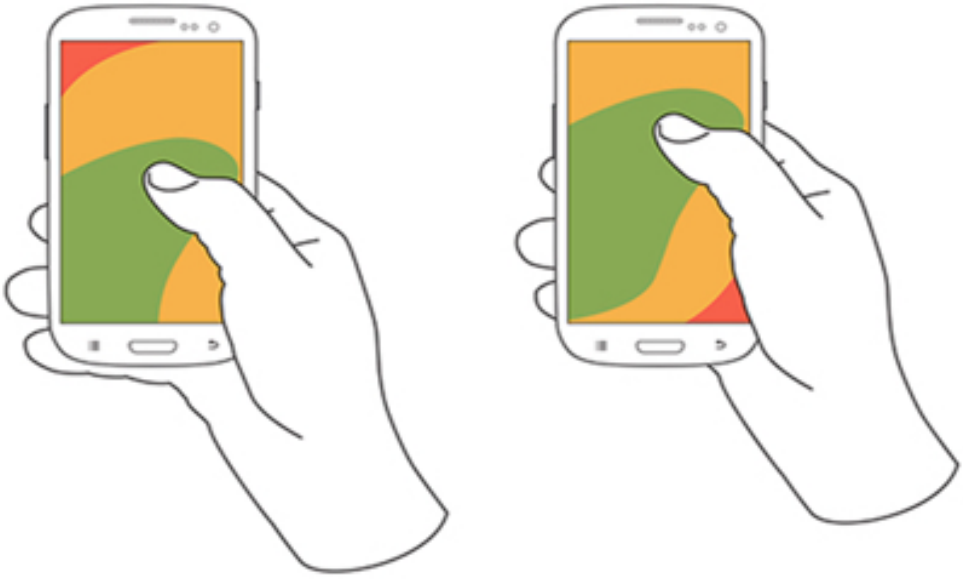


Figure 5.10: Portrait Thumb Heat Zones

The unreachable region problem however was not entirely solved, the thumb map below details the screen limitations when the phone is held in portrait mode, however with regards to the air hockey game this is minimalized where the player will be able to defend their goal as the unreachable regions are nearer to the top of the screen (a problem will exist in a real air hockey game) the player ultimately will have to stretch to move closer to the halfway line.

5.1.9 Final High Fidelity Mockups

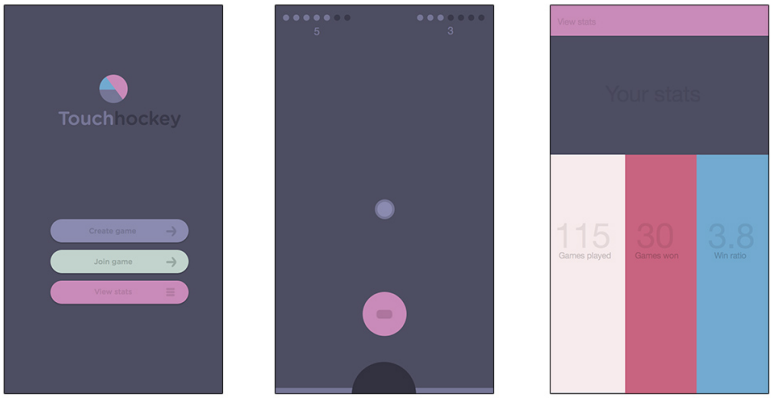


Figure 5.11: Main Game Screens : Menu screen,Main game,Stats screen

Upon completing the initial studies and deciding on the games screen flow and general design, high fidelity mockups of the game was then developed using sketch. The game followed a flat design approach aiming to promote simplicity and follow modern design trends rather than the more realistic skeuomorphism designs found in the older Air Hockey games. The decision to go with a flatter design enabled minimal assets to be used for game resources which results in less memory required and thus efficient load times.

Figure 5.12: In Compaision : Air Hockey Game With Skeuomorphic Design

5.1.10 Network Framework Design

When approaching the design of the Network framework for the software a lot of thought had to be put in to ensure that it could easily be integrated with a front end, which in this purpose is the game engine.

For this project, the framework had to be designed to handle the discovery and connection, and two-way communication between the devices and then integrated with a natively built game engine. At this time in the project the important implementations had been made and the platform and tools had been chosen (detailed later). The decision was made for Bluetooth to be the supporting wireless technology for the multiplayer gameplay. The diagram below summaries how the game front end and the Bluetooth framework aims to interact with one another.

