

Submitted for the Degree of B.Sc. in Software Engineering, 2018-2019

---

## AI For Games: Being a Good Dad

---

Registration Number: 201426494

Name: Mauro Di Nardo

“Except where explicitly stated all the work in this report, including appendices, is my own and was carried out during my final year. It has not been submitted for assessment in any other context.”

---

Signature

---

Date

“I agree to this material being made available in whole or in part to benefit the education of future students.”

---

Signature

---

Date

# Abstract

The main goal of a game is to provide a fun and entertaining experience for its use. For many games AI help provide this experience by providing the player with challenges which the player, through skill and knowledge of the game, can overcome. However, modern games seem to fail in this regard, providing players at fixed difficulty setting which rarely match the skill level of the user. This results in laughably easy AI which provide no challenge and get boring or incredibly strong AI which are near impossible to beat and quickly become frustrating. The solution is a dynamic AI which adjusts its playstyle to better match the users' skill level and thus challenge them in a way that better engages them. This project will address two player abstract strategy games, specifically the game of Ultimate tic tac toe.

Adaptive AI can be thought of as a kind of "Good dad", an opponent which isn't necessarily playing to win, but instead playing to give the opponent a more fun experience. To do this the "Good dad" may lower his level of play to that of his child so that they always have a chance of winning if they play well. This project aims to mimic this same type of interaction between a "Good dad" and his child by using different measures of the users' skill level and different approaches for selecting a move.

This project demonstrates some techniques for producing AI which are intended to provide exciting gaming experiences. The accuracy of the result may be an issue, however, with some minor changes and some future experiments could alleviate this problem. The project has produced a solid foundation for future work in adaptive AI systems in two player abstract strategy games.

**Key Words:** AI, Games, Ultimate Tic Tac Toe, Monte-Carlo Tree Search, Dynamic Difficulty, Web App

# Acknowledgements

Firstly, I would like to thank my supervisor Dr. John Levine for his support, ideas and inspiration. He provided an engaging environment and nurtured my creativity and allowed me to produce a product that I am proud of.

I would also like to thank Phil Rodgers, for making time for me and provided useful insight to help develop some of my ideas further than I could have done.

I would like to thank everyone who gave up their time to participate in my experiments.

Finally, I would like to thank my friends and family for all the support they have given me through my entire university career and through this project.

# Contents

1.	Introduction .....	1
1.1	Objectives.....	1
1.1.1	Additional Objectives .....	1
1.2	Report structure.....	2
2	Related Work .....	4
2.1	m, n, k Games.....	4
2.2	AI in Modern Games .....	4
2.3	Adaptive Behaviour.....	5
2.4	Monte Carlo .....	6
3	Problem Description and Specification.....	9
3.1	The Problem .....	9
3.2	The Good Dad .....	9
3.3	The Game .....	9
3.4	Specification.....	10
3.4.1	Ultimate Tic Tac Toe.....	10
3.4.2	Rules.....	10
3.4.3	The AI .....	11
3.4.4	The Good dad AI.....	11
4	System Design .....	12
4.1	Methodology.....	12
4.2	System Architecture.....	13
4.3	Design Patterns .....	13
4.4	Observer.....	14
4.5	Composite .....	14
4.6	Chain of Responsibility.....	15
4.7	Dependency Injection .....	16
5	Design and Implementation.....	17
5.1	Model.....	17
5.2	Requirements.....	17
5.3	Design.....	17
5.4	Implementation .....	18
5.5	View .....	19
5.6	Requirements.....	19
5.7	Design.....	20
5.8	Libraries.....	21

5.8.1	Bootstrap.....	21
5.8.2	Ngx-toaster.....	21
5.9	Modals .....	22
5.10	Toast.....	22
5.11	Login Screen Design .....	22
5.12	Game View Design .....	23
5.13	Navigation .....	23
5.14	Controllers.....	23
5.14.1	Client .....	23
5.15	Server .....	25
5.15.1	Requirements.....	25
5.15.2	Design.....	26
5.15.3	Implementation .....	26
5.15.4	Maintenance .....	26
5.16	Services .....	27
5.16.1	Client .....	27
5.16.2	Server .....	29
5.17	Providers .....	31
5.17.1	Design.....	31
5.17.2	Implementation .....	32
6	Validation and Verification .....	33
6.1	Testing Strategy .....	33
6.2	Mocking.....	35
6.3	Jasmine.....	35
6.4	Microsoft Unit Testing .....	36
6.5	AI testing .....	37
7	Results and Evaluation .....	38
7.1	The AI .....	38
7.2	Further Improvement .....	43
7.3	User Evaluation .....	44
8	Summary and Conclusion .....	47
8.1	Summary .....	47
8.2	Game Implementation.....	48
8.3	Good dad Summary .....	48
8.4	Future work.....	49

## Table of Appendixes

Appendix A – User Guide .....	53
Installing Frameworks .....	53
Running the project .....	53
Testing the project .....	54
Login .....	54
Game Setup .....	54
Game Play .....	55
Making a move .....	55
Winning a sub board .....	56
Winning a game .....	56
Game over screen .....	57
Appendix B - Maintenance Guide .....	58
Players .....	58
Data Access .....	58
View .....	59
Appendix C – The Stealing Strategy .....	60
Example of the strategy .....	60
Appendix D – User Evaluations .....	61
Participant Information Sheet .....	61
Participant Consent .....	64
Participant Questionnaire .....	65
Evaluation results .....	66
Appendix E – Git commit log .....	67
Appendix F – Project plan .....	68
Appendix G – Web Application address .....	68
Appendix H – SQL .....	68
Appendix I – Bandit Problem .....	69

