Water Pipe Puzzle Game

CS39620 Minor Project Report

Author: Arran Jones ([arj23@aber.ac.uk](mailto:arj23@aber.ac.uk))

Supervisor: Dr. Myra Wilson (mxw@aber.ac.uk)

25th April 2019

Version 1.4 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science / Mathematics (GG14)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name: Arran Jones

Date: 25/04/2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Arran Jones

Date: 25/04/2019

Acknowledgements

I am grateful to Myra Wilson for her support and guidance throughout my project and maintaining consistent meetings to ensure I'm progressing as planned.

I’d like to thank my user testers for their efforts and the advice they've provided for my project.

Abstract

As the popularity of puzzle games continues to grow day by day, there is a distinct lack of levels available to the consumer. There may be several reasons behind this, but the major two reasons are:

* Pay walling in order to get more money from the players and in exchange give them a new set of levels
* A lack of future level additions to the game, due to a lack of support or a struggle to design unique puzzles.

The premise behind this project is to create a simple puzzle game that goes against this trend, while also maintaining the replayability of classic puzzle games.

This project would be a simple water pipe puzzle game that would generate its own random (but solvable) levels, increasing the replay value of game. The game would also save some basic statistics for the game in order to give the user a slightly more competitive feeling when playing the game at different difficulties, this should further help maintain a more replayable experience.

Contents

1. Background, Analysis & Process 2

1.1. Background 2

1.1.1. Background Preparation 2

1.1.2. Similar Applications 2

1.1.3. Functionality Research 2

1.1.4. Project Goals 2

1.2. Analysis 2

1.1.3. Functional Requirements 2

1.3. Process 2

2. Design & implementation 2

2.1. Overall Architecture 2

2.2. Sprint 1 - User Interface 2

2.3. Sprint 2 - Grid Generation 2

2.4. Sprint 3 - Implementing Search Algorithm 2

2.5. Sprint 4 - Determining Difficulty 2

2.6. Sprint 5 - Saving Statistics 2

2.7. Sprint 6 - Reading Statistics from Database 2

3. Testing 2

3.1. Overall Approach to Testing 2

3.2. Automated Testing 2

4.2.1. Unit Tests 2

4.2.2. User Interface Testing 2

4.2.3. Stress Testing 2

4.2.4. Other Types of Testing 2

3.3. Integration Testing 2

4.4. User Testing 2

4. Critical Evaluation 2

5. Annotated Bibliography 2

6. Appendices 2

A. Third-Party Code and Libraries 2

B. Ethics Submission 2

C. Code Samples 2

# Background, Analysis & Process

## Background

This project is a recreation of a classic puzzle game where the user rotates pipes in order of connect the starting pipe to the end. The difference in this project being it would generate its own levels using a search algorithm to determine the number of solutions each puzzle has.

As this is a minor project, compromises have been made in order to achieve more functionality, such as a more basic user interface than those currently on the market. This would allow more development time to achieve a better performing system, instead of concentrating on the visuals of the application and failing to fully implement the functionality required.

### Background Preparation

Before beginning the project, some major decisions had to be made, such as the platform the game would be made for, the language it would be written in and the search algorithm required to check for solutions.

**Platform and Language**

The first major decision that had to be made was whether the project would be a mobile or a desktop application.

Desktop has the advantages of more powerful hardware and larger screen sizes, allowing for more flexibility in grid sizes and processing power for the search algorithm used. This could potentially make the game run smoother and allow for some customisability, such as allowing the player to choose the size of the grid.

Mobile has the advantage of a more widely adopted touch screen interface. This would allow for a more intuitive interface and a cleaner feel for rotating the pipes. The game is also very simple and feels more suited to a mobile experience, as it’s the kind of game one would play when they have a spare five minutes with nothing else to do.

If mobile is to be used, there is still the decision of either Android or iOS. There are similarities between the two, such as the touchscreen for input, but there are very fundamental differences.

iOS is commonly seen as the easier mobile platform to develop for as it uses swift, a language based on Objective-C which was created by apple, specifically for iOS development. This would allow faster and more efficient development of the project, while maintaining a similar level of performance. A major disadvantage to iOS development is the requirement of XCode which is a mac exclusive IDE, preventing development on a Windows or Linux machine. There is also no prior knowledge of Objective-C, so there would be a large amount of research needed in order to complete this project.

Android development uses Android Studio, which is completely cross-platform. This is a major advantage as it would allow for production on any desktop or laptop, including mac if necessary. Android also has a predicted mobile operating system market share of around 72.77%, in comparison to apple’s 26.21% as of March 2019 (according to statcounter.com). This provides a much larger pool of users who may wish to download the app, increasing popularity, and in a real-world scenario providing greater monetary returns.

If Android is chosen as the mobile operating system of choice, there would be one final decision to make, which would be the language used to develop the application. Android studio officially supports two languages; Java and Kotlin.

Java is a very reputable object-oriented language which has been the standard language for developing Android applications for many years. It has the advantage of a large amount of documentation and a much larger community of programmers using the language to write support material.

Kotlin is both an object-oriented and functional programming language, this has the advantage of allowing users to use elements from both forms. The number of Android developers adopting Kotlin is increasing day by day, increasing the amount of support material available, but there was no prior knowledge of this language, therefore a relatively large amount of research would be required to learn the language.

**Search Algorithms**

When first researching the search algorithm that would be required for this project, the search algorithm the internet repeatedly suggested was an A\* search. This is a heuristic search algorithm that always finds the shortest path between two points by always choosing a path which decreases the Euclidean distance as much as possible at each vertex. Upon further thought, this would not make sense for this project as it is required to find every solution, not just the shortest.

After further research, an iterative deepening depth-first search was looking like a very strong option. This is a depth-limited version of a depth first search which is run repeatedly at incrementally larger depths in order to find the shortest path. This can be adapted to allow for the search to continue running until the whole graph has been searched but increases the processing power required.

In order to keep processing power to a minimum it was decided either a simple depth-first or breadth-first search would also be suitable. A depth-first search is a non-heuristic search in which a path is traversed for as long as possible, checking at each stage to see if it has reached its goal. If the target vertex is not found by the end of the path it would take a step back and check for another path to follow, it would repeat this until all paths have been traversed. A breadth-first search is another non-heuristic search in which all neighbours of the root node are checked first, if the target vertex is not found, then the search would continue to the next level of vertices (neighbours of the previously checked vertices.

### Similar Applications

The most similar application that was found on the play store was the challenge mode on Plumber by Magma Moblie[1]. Game has a 6x8 grid and is very professionally made, but falls into the category of set stages (on the application they’re called “worlds”).

Another similar application Pipe Turn by AnercEntertainment[2]. This application has an easy, medium and hard difficulty, but this game has the same issue as most puzzle games in that each difficulty has set levels, so the game only has a grand total of 49 puzzles. The difference between this game and the classic one is that this game has the beginning tile within the grid, whereas there was a set start and end outside of the grid that had to be connected to in the original. This game looks very well made and has a range of grid sizes from 4x4 to 8x6. The problem this game had with the 8x6 grid was that on my phone, the time bar half covered the bottom row of pipes, which I believe is due to the addition of an advert at the bottom of the screen:

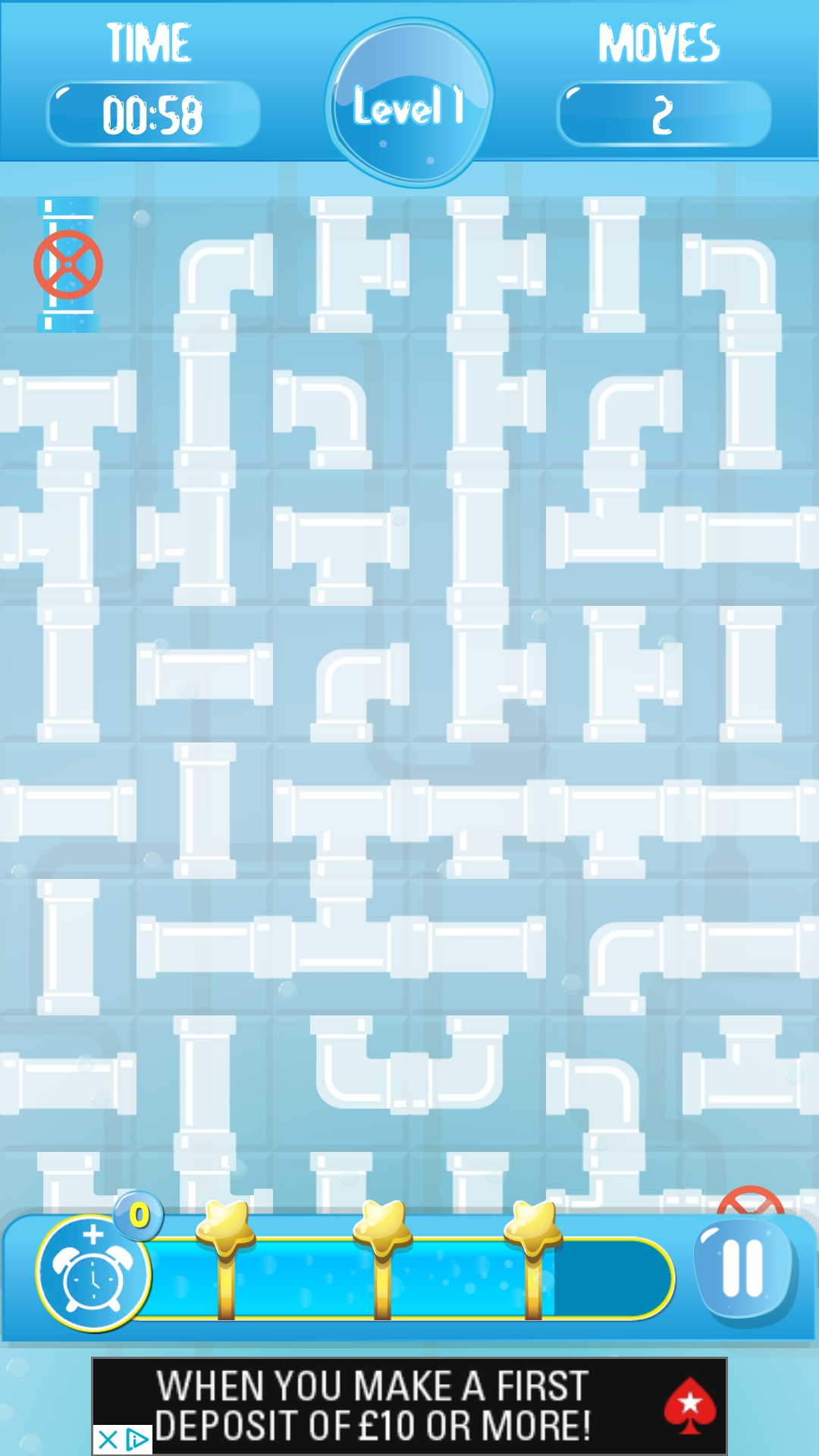


Figure 1. Pipe turn by AnercEntertainment

### Functionality Research

The main functionality research that would be required for this project would be Android development, as there was no prior knowledge of this topic. Android development with java is fundamentally similar but uses slight differences, such as the use of activities for the UI and a close integration with the Android SDK.

The majority of this research will be done by following the “Codelabs for Android Developer Fundamentals (V2)” training course on the Android Developers website. This is an introduction course to help programmers with prior Java knowledge learn Android development. For some of the other parts necessary for this project some of the other documentation provided on the Android Developers website would be used, along side some forum and other web searches.

The other functionality research required would be for the search algorithm itself as there was no prior knowledge of these either. This would entail the research already performed in the deciding of the search algorithm, the different ways of implementing a depth-first search and which way would be most suitable for this project and the steps that need to be added in order to deal with the changing connectivity of the graph.

### Project Goals

* **Create a fully functional self-generating game**

For this game to be considered functional, it should be able create a random solvable grid with a number the solutions determined by the difficulty the user selects. This should be done using a search algorithm and should not be pre-defined by the developer. The application should also be able to determine when a solution has been found by the user and notify them of the time taken to complete the puzzle and the number of pipe rotations it took the user to solve the puzzle. These solutions should then be saved into a database, where they can be used to determine some basic statistics.

* **Gain knowledge of Android development**

An aim of this project was to broaden knowledge for development on different platforms. This is essential for professional development and would be very useful for future projects that may be undertaken. A relatively good knowledge of Java was already there prior to this project, so Android development was a good place to start.

* **Gain knowledge of search algorithms**

Prior to this project there was little to no knowledge of search algorithms, which are an essential part of modern software development, therefore for further professional development, this was seen as a good project to undertake.

## Analysis

After the background research, some key decisions needed to be made.

Firstly, it was decided that a mobile operating system would be more appropriate for this project as it is a simple puzzle game. The touch screen interface of a mobile device makes the UI required for this game far more intuitive and the kind of game being created is much more suitable for a mobile game than a desktop game, due to the minimum amount of inputs necessary.

Once the platform was decided, the choice of operating system and language was a relatively simple one as there was still a lot of research that needed to be done before commencing the project. Therefore, Android would be chosen as the operating system, as the prior knowledge in Java and restricted access to a mac made this an easy choice.

After deciding that Java would be used, the next decision to make was what search algorithm to use. Initially, iterative deepening was looking like the best decision, but on further inspection this took far more processing power than a simple depth-first or breadth-first search. As this game works by connecting pipes, creating a single branch tree, in order to get from one point to another; the most logical search algorithm to use would be a variation on a depth first search. A basic depth first search would not be possible as some paths would not be traversable until a pipe had been rotated. Therefore, an extra stage would have to be implemented in order to check all possibilities for each rotation of each vertex.

### Functional Requirements

After analysing the problem, it was decided that there should be 5 main functional requirements:

* Generate a random solvable grid and determine when the grid has been solved
* Ensure the grid is solvable using a search algorithm
* Determine the difficulty selected and choose a suitable grid
* Save statistics for each difficulty into a database
* Read the statistics from the database and show them on a dedicated page

These seemed like the most sensible requirements for the project, as it provides a decent amount of content while maintaining a realistic amount of work for the time allowed.

## Process

It was decided that a Scrum methodology would be suitable for this project, as progressing week by week would help ensure the project would be complete on time while also allowing the planning for each sprint to be done at the start of the sprint. A logical process for each sprint would be to implement one of each of the functional requirements, with one additional sprint at the beginning for creating the user interface.

However, a variation on scrum is required as it is a solo project, therefore there would not be multiple people to fill the roles necessary for a scrum methodology (e.g. scrum master). This was chosen as the methodology as it is a good way of maintaining a solid management of time, with each sprint considering the amount of time left until the end date of the project. The sprints would each be a week long, with the first couple days dedicated to designing the work for that sprint.

Using the emulators provided by Android studio and the created APK files with physical handheld devices would ensure correct functionality across multiple device types and screen sizes.

The version control system that would be used is github. This is a very popular version control system which using git and is free to use. This would be used as the version control system for both the project and this report.

# 

# Design & Implementation

## Overall Architecture

As the project is being made in Android Studio using Java, a good place to start was to create a UML Class Diagram for the whole system. This Introduced a lot of early issues, but after restructuring it a couple of times it began to look much more structured:

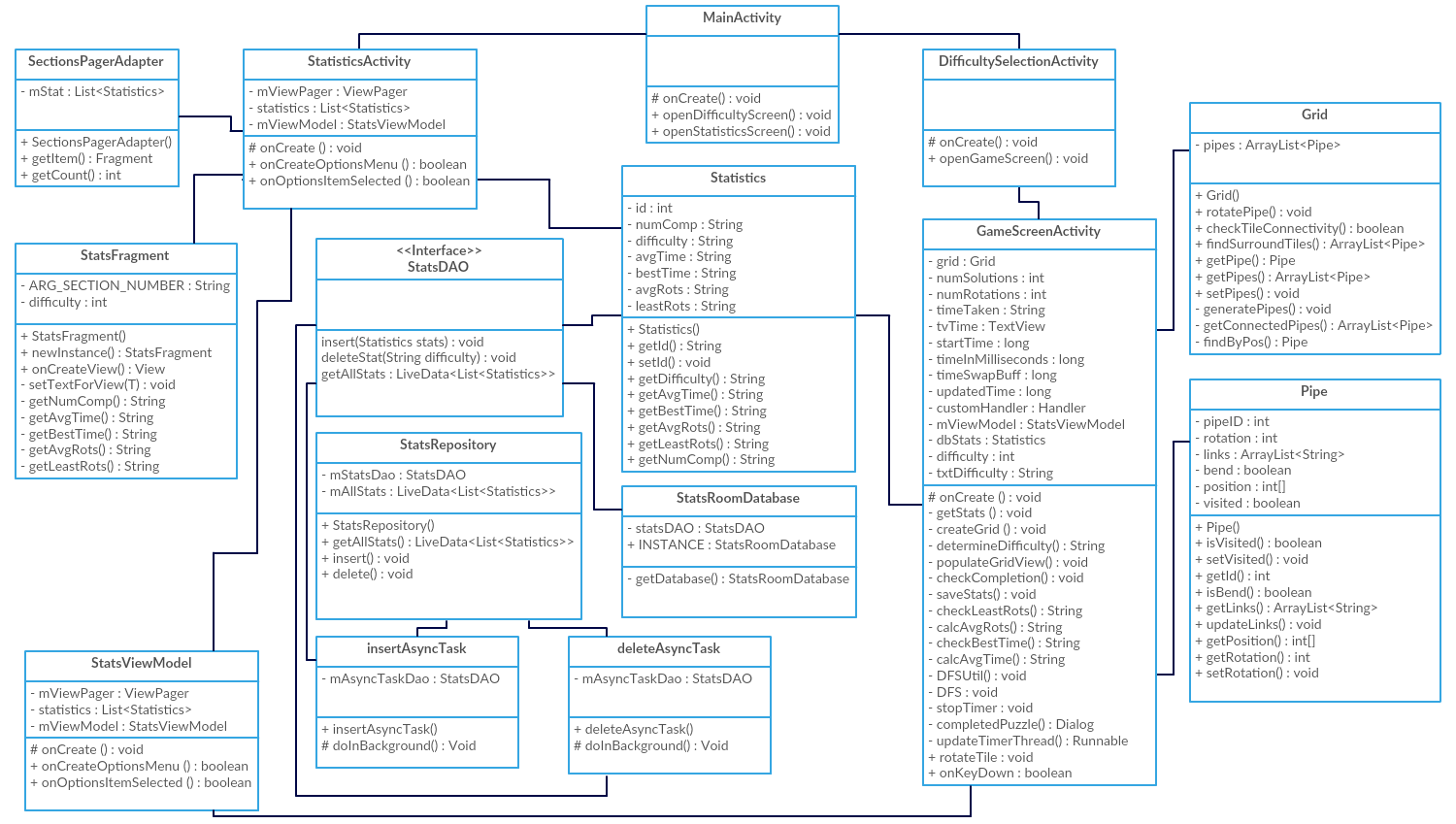


Figure 2. UML Class Diagram of entire project.

Creating this before beginning the project should allow for more efficient development once the implementation is started. In this diagram it can be seen that there are four activities (Androids word for screen), two objects for the game itself and one interface & four classes for the database.

## Sprint 1 – User Interface

**Design**

An aesthetically pleasing user interface is not the most important thing about this project as it is a minor project and getting a functional game is much more important in order to meet the goals of the project. In order to design the user interface FluidUI.com would be used. FluidUI is a website which is used to create prototype web and mobile prototypes with a drag and drop style interface. This would allow for a swift design of the user interface, allowing for more time to learn android development and more time to recreate the design.

The UI designed is very simplistic, but keeps a very easy to understand layout:

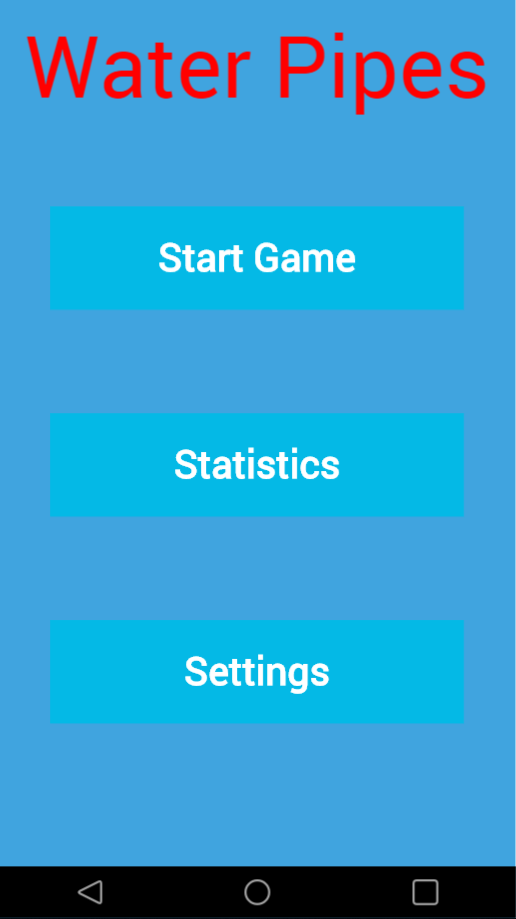
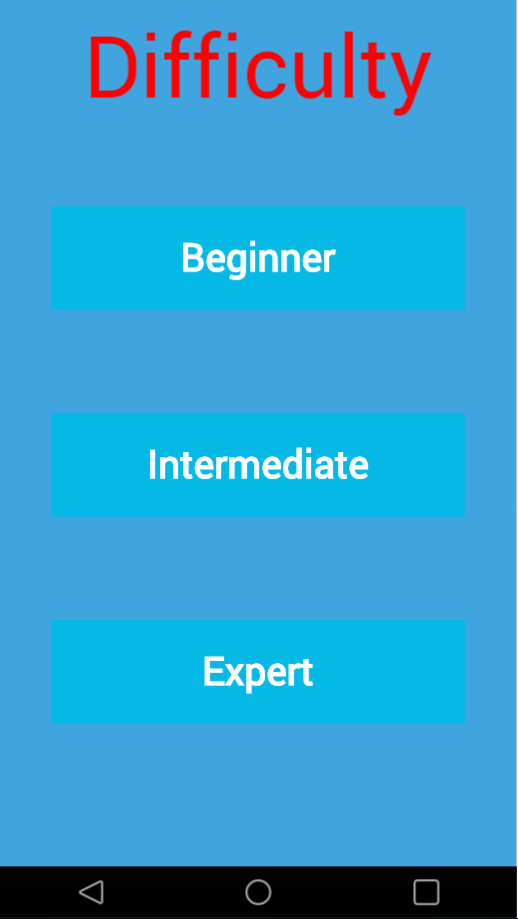
 

Figure 3. Prototype Main Activity Figure 4. Prototype Difficulty Selection Activity