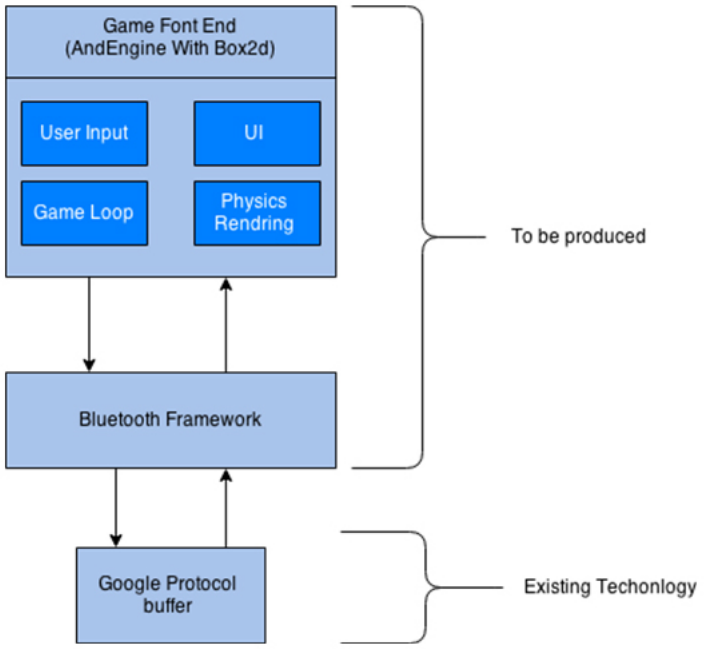


Figure 5.13: Interation between the game and underlying framework of the application

As detailed in the implmentation chapter, the game logic and visuals will be built using the open source AndEngine.With the Box2d library used to render in game physics. The front end will handle user input and will communicate with the bluetooth framework when networking tasks are required, the Bluetooth framework will be necessary to start a server or to join an existing game by connecting a device that has started a server. The bluetooth framework will then maintain the connection between devices and when messages are required to be sent between devices such as to update the position of the puck or update the score the front end can invoke methods from the framework which will

then use the google protocol buffer to serialize and send the mnessages and also deserialize messages that are recieved.

5.1.11 Test Driven Development

To approach the architectural design of the Bluetooth framework the methodology of Test Driven Development (TDD) was followed. TDD is a method that consists heavily of small iterations of writing a test, writing code then refactoring the code base. An overview of the process is written below.

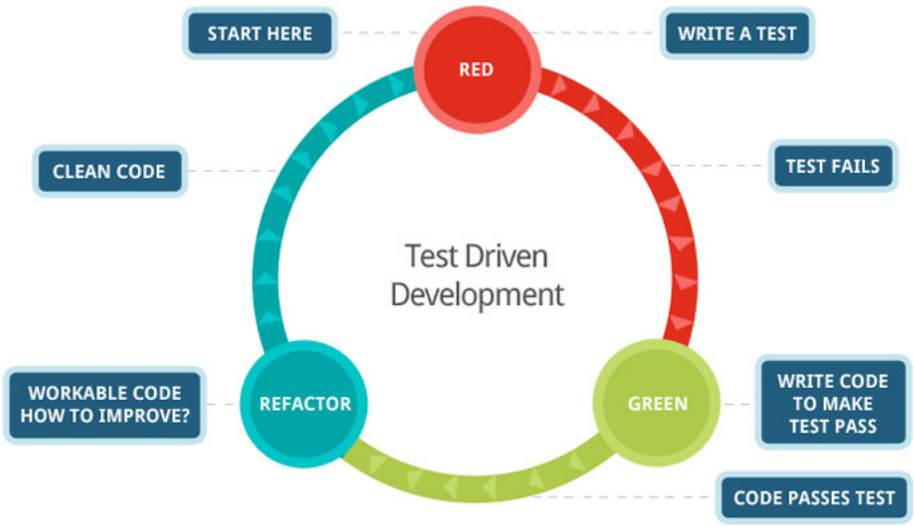


Figure 5.14: Test Driven Development Process

- Create a Test
  Based on the software requirements a small test is written. The purpose of this test it to define a desired improvement or new functionality.

- Run the Test
  All tests are run including the newly created one; the new test should fail implying the required improvement has not already been implemented.

- Make Minor Change
  Small code is then composed to pass the failed test. This is where the new functionality is implemented.

- Repeat
  The Cycle is then repeated.

This method was highly advantageous when writing the framework, It granted a better understanding of the functionality of the framework. It also allows testing to happen on

the go reducing the need for extensive testing once the framework was integrated with the game. This form of development is highly suited to a sole developer as it allows the framework to be built in small easily achievable steps whist. Although the testing ensures the code comes out highly functional the refactoring process also promotes high quality of code, which as an extensible framework is highly important. More detail on the implementation of this framework will be explained in the Implementation chapter.

6 Implementation

This chapter aims to detail the actual implementation of the whole software, which is the final android application. More explanation with any tools used in the implementation and problems encountered will also be detailed.

6.1 Main Implementation Decisions

As covered in the 4th chapter, there were various platforms and tools that were considered for the implementation of the application. This section will go though the decision for which technology to use for each aspect of the project followed by a justification.