## Table of Figures

Figure 2-1 MCTS diagram taken from wikipedia (Wikipedia, 2017) .....	7
Figure 3-1 UTTT Board .....	10
Figure 3-2 Progression of game adapted from (George & Janoski, 2016).....	11
Figure 4-1 Agile Development Cycle .....	12
Figure 4-2 Composite Design pattern UML.....	15
Figure 4-3 Chain of Responsibility UML .....	15
Figure 5-1 Tile makeMove method .....	18
Figure 5-2 TicTacToe makeMove method .....	18
Figure 5-3 example input and out put .....	20
Figure 5-4 Component decorator of UTTT class .....	20
Figure 5-5 Example of dynamic DOM allocation.....	20
Figure 5-6 Example of Service call .....	21
Figure 5-7 ModalService being injected into class.....	21
Figure 5-8 Example of toasts.....	22
Figure 5-9 Login screen .....	22
Figure 5-10 Wireframe of game UI .....	23
Figure 5-11 Navigation bar.....	23
Figure 5-12 Example of calling GameService .....	24
Figure 5-13 Example of extended code .....	25
Figure 5-14 Example of end point.....	26
Figure 5-15 BaseControllers ExecuteApiAction method signature.....	26
Figure 5-16 Example of server-side service call.....	27
Figure 5-17 Wrapping of HttpClient object.....	28
Figure 5-18 Game Services public fields .....	28
Figure 5-19 Data access layer UML.....	31
Figure 5-20 Database diagram .....	32
Figure 6-1 server-side test coverage.....	34
Figure 6-2 Example of test written to catch a bug.....	34
Figure 6-3 Example of mocking in Jasmine .....	35
Figure 6-4 Jasmine test .....	35
Figure 6-5 Angular file structure .....	35
Figure 6-6 Example of mocking in MOQ .....	36
Figure 7-1 Average optimal reward for player one and two .....	38
Figure 7-2 Average length of game for player one and two .....	38
Figure 7-3 Progression of game .....	39
Figure 7-4 Graph showing Good dad not choosing optimal move .....	40
Figure 7-5 MCTS vs Good dad win ratio.....	40
Figure 7-6 Random vs Good dad win ratio .....	41
Figure 7-7 MCTS vs Good dad v2 win ratio.....	41
Figure 7-8 distribution of reward for Minefield AI .....	42
Figure 7-9 Reward for Good dad and MCTS during a game .....	43
Figure 7-10 User reported difficulty .....	44
Figure 7-11 User reported enjoyment of each AI .....	44
Figure 7-12 User game results .....	45
Figure 7-13 Distribution of users reward .....	45

## 1. Introduction

Traditionally AI implemented for games are split into three categories, easy medium and hard. The problem with this is that players will often struggle to find an opponent which provides a fun and engaging experience. Instead they will either be bored by opponents which are far too easy for them or frustrated at opponents who make near perfect plays. These emotions directly contradict the experience someone looks for when playing a game. This project will attempt to create an AI opponent which addresses these issues, by creating a fun and engaging gaming experience for users of all skill levels.

### 1.1 Objectives

The primary objective of this project is to develop an AI system which will provide an engaging gaming experience for users of all skill levels. This can be summarised well by the project brief:

*"Some people have proposed that the AI players of games need to be like "a good dad" when playing their children at games: not play too hard, but put the player into situations which are challenging and interesting."*

This project will attempt to implement an AI which mimics the user's playstyle, and so put players in situations which they may find challenging but ultimately should be able to overcome.

#### 1.1.1 Additional Objectives

The project does have some supplementary goals which must be completed to either support the adaptive AI system or to measure how well it adapts. These include:

Objective	Description
Game Selection	The game should be quick to learn and play
Game development	The game must be implemented and allows different players to easily make moves
Standard AI	Different AI at different skill levels must be implemented to measure performance of adaptive AI



<b>Profiler</b>	A system must be implemented to gain knowledge of players play style
<b>Adaptive AI</b>	An AI system must be implemented that leverages knowledge gained from the profiler and uses it to pick moves.

Completion of these objectives will provide a robust and hopefully enjoyable adaptive AI system.

## 1.2 Report structure

This report has been structured around the timeline in which work was completed. There are eight chapters of content in this report. The seven chapters following this introductory chapter are:

### **Related Work**

This chapter will present work in the field of adaptive AI and some of the theory behind the methods used.

### **Problem Description and Specification**

A detailed description of the problems tackled by this project. It will include the specifications for this project and how they were reached. Moreover, it will describe the way this project will be carried out.

### **System Design**

This chapter will give a high-level overview of the system architecture. It will describe the architecture choices for both the server and client, as well as how their communications are designed. Furthermore, it will highlight the advantages that these architectures have, and some of the design patterns used in development.

### **Design and Implementation**

Discussion of the implementation details of this project. Giving greater detail about how abstract design patterns were realised. It will also discuss some

changes that were made during development and what issues these changes were in response to.

## **Verification and Validation**

This chapter will describe the testing strategies used in development. It will discuss the framework and techniques used to validate the code. It will also explain the rationale behind the design of the tests.

## **Results and Evaluation**

The resulting AI and the application will be discussed in this chapter. It will describe the performance of the adaptive AI against both other AI and users. It will present the results of experiments and what those results represent. Additionally, it will describe how the findings impacted certain design decisions.

## **Summary and Conclusion**

A summary of the report, addressing the success of the project in completing the original objectives. It will also recant how the project went and what can be done in the future to extend the project.

## 2 Related Work

---

In the field of games, AI have been a focal point in creating interesting and engaging gaming experiences. This differs from the Computer Science view of AI in which the goal is to make intelligent systems. In this project providing in engaging game is far more important than the intelligence of the player. This chapter will describe the work related to games; what people find interesting and fun in games. As well as, research into different methods for constructing AI and some of the technical aspects of games related to the one used in the project.

### 2.1 $m, n, k$ Games

An  $m, n, k$  Game is one in which players take turns placing coloured stones on an  $M$  by  $N$  board, the winner is the first to get  $k$  stones in a row. Popular examples of  $m, n, k$  games include Connect 4 and Tic Tac Toe, a typical tic tac toe game would be a  $3, 3, 3$  game. Games up to  $7, 7, 5$  and  $8, 8, 5$  have been solved, player two can force a draw in both instances (Wei-Yuan Hsu, 2019). The stealing strategy argument (Appendix C) shows that in  $m, n, k$  game player two cannot have a strategy which ensures they win. As these games have finite size the game is convergent, all games finish after some number of moves, the max number of moves available being  $m \times n$ . These are also zero-sum games as there are only three possible end states: a win, loss or draw. Moreover, these games have perfect information and are stochastic, this makes them great candidates for tree searching techniques such as Monte Carlo, or depth first search.

### 2.2 AI in Modern Games

Typically, in modern games AI are divided into three categories; Easy, Medium and Hard. The issue with this approach is that it either provides an AI which is too easy for the player, or one which is too difficult which can quickly become frustrating for the player (Chanel, 2008). To combat this issue game developers have turned to techniques known collectively as Dynamic Game Difficulty Balancing (DGDB). This is a process which measures a player's ability in some form, whether it is accuracy in a First-Person Shooter (FPS) or number of pieces won or lost in a board game. The data collected is then used to modify parameters in an attempt to keep players entertained. There are

many ways this can be implemented, one is to passively help the player by giving them more resources, such as health packs and ammo, in an FPS setting (Hunicke, 2004). Though this system was found to be effective it did result in situations which were not intended. A solution to this is to use the data collected to modify the behaviour of Non-Playable Characters (NPCs). To take this a step further, the 2014 game “Shadow of Mordor” featured what it called the “Nemesis System”. This was a system that tracked hostile NPCs and their encounters with the player, the NPC would then comment on their previous encounters in their next meeting with the player. This coupled with the NPC’s personality, gave the player a more in-depth experience with the game as interactions were more dynamic and let the player create stories independent of the main plot line of the game (Donlan, 2014).