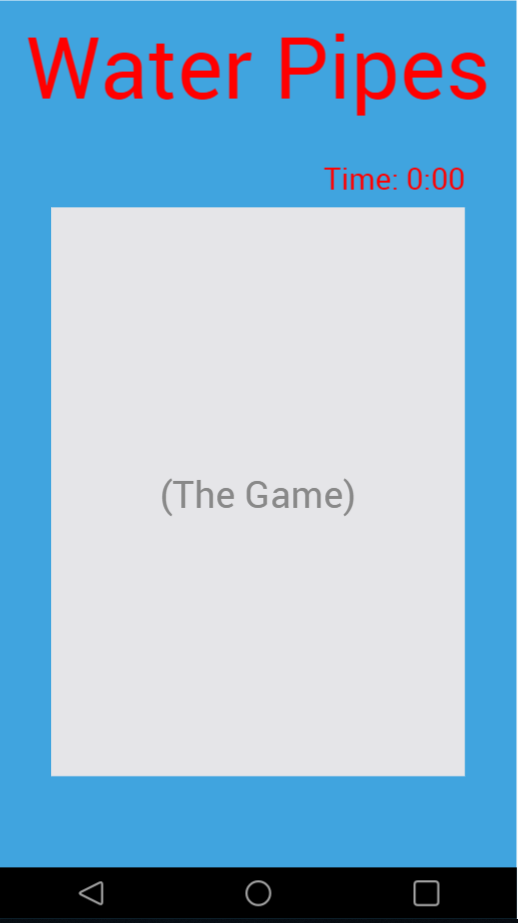
 

Figure 5. Prototype Game Activity Figure 6. Prototype Statistics Activity

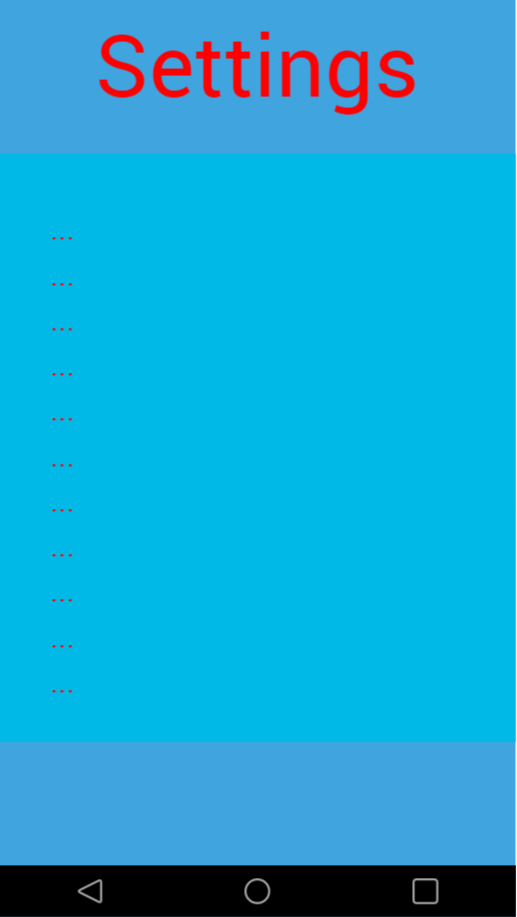


Figure 7. Prototype Settings Activity

The Prototype settings page lacks content as it was undecided as to what settings would be available to the user at the time, and the Statistics page only had a few items of text as the more time that was available at the end would determine how many statistics would be on the final product.

**Implementation**

The implementation of the user interface was relatively seamless, as Android Studio also has a drag and drop style UI builder, but there were a couple of changes made compared to the original prototypes, such as the removal of the Settings button and activity. The Settings button was removed as it is seen as bad mobile development practice to have a button to navigate the user to a settings page. The settings activity was then removed entirely as it was decided that adding settings may add more functionality to the application than is required, making it a larger install than necessary. The time was also moved on the game screen for aesthetic reasons, as there was a lot of blank space nearer the bottom of the screen.

The user interface ended up looking like:

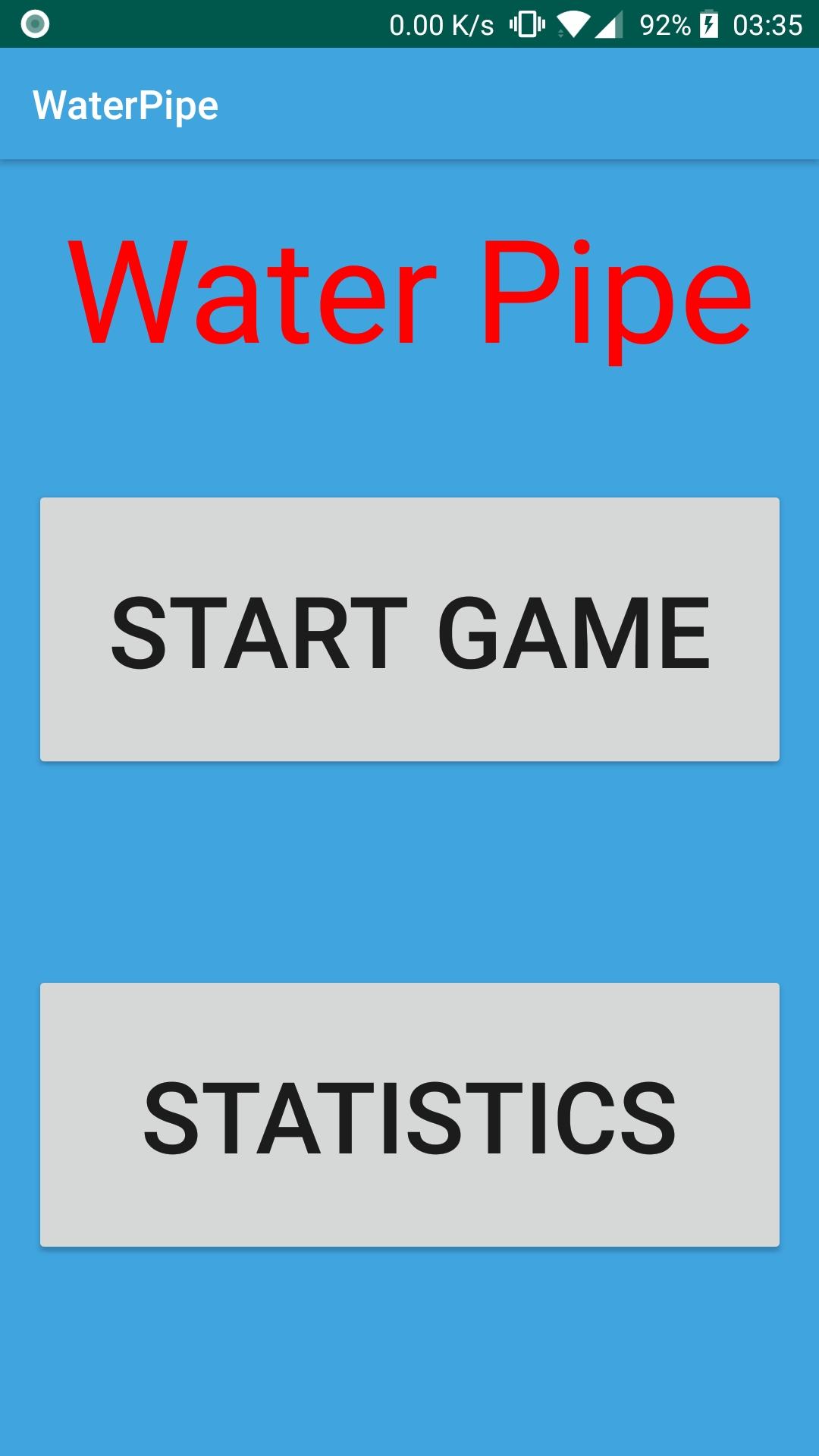
 

Figure 8. Main Activity Figure 9. Difficulty Selection Activity

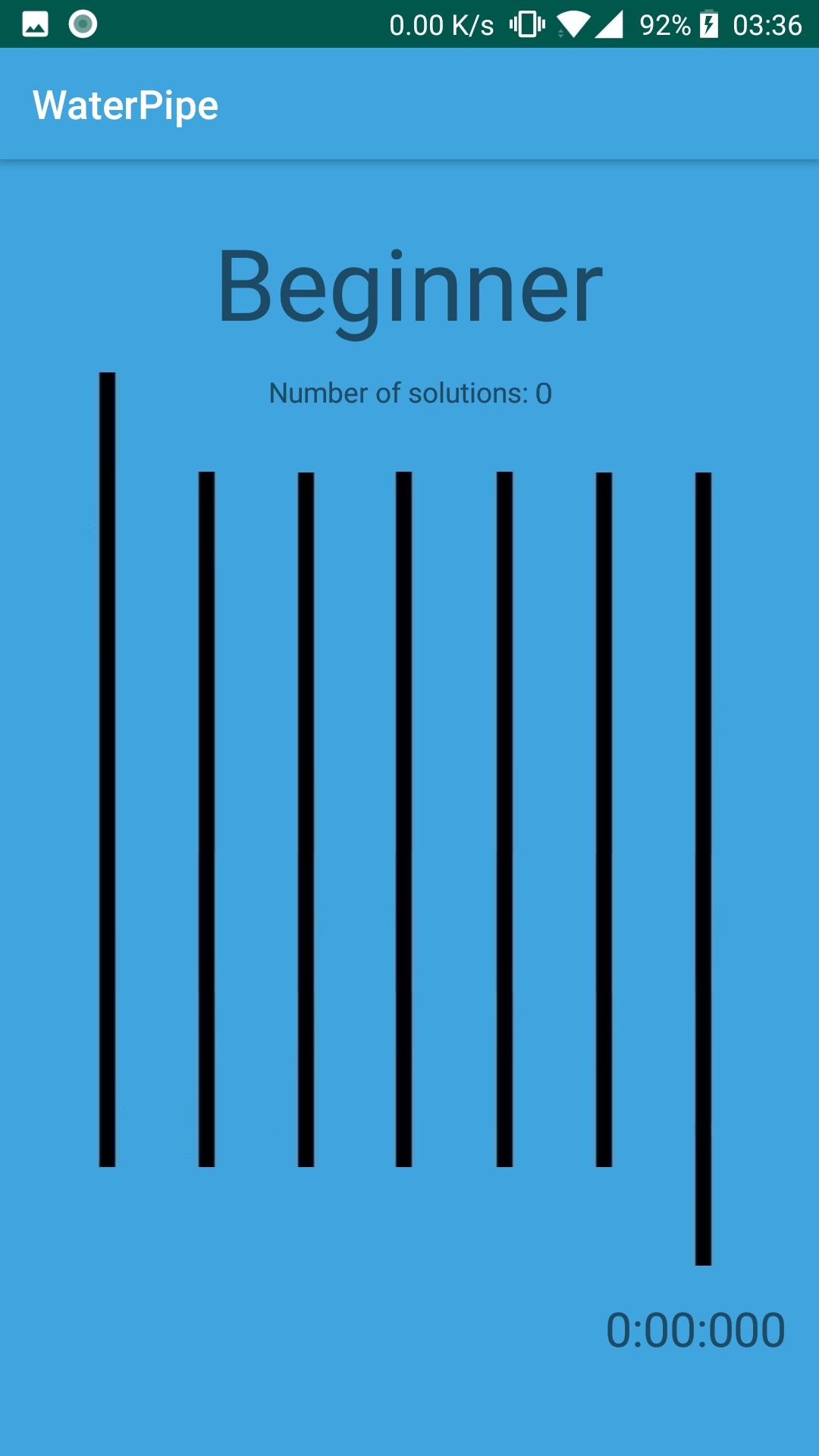
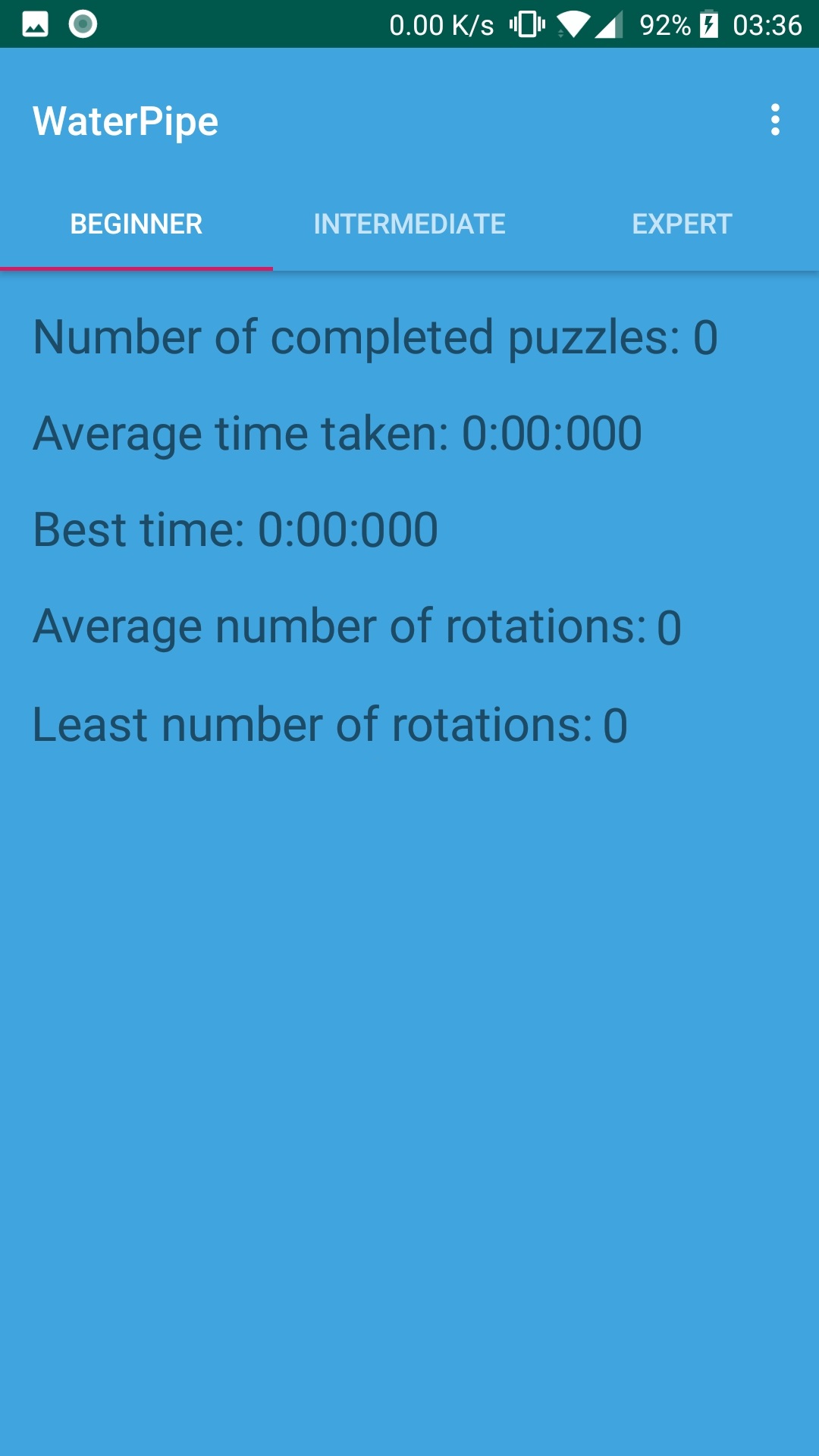
 

Figure 10. Game Activity Figure 11. Statistics Activity

Overall, the user interface stayed relatively close to the original designs, but upon uploading the game to a mobile, it was decided that the buttons should not be blue, as indirect sunlight it was tough to distinguish between the background and the buttons. It was also decided to add a text view to the game screen that shows the number of possible solutions for the puzzle and the difficulty the user chose. This helped fill in some of the empty space while also giving the user some information on the difficulty of the puzzle.

Having researched some other similar applications on the google play store, a 7x7 grid seemed like a good size for the grid, as it was large enough to give more possibilities, and make the puzzles slightly harder, while also accommodation many different screen sizes. Smaller screen sizes were the main concern, but having run it on several emulators, this seemed like a good size.

The grid on the user interface is made with a table layout, where 7 tiles are put on each row. The tiles are image views, and two more tiles were placed outside of the grid, one above the first tile and one below the last. This signal the start and end of the puzzle.

## Sprint 2 – Grid Generation

**Design**

The design for this section was mostly done earlier by the UML diagram. But a few things needed to be considered before implementing this section, such as how the pipes would relate to the image views, how the rotation of the pipe and links would remain synchronised and how their location in the grid would be calculated. For the relation to the image views, it was originally planned to search the grid by location with a check of the name of the image view and a switch case statement to determine location, but after more consideration, using the name of the image view as an ID would make the whole process much simpler. The links should be kept updated by relying them on the rotation of the pipe, so every time the rotation is changed, a method is called to update the links. For the final issue, a simple nested for loop from 0 to 6 should be enough to set the x and y co-ordinates for each pipe.

**Implementation**

The pipes were made into an object that contained all relevant information necessary for determining their location in the grid, the visited variable and the links that can be made by the pipe for the search algorithm, and some basic variables such as the rotation of tile and whether or not the tile is bent. There is also a pipeID variable that is used for linking the tile to the image view it was generated for. This is done by having the ID number be the same as the value at the end of the image view name. The necessary getters and setters were then added, and a method called “updateLinks” was created. This is a method that changes the links variable to the correct links, depending on the bend and rotation of the pipe.