6.1.1 Wireless Technology - Bluetooth

Bluetooth is a mature, widely prevalent and tried and tested technology used for wireless communication between mobile devices. The main reason it was selected was for its fully featured networking protocol implementation and for its prevalence unlike Wi-Fi Direct Bluetooth is present in 100 percent of android devices and although There is not clear direction on how to implement up a two communication stream between devices, Bluetooth full implementation of the 7 layer OSI model offers a vastly simpler solution when compared to Wi-Fi Direct.

6.1.2 Game Engine AndEngine with Box2d

Although Cocos2dx and Unity Engine are both heavyweight engines with large followings and support, it was decided that Andengine was the suitable tool for the task is relatively a lightweight engine and is tailored for the android device. Thus it is also written in Java (The Android native language) and would comfortably integrate with the native Bluetooth framework. Being a lightweight framework could also be advantageous as this results in less memory costs, which is beneficial to the end user. The Andengine library also provides a wrapper for Box2d the widely used and fully documented physics framework, which will ensure that the overall goal of creating a realistic air hockey simulator can be achieved.

Unfortunately Andengine is not officially documented which can slow down the implementation process. However it has been in use for over half a decade and provides plenty of tutorials and sample code to demonstrate its functionality.

6.1.3 Serialization Method - Google Protocol Buffer

As previously mentioned when it comes to the serialization of message data there also various tools to be considered. For this project the Google Protocol buffer was certainly the best choice as it was built specifically for the purpose of efficient serialization. According to the overview by Google:

Protocol buffers have many advantages over XML for serializing structured data. Protocol buffers:

- are simpler
- are 3 to 10 times smaller
- are 20 to 100 times faster
- are less ambiguous
- generate data access classes that are easier to use progamatically

This tool is wonderfully documented ensuring ease of its implementation. As previously mentioned efficient serialization of application data will greatly increase the speeds at which communication can take play between devices, which will directly affect user experience.

One disadvantage of the Protocol Buffer however is that a bit of work needs to be done to define application objects which need to be then compiled in the serializable objects before it can be used.

6.2 Implementing The Bluetooth Framework

The implementation of the framework was carried out separately from that of the game. This was due to the planned modularization of the framework allowing it to work with various different front ends. One major factor for the success of the framework is due to the ease in which it could be integrated with the game. This setup allowed this to be tested.

6.2.1 The Mediator Pattern

6.2.2 Client-Server Model

As per the documentation guidelines the networking structure for the Bluetooth framework should follow the Client-Server model. To create a connection between two devices,

one device must act as a server, this is done by holding an open Server Socket which will listen for incoming connection requests. The Server Socket must be assigned a 128-bit value known as a universally unique identifier (UUID). This ID allows clients to identify the server and request a connection. The second device acting as a client will then request a connection with the server and should it be successful a Bluetooth socket is provided two both devices which they can use to communicate with each other. The original Server Socket can be discarded to avoid any further incoming connections.

Communication between the server and client is achieved by sending data through output streams through the connection socket and then simultaneously reading the data through input streams.

The Bluetooth Framework was implemented to follow a clear structure and is centered around this model, it essentially provides the socket implementation alongside the data object serialization providing its user with an abstraction to the process of connecting and transferring data across devices.

6.2.3 Threading Issues

When an android application is started the operating system creates a single thread solely for that application. In this chapter we will refer to it as the main thread however it is sometimes referred to as the UI thread due to the fact that it handles all UI tasks such ass processing inputs and visually updating screen. So for instance, memory intensive tasks occurring on the main thread can cause major slowdowns on the user interface.

One challenge that is faced with developing this application for the android platform is that the networking code will cause blocking issues when running on the main UI thread. For instance if Server Socket is created on the main thread of the application this will cause the process to block, (footnote) as the execution of the code will halt until a client initiates a connection putting the thread to sleep. This means that the user will not be able to interact with the application as the user interface runs on the main thread.

According to the Android guidelines actions like this should be performed separate from the main thread. This can simply be solved on the server side by creating a new thread to listen for incoming connections and also using a separate thread to handle the communication streams between both devices as input streams will also block the thread as the halt the execution code whilst reading data. By employing multithreading in this manner, networking operations are kept separate from UI orientated tasks.

6.2.4 Android Service

However one issue still remains, threads are tied to Android Activities (footnote) which is the UI Component Class, if the user for instance changes the activity or momentarily exits the application in any form the thread will be lost. To solve this Android provides a Service (footnote) class, which can execute tasks without being bound to an Activity. This means that threads created in an activity will stay alive even when the user is not directly interacting with the application. This can prove to be extremely useful when completing long tasks such as maintaining a Bluetooth connection with another device over an undefined period.