### 2.3 Adaptive Behaviour

As discussed previously, having AI which differ drastically from the player in terms of skill level results in games that are not fun to play. Therefore, it is important to have AI which can satisfy players from both sides of the spectrum. Moreover, dynamic difficulty AI in particular has been shown to increase the players enjoyment of games, specifically in a Real Time Strategy game (Mirna Paula Silva, 2015). There are some requirements of dynamic AI which must be met before they would be suitable for use in a game, however. In the case of machine learning dynamically scripted agents it is important that they are quick in their computations, effective throughout the whole game (even while learning), robust to the randomness that is inherent in most games and finally they must be efficient in their learning as games can be short and similar situations will seldom happen in a single game (Spronck, 2006). Besides machine learning there are other methods that can be used to scale the difficulty of an AI opponent to a more appropriate level for the human player. For example “High-Fitness Penalising” is a system which gives higher rewards to mediocre moves instead of on how well the AI system is doing (Spronck, et al., 2004). Games are reported as being more fun when the difference between wins and losses are minimal and draws are uncommon (Tan, et al., 2011). It was shown that this can be formalised into the equation:

$$W = L = \frac{n - D}{2}$$

Where W, L and D represent the number of wins, losses and draws respectively and n is the number of games played. The idea of Artificial Stupidity is also mentioned in Tan et al's work. This is an AI system which makes plausible mistakes intentionally to help make the game more entertaining (Lidén, 2004).

Lidén talks about AI in the context of an FPS however. These include:

- Have Horrible Aim: As the AI can play optimally it is important that it intentionally misses so the player has a chance.
- Move before firing: Give the player a chance to react to the AI actions.

Moreover, it is mentioned that an AI opponent should feed into the players emotions to enhance the gaming experience most notably in the "Pull back at the last minute" scenario. Though Lidén does discuss in the previously mentioned section that the AI should let the player win the game, this would contradict what Tan et al. say, as excessive wins would maximise their equation and would suggest an unfun game. However, in this project the Minefield AI was built to accomplish the goal of raising the tension in the game even if it does not play very well to see if artificial stupidity or low win loss ratio create a more enjoyable game.

## 2.4 Monte Carlo

Games can be modelled as trees where each node in the tree represents the game state and each connection represents the action taken to move between game states. To decide which move is better than some other move one method is to simply count how often that move results in a win for the player compared to the others. The move which results in a win most often must be the best. The issue is for some games, such as Chess or Go, the game tree can become quite large quite quickly. As Ultimate Tic Tac Toe (UTTT), the chosen game for this project, has an estimated  $1.8 \times 10^{78}$  game states (Bredyn McCombs, 2018) it would be infeasible to perform an exhaustive search of the game tree and "Brute Force" a win. Monte Carlo Tree Search (MCTS) aims to make decisions based on the estimated chances of winning the game based on that particular move. The way it does this is by building up the tree asymmetrically, focusing on more promising sub trees and ignoring less promising ones, and more importantly by doing this it

performs better than more classical algorithms in large trees (Hao Cui, 2015). The algorithm for MCTS follows four basic steps.

- **Selection:** Starting from the root node, recursively select the child node which looks most promising (in this project the UCT value will be used to calculate this) until a leaf node is reached.
- **Expansion:** If the leaf node is not a terminal state of the game, generate all child nodes from the leaf and select one of them. Child nodes are any valid move from the leaf node.
- **Simulation:** Complete a random playthrough of the game from the selected child node.
- **Back Propagation:** Use the result of the simulation to update the nodes on the path between the child node and the root.

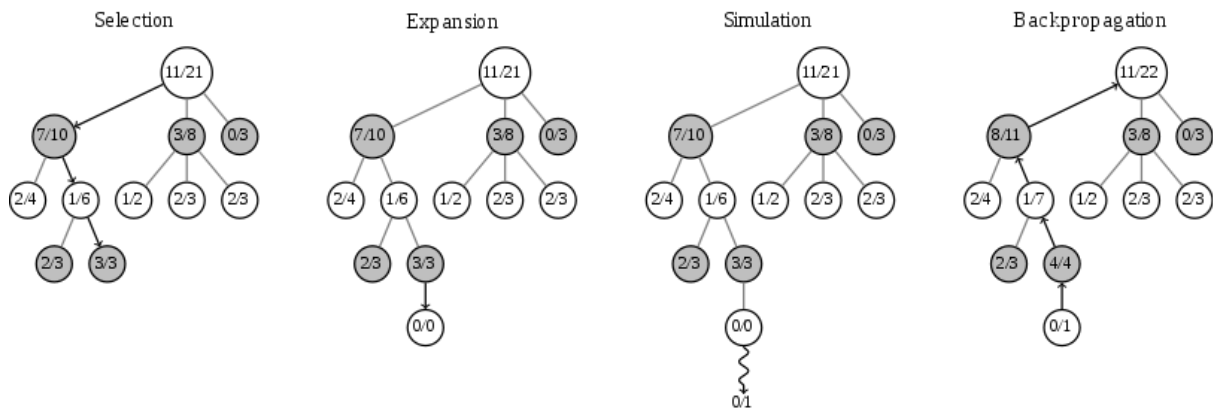


Figure 2-1 MCTS diagram taken from wikipedia (Wikipedia, 2017)

The UCT formula is a variation on the UCB1 (Upper Confidence Bound) that is used in bandit problems (Appendix I). The formula has been shown to have performance advantages over its competitors in some artificial game domains (Kocsis & Szepesvári, 2006).

$$UCT = \bar{x}_i + C \sqrt{\frac{\ln N}{n_i}}$$

Where  $\bar{x}_i$  is the reward for node  $i$ ,  $C$  is some constant, typically it is  $\sqrt{2}$  in a pure monte carlo implementation,  $N$  is the number of visits to the parent node of  $i$  and  $n_i$  is the number of visits to the node  $i$ . This formula balances out the exploitation of a node, the

first term, against the exploration of a node. This ensures that nodes are visited at least once and low scoring nodes still get visited, only less frequently.