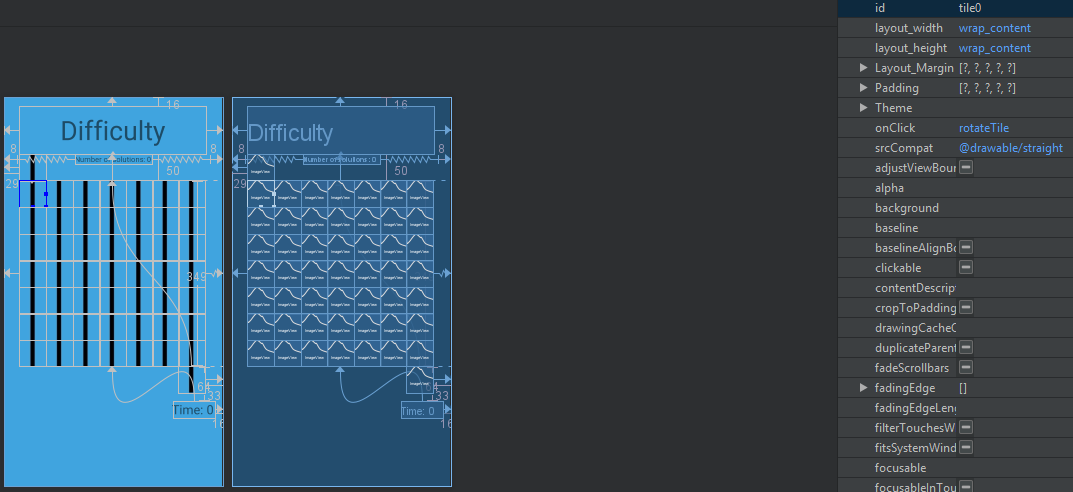


Figure 12. Showing that each image view has a number added to their id.

Java does not have a graph object natively, and it seemed unnecessary to design a graph and vertex object for such a small grid, hence the use of a position variable in the pipe object itself. Having decided this, the grid object is just an ArrayList of pipes. This class also contains most of the methods required to manipulate the pipes, such as rotation and finding pipes based on position or ID, or even just finding adjacent pipes.

The grid constructor calls a method called generatePipes, which completely randomly generates the 49 required pipes, while also setting their location in the grid and giving them their unique ID. The constructor of the pipes, then generates the pipes with a 70% chance of the pipes being bent. This is a completely arbitrary number that was chosen as it seemed to generate the most grids with a decent number of solutions.

## Sprint 3 – Implementing Search Algorithm

**Design**

Designing this in theory is a very simple task, it is a basic depth-first search with a rotation at each stage to adapt the grid in order to open new possibilities.

The basis of the depth-first search used is a recursive method found on HackerEarth.com:

DFS-recursive(G, s):

mark s as visited

for all neighbours w of s in Graph G:

if w is not visited:

DFS-recursive(G, w)

Table 1. The depth-first search algorithm that would be adapted[3]

I originally adapted this algorithm by making the following changes:

DFS-recursive(G, s):

mark s as visited

for all neighbours w of s in Graph G:

for i between 0 and 4:

if pipes don’t connect:

for i is between 0 and 4:

if pipe connected AND not visited

rotate w

else if pipeId == 48:

if connected AND a link == “down”:

solution found

else:

if pipe is NOT visited:

rotate w

DFS-recursive(G, w)

Table 2. First attempt at a search algorithm

**Implementation**

This sprint seemed like a straight forward one at the start, but after implementing the algorithm, there was a bug where the algorithm either found no solutions or ended in an infinite loop. After hours of debugging and moving from a recursive algorithm to an iterative algorithm, it was obvious that the algorithm needed to be completely re-designed. After 3 re-designs of the algorithm the bug remained, but after moving back to a recursive method and debugging for multiple more hours, the bug turned out to be due to 3 things.

The first problem found was that the pipes were set as visited but were never un-visited as the algorithm moved to another branch. This was obviously a major problem but was found relatively quickly. The second problem was the incorrect parameters of the if statements before rotations. This was a problem that took a couple of days to find but ended up being simpler than expected. As well as these problems, there was also one that was harder to find in the algorithms I haven’t shown, and that was marking the neighbour pipes as not visited in the for loop for each rotation, instead of the parent pipes at the end of the loop through its neighbours.

The algorithm ended up like:

DFS-recursive(G, s):

mark s as visited

for all neighbours w of s in Graph G:

for i between 0 and 4

if w is visited && pipes NOT connected:

rotate w

else if pipes connected AND pipeID == 48

if a link == “down”

solution found

rotate w

else if p NOT visited && pipes connected

DFS-recursive(G, w)

Rotate w

mark s as NOT visited

Table 3. The final algorithm

This was adapter slightly to make it more efficient, by only using 2 rotations for the straight pipes.

As well as this search algorithm, a simpler version was created to check for when the puzzle is complete on every rotation of a pipe. The algorithm for this search is:

DFS-recursive(G, s):

mark s as visited

for all neighbours w of s in Graph G:

if pipes connected AND pipeID == 48:

if a link == “down”

puzzle complete

else if p NOT visited && pipes connected

DFS-recursive(G, w)

mark s as NOT visited

## Sprint 4 – Determining Difficulty

**Design**

This is by far the simplest sprint of my project. In order to do this, the buttons that are used to open a new intent for the game screen activity just pass either a 0, 1 or 2 to the game screen intent. These values are then used to determine the difficulty. Following this, in order to set the difficulty of the game, the value passed through is used to choose the number of solutions for each difficulty. The difficulties are set as 1-2 solutions for expert, 3-4 for intermediate and 5-6 for beginner. Using a simple switch case statement.

**Implementation**

This was a very simple sprint to implement and only took a couple of hours to do. This was very helpful as the last sprint was not completed in time and required some of this sprint to finish.

Following the completion of this sprint there was some spare time before the next sprint. So, some simple things were added to the game; such as a dialog to warn the users that leaving the game screen would lose all progress, locking the screen to portrait on each activity and preventing multiple of the same activity being opened and the timer for the game that would be used to calculate some statistics later. Originally, multiple of the same activity could be opened if buttons were double pressed fast enough or the user moved between the activities in a certain order. This wasn’t a big issue but was a waste of system resources, so it was worth fixing in order to keep a clean feeling UI. It was at this stage a bug with the UI was found where the devices previously used to test the application were functioning as expected, but on installing the game to a smaller device, the pipes no longer aligned with each other. This was due to the constraints added. It was required to make the grid layout “wrap\_content” which prevents any white space around the contents of the grid, and to pin the tiles to each other rather than using margins.

## Sprint 5 – Saving Statistics in a database

In order to create a database for android, some Architecture Component libraries were added to the gradle files. These can be found on the Android fundamentals codelabs[2]. It follows that 4 classes and an interface are required to implement the database. The first is the entity class, this is the class that defines the column names of the database. In this class we also get to set primary keys and non-null fields.

The table name used would be stats\_table, just in case a future table needs to be added. The fields this table contains would be id, numComp, difficulty, avgTime, bestTime, avgRots and leastRots. These fields correspond to a unique id, the number of completed puzzles at the given difficulty, the difficulty, the average time taken to complete the puzzles, the best time, the average number of rotations to complete the puzzles, and the least rotations taken to complete a puzzle. This class also contains getters and setters for the database, but these would not work directly.

The next part of the database to implement the DAO, which is an interface that defines the SQL queries used to access the table and convert it to Java to be used elsewhere. In the interface, we would require three simple queries, an INSERT query, a DELETE query which would only delete the rows of a specified difficulty and a SELECT query which would retrieve everything from the table. These are simply INSERT, DELETE FROM stats\_table WHERE difficulty = :difficulty and SELECT \* from stats\_table ORDER BY difficulty ASC.

The next class is the room database, which is just a class which handles mundane tasks. In this case, it is merely to build the database.

The next class is a repository class. This isn’t a necessary class but is made to separate code and abstracts access to the data sources. In this case it is used to define the insert, delete and getAllStats methods which can be used by the view model to access the database and manipulate it as required.

The final class that is required is the view model. This is required for providing data to the UI and is necessary for the rest of the project to achieve access to the database.

After implementing the database itself, it would be required to calculate the statistics and update the database. This would be done using the getAllStats method to get the statistics, then looking for the statistics of the difficulty of puzzle that has just been completed. The stats would then be compared and calculated before using the delete query to remove the correct row, followed by inserting the new stats. This would prevent data duplication and keep all statistics up to date.