6.2.5 ConnectionService Class

As mentioned Androids Service class is a useful tool for running long tasks such as maintaining a connection between two devices. The ConnectionService Class extended the service functionality and contained all the networking implementation. A UML class diagram generated using Visual Paradigm is shown in Appendix A:

Key Features The Connection Service class is started once the application is ran, the class contains the following code to start a Server;

```
BluetoothServerSocket serverSocket = mBtAdapter
   .listenUsingRfcommWithServiceRecord(appName, mUuid);

mBtSocket = serverSocket.accept();
serverSocket.close();

mBtDeviceAddress = mBtSocket.getRemoteDevice().getAddress();

Thread btStream = new Thread(new ServerThread(mBtDeviceAddress));
btStream.start();
```

This method will be called from outside the service to initiate the Bluetooth ServerSocket.

The server socket is setup using a static application uuid. Which will allow clients to identify it. As previously mentioned the execution will halt at serverSocket.accept() instruction and listen for incoming connection requests from client Devices. Upon the Client connecting the code will continue to execute and the server socket which is no longer needed is closed. The Resulting connection socket and client address is stored

for future use. An input stream is setup in a new thread as previously detailed to start receiving data from the client.

Sending Messages Across Devices As briefly mentioned, data on a device is received and transmitted through their respective input and output streams (footnote), Essentially these streams are easily accessible and allows data to be read and sent in bytes. The Google Protocol Buffer as mentioned provides the functionality of serializing data into bytes and deserializing bytes back to their original data format. The serialization of the data is handled by the protocol buffer library in which serializable objects can be built with one line of code with getters (footnote) used to initialize and class variables.

A code snippet of the protocol buffer classes is seen below[1]

```
message GameState {
    required MessageType type = 1;
    optional int32 puck_pos_x = 2;
    optional int32 puck_velocity_x = 3;
    optional int32 puck_velocity_y = 4;

    enum MessageType {
    PUCK = 0;
    SCORE = 1;
    RESTART = 2;
    }
}
```

The following class represents the information that will be sent from a device to notify the connected device on one of three major events.The enum MessageType defines the three types of events;

- PUCK : This message is sent to notifiy the connecting device that the puck should spawn on its screen. When this message is sent the optional puck fields will be initialised.
- SCORE : This message is sent to notify the connecting the device that a goal has been conceded. The connecting device can thus increment the score for the opponent.

[1]The '= 1','= 2' markers on each element identify the unique "tag" that field uses in the binary encoding.

- REMATCH : This message will be sent when the game is in the game over state, this notifies the connecting device that the user has requested a rematch.

This class can be built with the following code:

```
msg = HockeyProto.GameMessage.newBuilder()
  .setType(MessageType.PUCK).setPosX(posX)
  .setVectorX(velX).setVectorY(velY);
```

As shown in the code above the message object is built by first setting the message type from one of the three enums. In this case the message type is PUCK and thus the its x corodinate position is set(the y coordiante is not necessary as it will always appear from the top of the opponents screen). The velocity vector values are set in the same fashion.Once the built the class can be written to an outputstream seamlessly with another helper method from the protoBuf library;

```
msg.build().writeDelimitedTo(outStream);
```

As the input stream reads data byte by byte the size of the object being read must be previously known to ensure that the object is fully read and the input stream does not attempt to read an incorrect number of bytes. Therefore the writeDelimitedTo[2] method is used to write the size of the data to the output stream before writing the actual data. This allows more data to be written to the stream after the message without the need to delimit the message data yourself, as seen in figure below a parseDelimitedFrom on the recieving side to read the delimited message from the inputsream.

```
HockeyProto.Puck puck = HockeyProto.Puck
    .parseDelimitedFrom(inputStream);
```

So far the details behind the creation of a connection between two devices has been explained along with the sending and receipt of data. This process all takes place in a Service class, which runs in the background of the application as a seperate process. For the application component to communicate with the service futher implmentation is required .

6.2.6 Communication with the Service

Due to the service handling communication with remote clients the Android Messenger class was used as per the documentation guidelines to allow the core application to send messages to the service[3].The Connection class was thus to created to make use of the messenger class and to provide an API for communicating with the service.The connection class is a standard java class(No implentation of any interfaces), which handles all

[2]https://developers.google.com/protocol-buffers/docs/reference/java/index

[3]`http://developer.android.com/guide/components/bound-services.html#Messenger`

communication with the service and provides interfaces for callbacks for the application such as when a connection is made, message is sent or a disconnection. An overview of the class is shown in appendix A. It is essentially the outermost class for the framework and the application can use it can instruct the service to start a server, connect with a device and send a message to a connected device through this class. It will also receive messages back from the service. Once the connection class is initiated it will launch the service using an Android Intent[4] as shown below:

```
Intent intent = new Intent(mContext,ConnectionService.class);
 mContext.bindService(intent,mServiceConnection,Context.BIND_AUTO_CREATE;
```

Upon the successfully connecting to the service, the Connection class will receive an IBinder object used for Interprocess communication between the class and the service. A Messenger object can be created from this object which will enable messages to be sent to the service. The Android Messenger class has the following attributes:

1. **what** An integer, can be used to define the message type.
2. **data** A Bundle[5] which is acollection of key value pairs, used to hold message data.
3. **replyTo** Can hold a messenger object for the service to reply back to.