### 3 Problem Description and Specification

---

Game playing has been one of the major areas of research since the inception of AI. The first example of game playing AI was developed by Eugene Grant, Herbert Koppel and Howard Bailer and it played Nim (Grant & Lardner, 1952). Since then game developers and computer scientists have built systems that play games with the intentions of creating unbeatable systems, not fun systems.

#### 3.1 The Problem

When playing games against an AI opponent there is one major issue the human player faces, the opponent does not match their skill level. This means that the AI can be too easy which results in a boring game, or too hard which results in frustration. AI which are too difficult are more common as computer scientists and game developers aim to make the most impressive AI. In most cases the game will allow the player to pick from some pre-determined difficulty setting, which is meant to provide a way for players to select which difficulty gives them the most enjoyment. However, these settings are difficult to scale and can end up with “artificial difficulty”, a system in which the game is not actually harder, the AI behaviour and overall experience is mostly the same. What changes is the amount of health enemies have, the amount of resources the player gets, the amount of damage the player can do etc. are changed (Suddaby, 2013).

#### 3.2 The Good Dad

For a game against an AI to be enjoyable it must avoid the pitfalls of artificial difficulty and adapt some sort of “real” difficulty to the player’s ability. The play style desired is analogous to the way a father might play a game with their child, challenging but rewarding and fun. The aim of this project is to develop an AI system that will play as described, by dynamically adjusting its difficulty to the perceived ability of the player. This will require that the developer of the Good dad system to be able to accurately quantify the ability of the player and use that data to adjust the AIs play style.

#### 3.3 The Game

The game chosen for this project has some requirements that it must satisfy. Those requirements are: it must be fully observable, all information about the game state is available to both players, it must be sequential, the current move will affect every



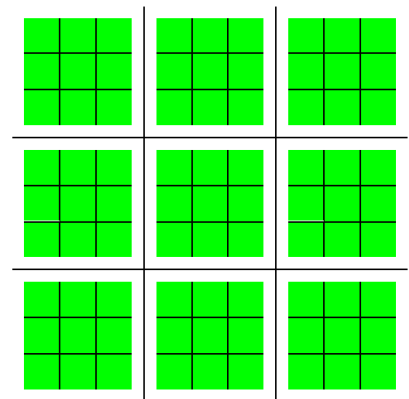
following move and lastly it must be static and the game state must not change while the player is thinking about their move. These requirements are in place to allow the human player to have access to the same information that the AI does.

### 3.4 Specification

In this section the selection of the game is discussed, as well as the approach for creating the AI.

#### 3.4.1 Ultimate Tic Tac Toe

The game chosen for this project was Ultimate Tic Tac Toe, a twist on the classic pen and paper game. The game is fairly new with the earliest post on it being dated 2013 (Orlin, 2013). The game features a regular tic tac toe board, except in each space where a tile should be there is another nested tic tac toe board. The choice of UTTT was made as the familiarity of tic tac toe would give players an idea of how it is played, but the added strategic complexity of the nested boards would make it much more interesting and novel.



*Figure 3-1 UTTT Board*

#### 3.4.2 Rules

- In the opening move player one can choose any tile to make a move on
- Players then take turns making moves on tiles
- Whichever position in a sub board a player makes a move in dictates which sub board the next player must make a move in
- A player wins a sub board when they place three of their tiles in a row in that sub board
- If a player is sent to a board that has been won by either player, they can make a move on any available tile on the board
- There are no available moves in a board that has been won or drawn

- A sub board is drawn if all tiles have had moves made on them
- When a player wins three sub boards in a row, that player is the winner and the game is over

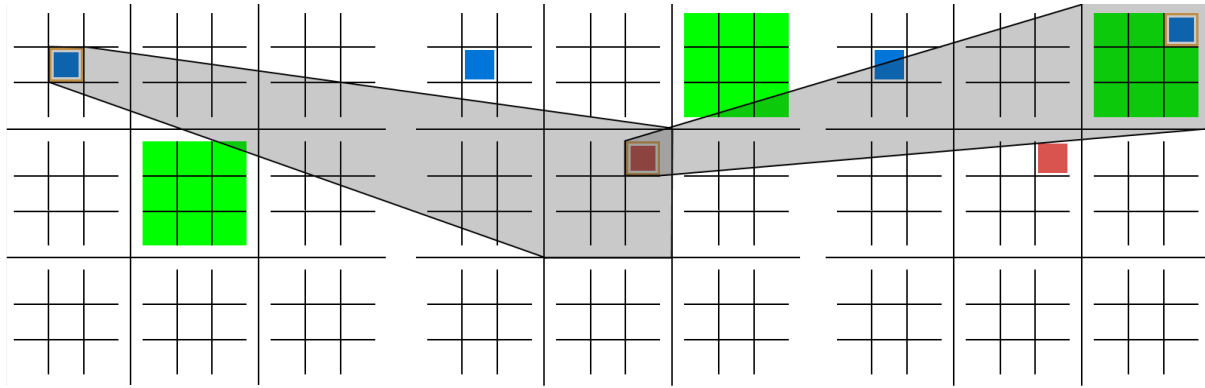


Figure 3-2 Progression of game adapted from (George & Janoski, 2016)

The positioning in the game adds a depth of complexity that users may not have experienced playing tic tac toe or any of its variations.

#### 3.4.3 The AI

To carry out experiments there will need to be a minimum of three AIs. This is necessary to create a spectrum of difficulty that a user can interact with. An easy AI, this will be an AI which selects its moves randomly, a hard AI, this will be an AI which will seek to maximise the reward gained from each move, and the Good dad AI, this will adapt the moves made to reflect the skill of the player. There will also be a fourth type of AI included in this project inspired by Artificial Stupidity (Lidén, 2004), the “Minefield” AI which will aims to play on the users’ emotions to produce an entertaining game.

#### 3.4.4 The Good Dad AI

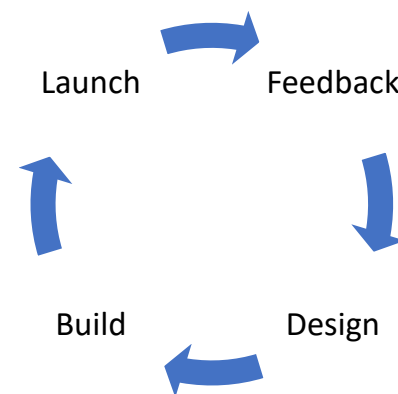
There were two different approaches used in designing the Good dad AI. The first and more simplistic was to build an AI which would select moves that were closest to the average reward that the opponent had. The second used a statistical analysis on the ranking of the opponents move to more accurately reflect the skill of the opponent.