**Implementation**

Prior to researching this, there was no knowledge of how to implement such a database, so most of this sprint was preparation followed by a lot of debugging. The calculation and comparisons of the statistics were fairly straight forward, but the creation of the database took several hours to implement. The main difficulty with this was working out how each of the database classes related to one another. The main reason this took a while to work out was that one of the classes (the DAO) has to be an interface. This didn’t make any sense to me as I assumed one of the other classes would have to implement the interface, but this wasn’t true.

Once the four classes and interface were implemented, saving the statistics to the database and retrieving them to be updated was simple.

## Sprint 6 – Reading Statistics from Database

**Design**

The design for this sprint should be mostly straightforward as most of the database has been implemented. All that is required for this sprint is to use the getAllStats method to retrieve all the statistics and set the relevant text view’s texts according to the difficulty chosen on the tab view.

**Implementation**

While attempting to implement this stage, the lack of knowledge prior to beginning caused a lot of issues. For reading from the database, the list of statistics that was returned from the database had to be returned using LiveData. LiveData is a lifecycle library that keeps all data on a UI up to date. This added a few more hurdles as now an observer was needed, meaning it wasn’t as simple as just reading and setting the text of a text view.

After many hours of debugging it became apparent that the text views must be updated within the observe method, which is called on creation of the activity/fragment. This was difficult to find as it was a timing error and caused many different issues depending on the time it took to retrieve the data from the database. The only way that was found to get around this issue was to pass the difficulty and all text views to the observer as final, due to the nested method it contained.

The only other issue with this sprint was finding the tab index, as the activity chosen was written in a rather complicated way. But after following the debugger, it became apparent that there was an integer value associated with each tab. This was perfect for the project we created, as we had been defining the difficulty by an integer value too, so all that was required was to compare the equality of the variables to determine which difficulty was chosen. After doing this, a simple if statement would be plenty to find the information from the live data.

# Testing

## Overall Approach to Testing

The main approach to testing this project would been manual testing. This would be done as I am creating the application by running through the entire application several times and using a debugger to ensure values are as expected. There has also been some automated testing where possible. Then the application was given to real users for testing to get feedback and to see if any bugs could be found.

## Automated Testing

The majority of my testing would be manual testing, but for major parts of the application, such as the search algorithm, automated testing would be used. This would include unit testing for large methods, some stress tests for methods such as rotating pipes and some automated testing for the user interface.

### Unit Tests

I would be unit testing two of my methods, one is the depth-first search itself (including the DFSUtil method used to call set it up and call it) and the check completion algorithm. I would do this by passing known grids to the algorithms and checking if the result of the algorithm matches the expected value.

After many hours of trying to implement these test, it could not be achieved. To try and resolve this, many hours of research was done on the internet, but the solution could not be found.

As this could not be done in the end, it was necessary to run through the debugger on the game several times to see if the algorithms were working as intended, and it couldn’t be seen to be incorrect, but this is not conclusive proof.

### User Interface Testing

In order to test the user interface, Espresso test recorder would be used. Espresso test recorder writes test based on the Espresso Testing framework, which is an API in AndroidX Test. 7 Espresso tests were made using this method, and 4 passed without issue. The other 3 work as intended, but there is a runnable method on the GameScreenActivity that is called onCreate of the activity. This Runnable method runs until a the stopTimer method is called, but this cannot be done in an espresso test. Therefore, in order to run any tests which include the GameScreenActivity, Lines 68 and 69 must be commented out. Once this is done, they all run successfully and show that the interface functions as intended.

## User Testing

For the user testing, the users would be given full access of the application do use as they please. They would then be given 5 questions about the project to answer, where the feedback would then be reviewed.

As evidence of the user testers giving consent to using their names was not collected, they would be kept anonymous.

The questions asked are:

* Did you think that the number of solutions for each difficulty was reasonable? If not, what would you recommend?
* Did the difficulties suit the game, or would you have preferred to have seen another system?
* How intuitive was the user interface? Did it feel professional?
* Were there enough statistics, or were there some others you wished were displayed?
* Have you got any other comments on the system?

### User Tester 1

User tester 1 seemed relatively pleased with the project overall but found a bug in the game’s completion check. This was simple to fix, all that was needed was a single if statement to check if the first tile is in the correct orientation. This also had the lucky side effect of skipping the whole algorithm if the first tile isn’t in the correct orientation. Other than this, the user didn’t have much else to say other than it was a simple puzzle game with a suitably simple user interface.

### User Tester 2

User tester 2 seemed happy on the difficulty settings for the application but talked about adding a custom difficulty where the user can choose their own number of solutions. This would be possible to implement, but the larger the number of solutions, the longer it would take for a grid to be found by the algorithm. So, although this is a possibility, it would not be implemented in the app.

The user then talks about a progression system, this was already a consideration when designing the project, but seemed like a difficult thing to implement for the size of the project. This would have needed to use a mix of the minimum number of rotations needed to fix the puzzle and the number of solutions to the puzzle. Which again would be possible but would take more time to generate the grids.

The user answers the third question by talking about the aesthetic of the project, saying that custom animations and images would be nice to add. As mentioned earlier in the project, this was never a priority for this project as it is only a minor project and the functional requirements were much more important to implement. On the flip side, the user did like the simple and intuitive navigation.

The user also liked the statistics page as it gave more reason to attempt the puzzles multiple times.

### User Tester 3

In response to question 1 and 2, user 3 agreed that the number of solutions for each difficulty was a good choice but believed that the game should also utilise the grid size to make the difficulties more varied. This is a good decision but would require a complete reformatting of the way this program was made.

They then responded to question 3 saying that the red text on the blue background was a difficult contrast to see and made the text hard to read. This was not seen before due to colour blindness. Another comment was the standard colouring of the buttons. This was a decision made after changing the button colour to the original colour from the prototype design, which made it hard to distinguish the buttons from the background.

For question 4, the recommendation received was to have a statistic for number of puzzles attempted. This is a very good suggestion that may get added to a later version of the project. Question 5 had another good suggestion which was to add a back button to the activities. This was originally planned but every android phone has navigation buttons of one form or another, so this was seen as unnecessary.

# Critical Evaluation

Overall, I was pleased with the application I managed to create. It functions very well and having done no Android development or search algorithms before beginning the project, I achieved the core functionality that I had intended. All requirements of the project were implemented and having to modify the search algorithm made this a challenging process for a minor project.

My design decisions were mostly correct, the only change I would make if I was doing it again would be to keep a grid of each difficulty saved locally instead of generating them after the difficulty is chose, as there is currently an inefficiency where the game may have to generate several grids before it finds one of the correct difficulty.

The aim I added of saving statistics to a database was a challenging addition that took more time to implement than was expected. This was partly because of my lack of knowledge of implementing android databases, and partly because of the activity I decided to use. The current statistics activity mostly looks professional so I am happy with the way it turned out, but if I were to do it again, I would consider creating a custom activity instead of the standard one. This is because the code is not the tidiest and I feel as if it could be done in a better way.

If I was to do the project again, I would research structuring android better to make unit testing easier and research fragments more. Between these, I believe the project could have looked more professional and not had a relatively long pause after pressing the button for the chosen difficulty. These aren’t the most important part of the project, but it would have made the whole thing feel more professional had I known more about this side of android.

# Annotated Bibliography

This final section should list all relevant resources that you have consulted in researching your project. Each reference should also include a brief annotation.

1. Sylvia Duckworth. A picture of a kitten at Hellifield Peel. <http://www.geograph.org.uk/photo/640959>, 2007. Copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.

1. Mark Neal, Jan Feyereisl, Rosario Rascunà, and Xiaolei Wang. Don’t touch me, I’m fine: Robot autonomy using an artificial innate immune system. In *Proceedings of the 5th International Conference on Artificial Immune Systems*, pages 349–361. Springer, 2006.

This paper…

1. W.H. Press et al. *Numerical recipes in C*. Cambridge University Press Cambridge, 1992.

This is my annotation. I can add in comments that are in **bold** and *italics*and then further content.

1. Various. Fail blog. <http://www.failblog.org/>, August 2011. Accessed August 2011.  
     
   This is my annotation. I should add in a description here.
2. Apache Software Foundation (2014) “*Apache POI - the Java API for Microsoft Documents*” (Online) Available at: <http://poi.apache.org> Accessed: 14th March 2014.