Constant fields were used to define the message types to be represent the what field such as:

```
public static final int MSG_START_SERVER = 100;
public static final int MSG_CONNECT_SERVER = 101;
```

So for instance to instruct the service to start a new server the following code is used:

```
Message msg = Message.obtain(null, MSG_START_SERVER,0,0);
msg.replyTo = messenger;
mService.send(msg);
```

[4]Android class - An abstract description of an operation to be performed. `http://developer.android.com/reference/android/content/Intent.html`

[5]Android class - A mapping from String values to various Parcelable types. `http://developer.android.com/reference/android/os/Bundle.html`

Instruct the service to connect to a device running a server:

```
Message msg = Message.obtain(null, MSG_CONNECT_SERVER,0,0);
Bundle bundle = new Bundle();
bundle.putString(DEVICE_NAME,deviceName);
mService.send(msg);
```

The Message.obtain is more efficient than using the new Message() constructer to create an object as it pulls a message from a pool of recycled objects rather than creating a new one[6].

Messages are received and sent through the Android Handler[7] class, which can be implemented to process messages received as desired. A Handler class is used in both the Connection and ConnectionService class to provide two-way communication. An example of this is shown below:

```
final class IncomingHandler extends Handler {
   @Override
   public void handleMessage(Message msg) {
      processMessage(msg);
    }
}

private void processMessage(Message msg){
    switch (msg.what) {
        case MSG_START_SERVER :
          startServer();
          break;
        case MSG_CONNECT_SERVER :
          Bundle bundle = msg.getData();
          String deviceName = bundle.getString(DEVICE_NAME);
          connectToServer(deviceName);
          break;
          ...
```

[6] http://developer.android.com/reference/android/os/Message.html
[7] http://developer.android.com/reference/android/os/Handler.html

With this two way messaging system between the Connection and ConnectionService class the application will only have to instantiate the Connection class and then invoke methods on the resulting Connection object. By passing the object interfaces the connection class can then notify the application on particular events. The interfaces in the Connection class are shown below:

```
public interface OnConnectionServiceReadyListener {
    public void onConnectionServiceReady();
}

public interface OnDeviceConnectionListener {
    public void onDeviceConnection(String device);
}

public interface OnMessageReceivedListener {
    void onMessageReceived(Bundle obj);
}

public interface OnConnectionLostListener {
    public void onConnectionLost(String device)
}
```

An application aiming to make use of the framework will thus have to implement these four interfaces to receive callbacks from the Connection Class. Therefore the Connection class is designed to receive implementations of these interfaces and then call its respective methods when desired. For instance The Connection class is instantiated with the following code :

```
mConnection = new Connection(context,
    serviceReadyListener,connectionListener,disconnectListener ,
    msgRecievedListener);
```

As shown the Connection class will receive the context that the application is running in along with the four interfaces to send callbacks. For instance when the Connection class a message from the ConnectionService it can notify the fronted of the application with the following code.

```
msgRecievedListener.onMessageReceived(message);
```

6.2.7 Framework Implmentation Summary

This section essentially detailed how the core application or in this case the frontend will communicate with the framework. The frontend can call Connection methods to start and connect to a server or send a message and can receive information back from the Connection class through the interfaces. The Connection class passes and receives messages to the Service using Android Messengers and Handlers. The Service will handle all the true networking code such as the socket programming and also will serialize data to be sent and deserialize data that is received form a remote device using the Protocol Buffer helper library.

6.3 Implementing The Game

The section will detail the main components of the TouchHockey game. As previously detailed the Andengine framework was used to implement the game including but not limiting to: The transitions between game scenes, the rendering of physics the handing and loading of all game resources.

6.3.1 Singleton Pattern

```
"Ensure a class has one instance,
and provide a global point of access to it."
```

The Singleton design pattern is a highly useful and simple design pattern, with great usefulness for small sizes of software seen regularly in game design. In this pattern the software architecture is designed to limit the simultaneous instances of desired classes to one instance per application. This is particularly useful when a class cannot perform correctly when multiple instances of it are available. The Singleton pattern also provides a global point of access to these classes for external components of the application. To follow the pattern a singleton class must be structured as follows[17]:

- Should not contain a public constructor. All constructors should be protected, prohibiting the external instantiation of the class.
- Should feature a static member of itself. This will hold only one instance of the class. This will be the single running instance of the class for the entirety of the applications runtime.
- The Singleton class should contain a method, which enables access to the single instance of the class.