## 4 System Design

The system was designed to allow many users to interact with the application while at the same time, while allowing future development work to take modify the behaviour of the application relatively easily. Test driven development was used throughout the creation of the application as it allows more robust and maintainable code to be developed.

### 4.1 Methodology

An Agile development cycle was employed for the implementation of the application. The Agile development cycle follows a four-step cycle in which feedback about previous development is fed into the next development cycle (see figure 4-1), so the product better reflects the wants and needs of the client. The cycle that was used in the development of this system is on the simpler side as the project does not have the industrial scope that is normally paired with this type of development. However, the advantage of the Agile development cycle is that, unlike Waterfall, it allows for requirements to be updated during the entire development process.



*Figure 4-1 Agile Development Cycle*

Frameworks are normally used in conjunction with Agile to help developers maintain the work load and make sure all tasks are completed. The tool that was used in this project is Azure Devops. Azure Devops maintained the work board, this contained all work completed and work to be completed, known as the backlogs, and allowed me to quickly and easily check what was next to be done. The use of these tools is not necessary in Agile but by having them a development team can better manage their weekly work load and better assign resources to the tasks that need it. The Build step could be further broken down into another cyclical development cycle, that being Test Drive Development (TDD). The TDD cycle is a simple one but is designed to make code flexible and robust. It is intended for paired programming, however, as the process creates a system with high test coverage and gives the developer much higher confidence in the code it was used for this project. Along with its other benefits, TDD allows code to be documented while it is developed. This project contains two hundred

and forty tests, each one describing a small piece of functionality and each piece of functionality working as intended as the product will not be published if tests do not pass

## 4.2 System Architecture

The system will be a web app containing an Angular front end and an Asp.Net Core back end. The choice of a web app was made as this would allow more people to access the application with ease, as well as have the user experiments take place in the application. The server follows a Layered Architecture with three layers; the controller layer, business layer and the data access layer. The layered architecture allows for each layer to be easily changed without disturbing any other layer. It also clearly separates different layers of abstraction such as, separating retrieving data from a database from interacting with the game logic. The system could easily be changed to support different types of databases for example it currently interacts with a SQL database but changes could be made to the data access layer to interact with a no-SQL database or some other form of storage. The first two layers also behave as the controller and the model in a Model-View-Controller (MVC) architecture. The View is the HTML front end again the MVC architecture allows for parts of the system to be easily changed as long as all concrete classes satisfy the interfaces in the system, this principle is known as “Programming to an Interface” (Eric Freeman, 2004). Lastly, the front end uses an Event Driven architecture to communicate between components. This was chosen as being a web application connected to a server through the internet responses from API calls would take an unspecified amount of time to return, as such events are used to signal these returns to any component that is interested in the result. Moreover, as the front end needed to interact with the player, eg clicking on the screen (an interaction which is easily modelled with events), an event driven architecture was a good solution to all of these problems.

## 4.3 Design Patterns

A project of this size must employ established design patterns that allow code to be flexible to change and easily extendable. As such this project features several different design patterns for behaviour and creation.

#### 4.4 Observer

The observer is a common design pattern normally used to communicate updates in a model, returns from asynchronous calls or user events. The pattern was used exclusively in the front end and allows components to maintain a loose coupling on each other while still communicating updates to some part of the system. The participants in a typical Observer pattern are:

- The Subject: The class which is being observed, it provides some mechanism for adding and removing observers
- The Observer: The interface which is called when an event is fired, typically it only has the “Update” method defined
- The Concrete Subject: The object which updates observers of change
- The Concrete Observer: Implements the Observer interface.

As Angular was used as the framework the implementation of the Observer is slightly changed in that classes do not implement an Observer interface, instead the subject class has an “Event Emitter” field which can be subscribed to. In effect this is analogous to adding the subscribed object to the subject objects’ observer list, which would be done in a typical observer pattern. On subscription an anonymous function is passed to the subjects’ event emitter which is called when an event is emitted. This implementation of the Observer pattern accomplishes the same goal as the more traditional pattern, but it does not require observer classes to implement an interface, while still maintaining loose coupling between the observer and the subject.

#### 4.5 Composite

The composite pattern is a design pattern that allows tree structures to be easily modelled and to allow clients to treat parts of the hierarchy uniformly. This pattern was used to model the game of UTTT, as it is a tic tac toe game made of up tic tac toe games this pattern lends itself well to the problem at hand. The pattern only has three participants normally, these are:

- The Component: Declares the interface for objects in the composite pattern

- The Leaf: Represents a leaf object in the tree structure (in the case of UT TT this would be a tile)
- The Composite: Stores child components and defines the behaviour for components which have children

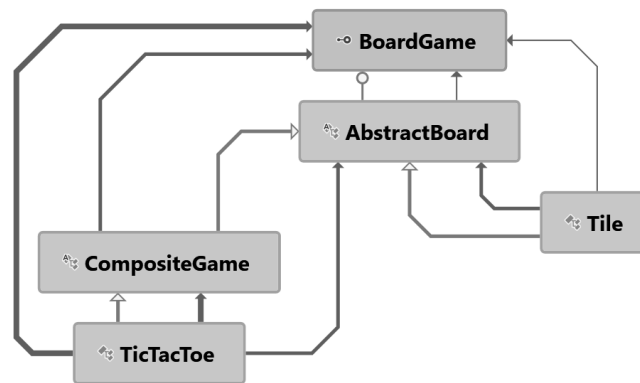


Figure 4-2 Composite Design pattern UML

In this example of the composite design pattern the Leaf class is the tile and the composite class is the TicTacToe class. The component that obscures this implementation from the client is the BoardGame interface which does not reference itself.