This is my annotation. I should add in a description here.

1. Apache Software Foundation (2004) “Apache License, Version 2.0” (Online) Available at: <http://www.apache.org/licenses/LICENSE-2.0> Accessed: 14th March 2014.

This is my annotation. I should add in a description here.

1. Neil Taylor, “MMP\_S08 Project Report and Technical Work”, 2019 (Online) Available at: <http://blackboard.aber.ac.uk/> Accessed 19th February 2019.

A document that outlines information about the marking guide for the Project Report and Technical Work. This is published in the Resources folder on Blackboard.

# Appendices

* 1. Third-Party Code and Libraries

If you have made use of any third-party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you would make use of third-party code and software libraries. If third-party code or libraries are used, your work would build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

* 1. Ethics Submission

**Details**

**Submitted by:** arj23

**Date and Time:** 2019-03-25 18:40:06

**Current status:** closed

**Moderator:** Reyer Zwiggelaar [rrz@aber.ac.uk]

**Moderation message:** Minimal (daily equivalent) risk.

**AU Status**  
Undergraduate or PG Taught

**Your aber.ac.uk email address**  
arj23@aber.ac.uk

**Full Name**  
Arran Jones

**Please enter the name of the person responsible for reviewing your assessment.**  
Reyer Zwiggelaar

**Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment**  
rrz@aber.ac.uk

**Supervisor or Institute Director of Research Department**  
cs

**Module code (Only enter if you have been asked to do so)**  
CS39620

**Proposed Study Title**  
Water pipe puzzle

**Proposed Start Date**  
30/2/19

**Proposed Completion Date**  
3/5/19

**Are you conducting a quantitative or qualitative research project?**  
Mixed Methods

**Does your research require external ethical approval under the Health Research Authority?**  
No

**Does your research involve animals?**  
No

**Are you completing this form for your own research?**  
Yes

**Does your research involve human participants?**  
No

**Institute**  
IMPACS

**Please provide a brief summary of your project (150 word max)**  
A simple android based puzzle game, where pipes are rotated within a grid in order to form a path from a start point to an end point. All levels must be self generated using an AI search algorithm.

**Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?**  
Not applicable

**Will appropriate measures be put in place for the secure and confidential storage of data?**  
Yes

**Does the research pose more than minimal and predictable risk to the researcher?**  
Not applicable

**Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?**  
No

**Please include any further relevant information for this section here:**

**If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you would ensure you comply with this requirement should you identify that you require one.**  
Yes

**Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you would inform your department should the proposal significantly change.**  
Yes

* 1. Code Samples

**Depth-First Search**

This is the algorithm used when creating a grid, to find the number of solutions the grid has.

private void DFS(Grid searchGrid, Pipe src) {

src.setVisited(true);

for (Pipe p : searchGrid.findSurroundTiles(src)) {

if (p.isBend()) {

for (int i = 0; i < 4; i++) {

if (!p.isVisited() && !searchGrid.checkTileConnectivity(src, p)) {

searchGrid.rotatePipe(p);

} else if (searchGrid.checkTileConnectivity(src, p) && p.getId() == 48) {

if (p.getLinks().get(0).equals("down") || p.getLinks().get(1).equals("down")) {

numSolutions++;

}

searchGrid.rotatePipe(p);

} else if (!p.isVisited() && searchGrid.checkTileConnectivity(src, p)) {

DFS(searchGrid, p);

searchGrid.rotatePipe(p);

}

}

} else {

for (int i = 0; i < 2; i++) {

if (!p.isVisited() && !searchGrid.checkTileConnectivity(src, p)) {

searchGrid.rotatePipe(p);

} else if (searchGrid.checkTileConnectivity(src, p) && p.getId() == 48) {

if (p.getLinks().get(0).equals("down") || p.getLinks().get(1).equals("down")) {

numSolutions++;

}

searchGrid.rotatePipe(p);

} else if (!p.isVisited() && searchGrid.checkTileConnectivity(src, p)) {

DFS(searchGrid, p);

searchGrid.rotatePipe(p);

}

}

}

}

src.setVisited(false);

}

**Check for puzzle completion**

This is the method called when a pipe is rotated to check if the puzzle has been completed. If so, it would stop the timer, save the necessary statistics and open an alert dialog to notify the user of the completion of the puzzle.

private void checkCompletion(Pipe src) {

if(grid.getPipe(0).getRotation() == 0) {

src.setVisited(true);

for (Pipe p : grid.findSurroundTiles(src)) {

if (grid.checkTileConnectivity(src, p) && p.getId() == 48) {

if (p.getLinks().get(0).equals("down") || p.getLinks().get(1).equals("down")) {

stopTimer();

saveStats();

Dialog alert = completedPuzzle();

alert.show();

return;

}

} else if (!p.isVisited() && grid.checkTileConnectivity(src, p)) {

checkCompletion(p);

}

}

src.setVisited(false);

}

}

* 1. User Testing – 1

Did you think that the number of solutions for each difficulty was reasonable? If not, what would you recommend?

- I do believe the number of solutions was reasonable - being at least one for every game I tried, and less so as the difficulty improved.

Did the difficulties suit the game, or would you have preferred to have seen another system?

- I like the difficulty system, being harder to find a path due to less existence as you increase the difficulty.

How intuitive was the user interface? Did it feel professional?

- I had no issues with the UI. Though simple, it did feel clean and professional for the context of a small game such as this.

Were there enough statistics, or were there some others you wished were displayed?

- I personally had no need for statistics, so any added were a bonus. The quantity of them seems well thought out, not being too cluttered when shown back to you.

Have you got any other comments on the system?

- There is a slight bug where the first pipe can be in the wrong orientation and still think the puzzle is complete. Overall the game does its job well and was responsive.

* 1. User Testing – 2

Did you think that the number of solutions for each difficulty was reasonable? If not, what would you recommend?

I do like the amount of solutions for: beginner (5/6), intermediate (3/4) and expert (1/2) however, if you had more time maybe implement more solutions for the easier difficulty maybe even include a custom mode where you can enter the amount of solutions you want. Overall, I feel that this Is the correct amount of solutions for each difficulty.

Did the difficulties suit the game, or would you have preferred to have seen another system?

Each difficulty suited the game, there could have been different additions to that game such as a progression system, possibly at the start it’s easier and throughout each level the game gets progressively harder. This is just an alternative and something to think about next time.

How intuitive was the user interface? Did it feel professional?

The user interface was basic, if there was more time I would suggest adding custom images and icons to the game, instead of using buttons with possibly animation. On the other hand, the games interface is easy to follow so someone using this can navigate throughout the app.

Were there enough statistics, or were there some others you wished were displayed?

The statistics page displays good feedback for the user giving different scores. This can help the user compete against each other and make this more challenging to beat their own statistics, while maintaining the calm gameplay of a puzzle.

Have you got any other comments on the system?

Overall, I was happy about the game, you have added multiple solutions for the easy mode and little for the hard mode which gives verity to the game. If you’d of had more time implementing a cleaner UI system to the project would have made this look outstanding but this still has a brilliant interface.

* 1. User Testing – 3

Did you think that the number of solutions for each difficulty was reasonable? If not, what would you recommend?

Yes, I believe these are reasonable number of solutions for each difficulty. As the game space is quite small you cannot have like 10 different solutions. Therefore, the number is reasonable.

Did the difficulties suit the game, or would you have preferred to have seen another system?

The difficulties did suit the game, however you could have possibly had different sized puzzles for each difficulty. So easy had larger pieces so was a smaller puzzle. Then expert would have smaller pieces but be a larger puzzle and therefore more difficult to solve.

How intuitive was the user interface? Did it feel professional?

The layout was clearly laid out and every button was easy to read. However, it did not really look professional as the text is extremely large. The red is a little hard to see on the blue background without having an outline around the wording. And the buttons look like they have the standard colour and have not been changed.

Were there enough statistics, or were there some others you wished were displayed?

Possibly include how many times you have successfully completed it out of how many times you have actually attempted the puzzles. So, for example the statistic could you have complete 25/50 games successfully.

Have you got any other comments on the system?

Possibly put a back button on the actual app to go back to the main menu. Possibly just on the navigation bar at the top of the game.