For this project for use of singletons poses two main advantages:

1. Instances of singleton classes will not be instantiated unless needed. This, means that memory will be saved and optimizations of this kind are always favorable on mobile devices especially on the lower end..
2. Initialization takes place at runtime (Lazy initialization). Unlike static variables singletons are not initialized at compile time. This allows them to use information that is only present once the program is running (Such as loading cached information from the device). The lazy initialization also enables singletons to reference each other during their initialization..

6.3.1.1 Singleton Classes for TouchHockey

TouchHockey uses two singleton classes; the ResourceManager and SceneManager. TheResourceManager class will manage all game resources. It is designed to create all necessary game resources and load them upon request. It will also handle the unloading of resources to free up memory when they are no longer necessary.

SceneManager The SceneManager class acts as the main interface for all game scenes[8] allowing scenes to be appropriately loaded and unloaded upon request. The Scene Manager communicates with the ResourceManager to load the desired game resources and unload resources when a scene destroyed. This class contains references to all scenes and thus can trigger events in a scene when necessary.

Game Scenes The game contains four scene classes for each of the four game screens all of which extend a Base class (BaseScene) which provides s constructor to initiate required fields for the scenes such as the camera[9]. The Scenes are:

- **SplashScene** The startup scene displaying the splash screen[10]. It is used disguise the loading of the applicaion resources.
- **MenuOptionScene** The Scene used to present the user with the menu options to start, join a new game or view statistics.
- **GameScene** The Singleton class should contain a method, which enables access to the single instance of the class.
- **GameOverScene** The scene that will be displayed on the game ending. Allowing the user to decide where to return to the main menu or quit the game.

As previously mentioned the SceneManager contains methods to start and unload desired scenes. An example of starting the Menu Scene is shown below:

```
ResourceManager.getInstance().loadMenuResources();
menuScene = new MenuOptionScene();
setScene(menuScene);
disposeSplashScene();
```

[8] Andengine class: A scene refers to a substantial section of the game featuring sound, graphics and necessary logic. It is self contained and contains necessary assets to present a coherent and playable scenario to the player

[9] Andengine class: provides the projection dimensions for the current screen

[10] essentially the game logo

The SceneManager firstly instructs the ResourceManager to load the required resources for the scene before then creating the desired scene object. The scene then comes into the users view after the setScene Method which will call the createScene() method which is overridden by each of the Scene classes this method will contain the implementation logic for that scene. The previous scene (Splash Screen) is then disposed.

Following the singleton design pattern, only a single instance of the ResourceManager and SceneManager classes should be running for the entirety of the application lifecycle, as resources will need to be made available throughout the duration of the game and at least one scene will always need to be present and scene transitions will take also take place for the entirety of the application lifecycle.

6.3.2 Game Logic

The logic for the real time game is all contained within the GameScene class. On startup a puck will be loaded on each players device, depending on whether the player is a host of a client the puck will then spawn on the players screen or out of view respectively. To create the effect of the puck travelling from one players screen to another, the puck will only be visible on one players screen at any time. The diagram below aims to illustrate this effect.

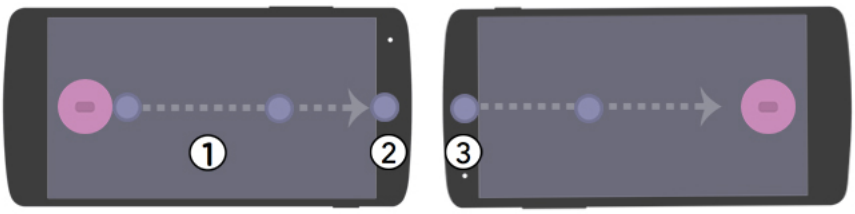


Figure 6.1: Puck travelling from device

1. The Player hits the puck
2. The Ball reaches the screen boundary and is held in place
3. The oppenents puck will then begin moving completing the effect

The collision detection and physics between the mallet and the puck are handled by the game engine. The main game loop runs every time the frame is updated, thus this would be the ideal place to check whether the puck has reached the boundary. If this is true then the puck is velocity removed and it is placed just outside the boundary out of the view of the player. A message is the sent to the opponent device using the Bluetooth framework detailing the pucks speed and pucks x position.

On receipt of this message the puck of the device as shown at the third label be will then the x position and velocity of the puck must be translated to give the complete the affect.