#### 4.6 Chain of Responsibility

This design pattern is flexible and was used for two different reasons in this project: for behavioural changes and as a creational pattern. Chain of Responsibility allows a

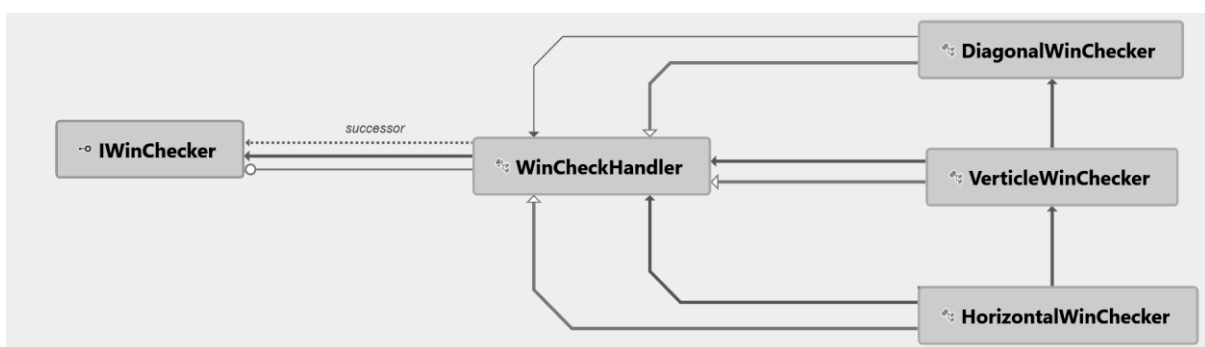


Figure 4-3 Chain of Responsibility UML

problem to be broken down into component parts and for each one to be linked independently. This allows code to avoid long strings of “if else” blocks which can quickly become cumbersome and unmaintainable. The pattern consists of three classes;

the chain interface, which defines a singular method used by the client to interact with the chain, the abstract chain class which defines an if statement and a mechanism for linking parts of the chain and the concrete chain class, this class defines the behaviour for part of the chain. This pattern was used to decouple the system that checked if a game was over from the game itself, while also allowing the win checking to be better tested. The chain components in that example were a VerticalWinChecker, HorizontalWinChecker and a DiagonalWinChecker, which checked for a win condition vertically, horizontally and diagonally respectively. This allowed the problem to be broken down into manageable pieces and avoid multiple if else blocks. As a creational pattern it is used when objects arrive to the controllers over the internet. As objects arrive as JSON and functions cannot be transferred objects must be created based on the properties of the JSON that is received. Different players have different properties so a Chain of Responsibility is used to decide which player should be created and to create it. This allowed me to easily add new links to the chain whenever a new player was created with little change to other code.

#### 4.7 Dependency Injection

The final pattern that was used in this project is dependency injection (DI). This is a creational pattern that allows the developer to define what is known as a Container, in which mappings between interfaces and concrete classes are established. Then when writing code, the constructor is simply given a reference to the interface and at run time the correct concrete class is provided. This allows the developer to create very flexible code as no class has a reference to any other classes' constructor. Moreover, no class has access to any concrete class instead only the interface which was defined in the container. However, this pattern does not allow for instances of a class to be swapped during runtime and so is not always applicable.

## 5 Design and Implementation

---

In this section I will cover in greater detail the design of the application, the models used, and the implementations created for this project. This section will go over the changes that the project went through, and some testing will be touched on.

### 5.1 Model

The models are the object representation of the game at any one point, it details where players have made moves, what moves are next available and which player is to move.

### 5.2 Requirements

The requirements of the model were:

- Store the state of the game
  - The position of moves on the board
  - The types of players in the game
  - The current Player
  - The sub board the player must make their next move in
  - The sub boards that have been won

It must expose methods to the controller that allow moves to be made and to check if the game state is terminal, whether that is a win or a draw. As the View is completely separated from the server and objects are passed in through JSON methods are not transferred. Because of this it is not possible to expose any methods to the view and all interactions with the model must be done through the API.

### 5.3 Design

The design for the games model was decided early on in development and fit quite well with the behaviour that was needed. The UTTT board would be modelled as a two-dimensional list of objects each of which would have another two-dimensional list of tiles. This model lends itself to the composite design pattern that was used to perform calculations on the board. Moreover, it also made drawing the board on the screen simpler as Angular allows HTML components to be dynamically allocated in the DOM, which means that simply by defining the logic to decide if a tic tac toe board or a tile



should be placed in one location would allow Angular to populate the DOM with all the necessary components. AI Players were not implemented on the client side, they ran on the server as to not have the performance of the clients machine impact their performance. To maintain the information of which players where playing a game the players were enumerated and the enumerated value was sent along with the board to the server where the correct player could make its' move. The classes involved in the model are the TicTacToe class and the Tile class, both of which implement the BoardGame interface but the TicTacToe class also extends the CompositeGame class which is where it derives its' composite behaviour. The BoardGame interface defines only a handful of methods, the most important of which are the makeMove and the getAvailableMoves. Both these methods take advantage of the composite nature of the TicTacToe class to allow code to be more concise and maintainable. getAvailableMoves in particular takes advantage of the composite design pattern, by treating nested components identically no concession needs to be made for leaf components, the TicTacToe board simply calls to get available moves from which ever sub boards are defined as available and returns that list.

#### 5.4 Implementation

The key methods in the model are the MakeMove and the TicTacToe class. MakeMove takes a Move

```
public override void makeMove (Move move)
{
    owner = move.owner;
}
```

Figure 5-1 Tile makeMove method

```
public override void makeMove (Move move)
{
    board[move.possition.X][move.possition.Y].makeMove (move.next);
    validateBoard();
    registerMove (move);
}
```

Figure 5-2 TicTacToe makeMove method

object, which is also a Composite class, and because of the composite nature of the TicTacToe class, it can simply treat all child classes identically pass the move along for processing. The validateBoard method is used to check the winner of the game as well as perform some simple checks after a move is made, registerMove allows the class to restrict what the next available sub board will be. The Tile class, which is the leaf of the

composite pattern, must also implement the makeMove method and this is what allows the TicTacToe class to treat all sub boards identically.

## 5.5 View

The view is a web app that displays and allows the user to interact with the model. It was made using Angular 5 framework which enables elements to be dynamically allocated to the DOM.

## 5.6 Requirements

The fundamental objective of the view is to provide the user with a clear and intuitive representation of the underlying data. The view does not provide the user with any functionality to manipulate the data however as this would break the single responsibility principle. The functionality the view does provide include:

- A Start screen with the functionality to:
  - Ability to selecting players in a game
  - Ability to select the number of games to play
- Login screen which provides functionality to
  - Ability to register as a new user
  - Ability to log in as an existing user
- Game screen with functionality to
  - Allow a player to play a move
  - Clearly display the moves available to the user
  - Display moves previously made
  - Allow the user to differentiate between the two players moves
  - Allow the user to see who one each sub board at a glance

## 5.7 Design

The view was implemented with Angular Framework. It makes use of the Model-View-ViewModel (MVVM) design pattern. This means that the representation of the model in the view does not necessarily need to completely reflect the model, only the parts which are relevant. It also allows for separation between the logic and the HTML that displays the model to the viewer. Angular allows components to be dynamically loaded to the DOM, this was beneficial as it allowed components to be nested without knowledge of how deep the UT TT game is nested. The “\*ngIf” decorator is part of the

```
<div *ngIf="space.board != null && space.board != undefined">
  <ultimateTictactoe [lastMove]="getLastMove(x, y)" [available]
</div>
<div *ngIf="space.board == null || space.board == undefined" >
  <tile (moveEvent)="moveMade($event, x, y)" [lastMove]="getLa
</div>
```

Figure 5-5 Example of dynamic DOM allocation

angular framework and is the mechanism by which angular dynamically selects components to render in the DOM. Both the “ultimateTictactoe” and the “tile” tags will be resolved to the HTML that is pointed to by the component with the selector decorator

```
@Component({
  selector: "ultimateTictactoe",
  templateUrl: "../ultimateTictactoe.component.html",
  styleUrls: ["../ultimateTictactoe.styles.css"]
})
```

Figure 5-4 Component decorator of UT TT class

that matches the tag. The variables in the square brackets denote what angular define as “Inputs”. These are values that are passed from a parent component to a child component through the DOM.

```
@Input() lastMove: Move;
@Output() moveEvent = new EventEmitter<Move>();
```

Figure 5-3 example input and out put

Additionally, the event in the round brackets is what is known as an “Output”. This is the angular implementation of the observable, the function that is made equivalent to the output will be called whenever the emit is called on the event emitter that is tied to this output. The variable “\$event” will be whatever variable is emitted. This is part of what makes the MVVM architecture powerful, as the logic that drives the UI can be completely refactored and remade without impacting the way the application looks, as long as the interfaces defined are respected.

## 5.8 Libraries

In the client a number of libraries were used to add additional functionality and save time in development. Besides dependencies the main libraries used in this project were; Bootstrap, angular2-toaster and rxjs.

### 5.8.1 Bootstrap

Bootstrap provides an extensive css library which allows developers to quickly and easily create web pages that look clean and professional. The basic version of bootstrap however does not integrate well with angular, as it does not provide a way to interact with certain components such as modals. For this a wrapper library was used, “ngx-bootstrap”, which provides components which can be injected into code and provide extra functionality to the UI. The BsModalService is provided by ngx-bootstrap and is

```
constructor(private gameService: AbstractGameService, private modalService: BsModalService, private userService: UserService, private toast: ToasterService) {}
```

Figure 5-7 ModalService being injected into class

a wrapper for the bootstrap native modal class. This allows the developer to have greater control over how the UI behaves. As seen in figure 5-6 the modal service allows bootstrap components to be interacted with in a concise and intuitive way. The show

```
playAgain(playAgain: boolean): void {  
  if (playAgain) {  
    this.overlayVisable = true;  
    this.gameService.createGame(3, this.gameService.getPlayers());  
  } else {  
    this.gameStarter = this.modalService.show(GameSetupComponent, {class: "modal-lg"});  
    this.gameStarter.content.opponentSelectedEvent.subscribe((players) => {  
      this.startGame(players);  
    });  
  }  
  this.gameOver.hide();  
}
```

Figure 5-6 Example of Service call

method simply displays the passed in component as a modal as well as applying the style defined by the user. Moreover, it allows developers to access to components encapsulated in the modal by way of the content variable.

### 5.8.2 Ngx-toaster

This library provides services to display toasts. The library was selected over the ngx-bootstrap solution as there were version conflicts with the installed version of bootstrap and an attempt to resolve the conflicts proved to be cumbersome and time consuming. This library does not provide as clean a solution as ngx-bootstrap does as every

component that wishes to display a toast must have a “toast-container” tag in its HTML. This in itself is not much of an issue in ways of separation of concerns, where it does become a problem is when errors occur in components that do not have views, such as services. In this case the component that uses that service must recognise the services use of toasts and provide a toast-container for it. This results in high coupling between components in the view.

## 5.9 Modals

Modals are a small component that pop up over the application, these are useful for giving the user a small piece of information or let the user input such as a confirmation box.

## 5.10 Toast

Toasts are small popups that contain a single piece of information. These do not interrupt the use of the application. These toasts can be different colours to communicate extra information to the user. The major difference between this and a modal is that toasts do not have any

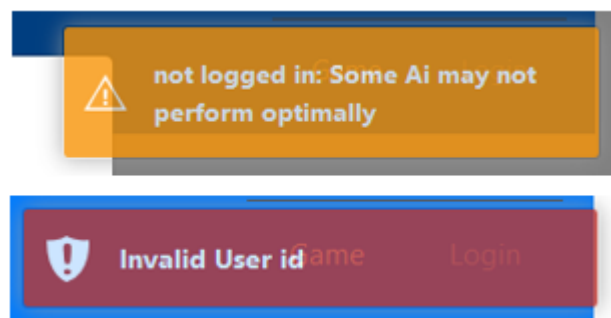


Figure 5-8 Example of toasts

interactions with the user. They are used strictly to inform the user of some vital piece of information. The library used for toasts is “angular2-toaster”. This library encapsulates bootstraps native toast classes which allows them to be better integrated with angular, by allowing a toast service to be injected into components that will need to use it.

## 5.11 Login Screen Design

The loading screen is fairly simple in its design as it does not provide lot of functionality and should very easily afford logging in. It is a simple input with a login button appended and another button which allows users to create new accounts.

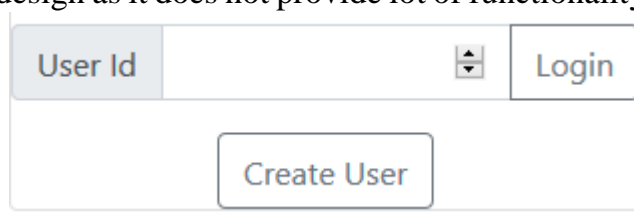


Figure 5-9 Login screen

### 5.12 Game View Design

It was intended for the screen to be split into three sections when a game was in play. The outer most sections would be dedicated to each of the players, displaying information on what colour each player was, their name and other stats. Due to time constraints this design of the UI was never realised, and both end components were not implemented.