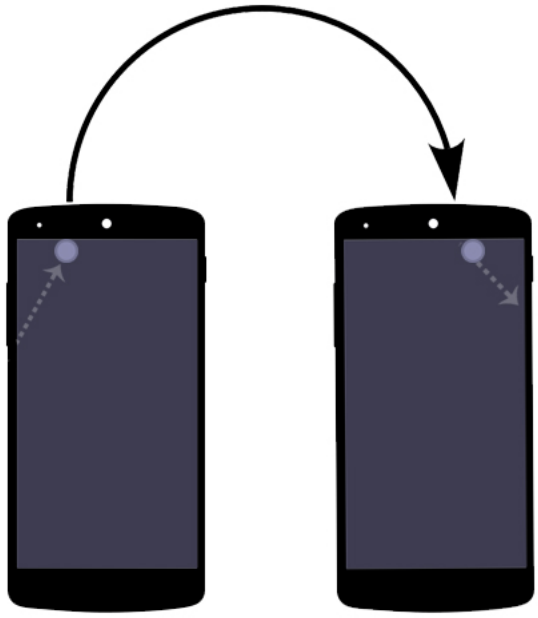


Figure 6.2: Puck travelling from device

As seen in the figure above, the pucks position on the x axis and velocity must be translated to ahcieve the seamless affect of the puck taveveling across the screens. The formula for the translations are shown below:

x coordinate translation:

```
new x Cordinate = absolute value of (old X coordinate - maximum X)
```

velocity translation

```
new veolcity = -(old velocity)
```

6.4 Problems Encountered

As with all software projects, there were various problems encountered throughout the implementation stage. One of the initial problems was due to Bluetooth discovery, The Android Bluetooth API will return all discoverable Bluetooth devices in range when running the discovery method. The problem with this however was that when playing a game, players will only want to discover nearby devices that are actually running the game and not a collection of Bluetooth devices (some of which may not even be phones).

To overcome this when running the application, the devices Bluetooth name which identifies it to other devices was programmatically altered slightly to contain a code specific to the application. This will mean that although the API returns all Bluetooth devices, devices not running the application could be easily disregarded. This however is not an ideal implementation as there is no way to ensure that the code is completely unique however it ensures that users are only able to pair with desired devices in majority of cases. Upon exit of the application the Bluetooth name can then be set to its original setting.

Another problem that was faced was in regards to testing, due to the sheer fragmentation of android devices it is important to ensure that software runs as expected on the variety of target platforms. There are many emulators that can provide virtual devices to test applications on however none of them come with Bluetooth support meaning that the core functionality of the game could not be properly tested for a range of devices.

This problem was difficult to overcome however when implementing the project the software was always run on two different phones with a vast difference in manufacture time(One much older than the other) and thus different Operating system versions to ensure backward compatibility. Dimensions used within the game where mainly calculated relative to the screen dimensions instead of being fixed to allow a scalable user interface for smaller or even larger screens. When testing the general physics of the game the Genymotion emulator was used allowing the application to be tested on 20 different devices of varying screen and Operating System versions.

7 Testing And Evaluation

7.1 Functional Testing

For this section each of the functional requirements defined in chapter 4.1 where assessed in turn. A complete list of test conducted can be found in appendix a.

7.2 Non-Functional Testing

7.2.1 Accessibility

As previously mentioned ensuring complete functionality of all android devices for all API levels is a difficult task. However the minimum requirements for the application where set to android SDK level 15 to ensure that the application can run on 95 percent of android devices[19]. The Bluetooth API supports this API level as expected and the framework should thus be fully functional at this level. Using Genymotion[1] emulators the functionality of the application at this level and above was as expected. Meaning the majority of users would be able to run the software on their android device with no issues.

7.2.2 Performance

The performance of the application was tested using Squares spoon framework[2]. Spoon is a testing tool and also provides performance analysis for android code running on multiple devices. It can track useful details such as user events, CPU usage and memory allocation and leaks. The tool was used alongside AndEngines Framework Per Seconds meter to ensure that the application runs smoothly with no memory leaks.

7.2.3 Recoverability

Early on in the duration of the project, after a loss of source code stored on a local drive. The source code was moved into a private GitHub repository. Removing the chance of further loss of code.

[1] `https://www.genymotion.com`
[2] `https://github.com/square/spoon`

7.3 Evaluation of Bluetooth Framework

One of the core goals of the Bluetooth Framework was to not only create a functional framework but also a portable one allowing it to be easily expanded upon and to work with various other implementations. To do this I had to ensure that it was built independently of the game logic. I believe that I could have abstracted the process a bit better as unfortunately the framework code does contain some implementation specific to the touch hockey game. Such as when sending a message the objects that are built and sent within the framework are protobuffers defined specifically for the touch hockey game. This means that to use the Bluetooth framework in another environment minor re implementation needs to be done to define the message objects. This however is still an improvement from re implementing the framework from scratch as the developer will only need to alter around 30 lines of code out of the 500+ lines of the framework. The implementation for creating server sockets, input streams and handling connections disconnection are fully portable and are independent of the system.

7.4 Evaluation of TouchHockey Application

I believe the TouchHockey game was an ideal one of showcasing androids Bluetooth capabilities and also the possibilities of multi screen gameplay. The goal was to develop a unique spin on traditional hockey games and the multiple screen capability added a fresh dimension to the game. Whilst there is certainly a lot more work to be done before the app can be released unto the Google play store, I believe that the TouchHockey game has shown that the Bluetooth technology is far from dead and can still provide unique and enjoyable gameplay in an offline environment.

8 Futher Work and Reflection

8.1 Futher Work

To fully productionise this application for it to be suitible for an app store futher implementation needs to be done. In regards to both the framework and the actual Touch-Hockey application, 2 main tasks need to be focused on to continue with this project.

- Multi-Player Functionality - In regards to the framework, it is currently only implmented to support a connection between a single server and a client. However the Bluetooth API supports up to six connections to a server. Futher work could be done to allow mulitple conenctions to support games and applications with more than two players.
- Single Screen Functionality - Although the multi screen functionality if a unique addition for the touckhockey genre an additional game mode to support multiple players on a single device could be added to allow gameplay on a single application instance.
- Quality Assurance - As with any game alot of QA testing needs to be undergone to ensure that it plays as expected. Due to the scope of the project and the requirement in man hours this was something that was not possible but may need to be consisdered before releasing to the Google play store.

8.2 Reflection

This project was challenging but also very rewarding. The networking aspect of the project was very difficult due to the lack of examples and detailed documentation availible. I am glad that the framework is functional and can be used elsewhere to hopefully save someone else the time investied in its implementation. The game was also a little success for me as alot of satisfaction comes from being able to see the final project in the form of something playable which others can enjoy. I am definetly excited to continue its development to hopefully release it on the Google Play Store by the end of this year.

References

[1] Anshul Srivastava *2 Billion Smartphone Users By 2015.* [2014]. "http://dazeinfo.com/2014/01/23/smartphone-users-growth-mobile-internet-2014-2017/"

[2] NDP Group *"Average Time Spent Playing Games on Mobile Devices Has Increased 57 Percent Since 2012".* [2015]. https://www.npd.com/wps/portal/npd/us/news/press-releases/2015/average-time-spent-playing-games-on-mobile-devices-has-increased-57-percent-since-2012/

[3] Erik Kain *"The Top Ten Best-Selling Video Games Of 2014".* [2015]. http://www.forbes.com/sites/erikkain/2015/01/19/the-top-ten-best-selling-video-games-of-2014/

[4] App Annie *"iOS Top App Charts".* [2015]. https://www.appannie.com/apps/ios/top/united-kingdom/games/?device=iphone

[5] Kevin Hoffman *"Windows Phone 7 for iPhone Developers",pg 256.* [2011]. Addison-Wesley Professional

[6] Quora *"Why arent there many multiplayer mobile games being developed".* [2014]. http://www.quora.com/Why-arent-there-many-multiplayer-mobile-games-being-developed

[7] Peter Stanford *"Are smartphones making our children mentally ill?".* [2015]. http://www.telegraph.co.uk/news/health/children/11486167/Are-smartphones-making-our-children-mentally-ill.html

[8] Fadi Chehimi and more *"Games on Symbian OS: A Handbook for Mobile Development".*[2008]. John Wiley and Sons

[9] John Donovan. *Bluetooth Goes Ultra-Low-Power* [2011]. http://www.digikey.com/en/articles/techzone/2011/dec/bluetooth-goes-ultra-low-power

[10] Killian Bell. *Google Play Will Surpass App Store Downloads Within Months.* [*Cult Of Android*]. http://www.cultofandroid.com/28820/google-play-will-surpass-app-store- downloads-within-months-report/

[11] Elise Vogler. *"Bluetooth vs. Wi-Fi Power Consumption".* [*Opposing Views*]. http://science.opposingviews.com/bluetooth-vs-wifi-power-consumption-17630.html

[12] Game Developer Magazine. *Mobile game developer survey leans heavily toward iOS, Unity.* [2012] http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php

[13] Laura Rohde *"Ericsson demos first Bluetooth phone"*.[2000] http://www.pcadvisor.co.uk/news/desktop-pc/100/ericsson-demos-first-bluetooth-phone/

[14] Bluetooth *"Skyrocketing demand for Bluetooth appcessories for latest phones"*.[2015] http://www.bluetooth.com/Pages/Mobile-Telephony-Market.aspx

[15] John Herman *"Why Everything Wireless Is 2.4 GHz"*.[2010] http://www.wired.com/2010/09/wireless-explainer/

[16] Erik Bohemia, Alison Rieple, Jeanne Liedtka, Rachel Cooper *Proceedings of the 19th DMI: Academic Design Management Conference , pg 1661*.[2014]. Design Mangement Institute

[17] Alan Thon *Game Engine Design and Implementation , pg 90* .[2011]. Jones & Bartlett Learning

[18] Jeff Friesen *Beginning Java SE 6 Platform: From Novice to Professional , pg 98* .[2007]. Apress

[19] Google Dashboard *Dashboards*.[2015]. https://developer.android.com/about/dashboards/index.html