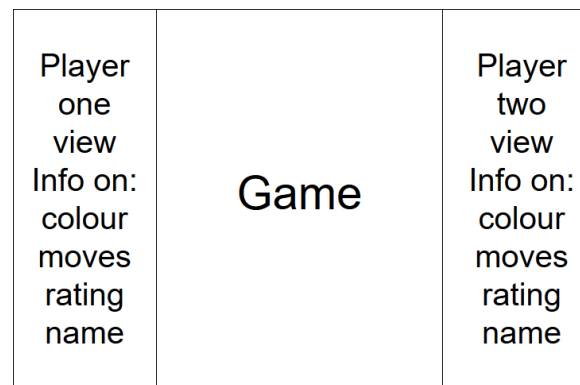


Figure 5-10 Wireframe of game UI

### 5.13 Navigation

The navigation uses a simple nav bar places across the top of the screen. There are two pages contained in the app, the login page and the game page. It was intended for the login functionality to be encapsulated in a modal, however it was decided that a better separation of functions between this and other parts of the application made navigation easier.



Figure 5-11 Navigation bar

### 5.14 Controllers

The controllers are split across the server and the client. They provide the user with a mechanism by which they can interact with and mutate the model.

#### 5.14.1 Client

On the client side the controllers provide a way for the user to interact with the model through the view.

#### 5.14.1.1 Requirements

The requirement on the client side are simple and surround the user's ability to do two things, log in and play the game. For logging in the controller must:

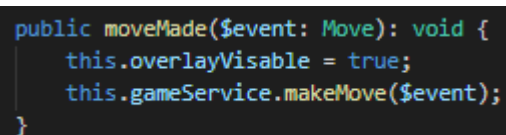
- Communicate to the server the users actions
  - Login
    - Create new user
    - Login as existing user
  - Game play
    - Communicate with server which players to construct
    - Communicate with server which move to make

#### 5.14.1.2 Design

The client side has only one controller. It is implemented as an Angular component because although it does not provide any view of its own it wraps the UTTT view and so must have HTML associated with it. It uses a system of modals and events to communicate with the view and underlying services. Moreover, it does not in any way mutate the model it simply provides the mechanism for the user to communicate the actions they would like to take to the server, which in turn mutates the model.

#### 5.14.1.3 Implementation

The client-side controller consists of only two classes, the “GameComponent” class and the “UserComponent” class. The GameComponent class interacts with the UTTT Component, which provides the view for the game, and responds to click events that propagate out from the view. Observers for click events are registered in the HTML, when a click is registered on a tile it emits a move event up to the parent UTTT component. The UTTT component will then recursively emit move events to their parents until it reaches the GameComponent where it is then processed. Processing of moves simply involves sending the relevant information to the GameService, which will



```
public moveMade($event: Move): void {  
    this.overlayVisable = true;  
    this.gameService.makeMove($event);  
}
```

Figure 5-12 Example of calling GameService

then send it to the server in the correct format. Also, the GameComponent is responsible

for communicating which players the user wishes to play. Again, to do this the component simply gets the relevant information from the view and passes it to the appropriate service to send it to the server. This separation of concerns between the controller and the server is vital for several reasons, firstly one the endpoints on the server may change, secondly the server may be replaced with some different system, possibly client side. These changes would be cumbersome if the GameComponent were to directly interface with the server. The other controller on the client side is the UserComponent, it is responsible for communicating login details and requests to the UserService, it does this in the same way that the GameComponent does.

#### 5.14.1.4 Maintenance

In the first implementation of the game players were only given one option when setting

```
private startGame(players: Array<Player>): void {  
    this.noGames = this.gameStarter.content.getNoGames();  
    this.gameService.createGame(3, players);  
    this.gameStarter.hide();  
}
```

up a game, this was who the

*Figure 5-13 Example of extended code*

opponent was going to be. As development progressed new needs arose, those being, two AI should be able to play each other, the human may not always want to play first, for experiments multiple games between the same opponents should be queued up. To accommodate these changes the GameSetupComponent, a component displayed by the GameComponent when the game window is first visited, was extended to allow players to select both players and a new input field was added to allow some number of games to be queued. That is why, in the above figure players is passed as an argument, as an array of players was always constructed by the GameSetupComponent, but the number of games is retrieved in the method call.

### 5.15 Server

In the server controllers behave slightly differently, they are responsible for retrieving web requests and calling services to do work on the information they receive.

#### 5.15.1 Requirements

The requirements for a server-side controller are minimal. They must:

- Expose an endpoint of the server
- Pass the appropriate information form the web to the service



- Return the request in the appropriate format

#### 5.15.2 Design

To abstract some of the basic web interaction from more specialised controllers, all controllers extend a class called BaseController which in turn extends Controller. The Controller class is provided by the Asp.Net framework which provides methods for receiving and returning HTTP requests. There are only two classes which extend the BaseController these are, The GameController and the UserController. Both of these controllers expose various endpoints which allow a client to request actions be made to a model or some value be updated in a database. The Asp.Net framework also provides some decorators which are used to expose methods to the web. Furthermore, there are decorators which tell the framework that certain arguments to a method will be provided in the HTTP request.

```
[HttpPost("rateMove")]
public IActionResult rateMove([FromBody] MoveDto moveDto)
```

*Figure 5-14 Example of end point*

#### 5.15.3 Implementation

The BaseController has only one method, ExecuteApiAction, this method must be called by any controller that wishes to respond to a http request as it is responsible for

```
protected IActionResult ExecuteApiAction<T>(Func<ApiResult<T>> function, string uri = "")
{
```

*Figure 5-15 BaseControllers ExecuteApiAction method signature*

returning the correct HTTP status code. The ExecuteApiAction method takes an anonymous function as an argument, this function will be called by the BaseController, if no error occur the result of the call will be wrapped in an IActionResult object, which is provided by Asp.Net core. If an error does occur, then the BaseController handles the exception and returns the correct HTTP error code. The decision of an anonymous function was made as it helps the BaseController to be more generic. The ExecuteApiAction method can return an IActionResult of any type. It allows all inheriting classes to define any methods they wish as long as they return an ApiResult.

#### 5.15.4 Maintenance

When a request arrives from the web it is in JSON. In the initial implementation of the GameController the object passed into the rateMove method had a variable called game, which is a representation of the board, and was typed as the interface BoardGame. This

became an issue when the client was able to send requests to the server, as the interface is implemented by multiple classes so the JSON deserialiser provided by Asp.Net could not decide which class to use and through an exception. In an attempt to correct this issue, the JSON deserialiser was sub-classed to add functionality that would determine the correct subclass to return. This also proved problematic as the new deserialisers could not have classes injected into them which were needed by the concrete classes of BoardGame. Another solution was implemented which saw the game variable's type being changed from a BoardGame to a JsonObject and a new class was implemented to properly construct the correct BoardGame from the JSON. This solution, though requiring more code, allowed the client to remain unchanged while also allowing classes to be injected into the new BoardGameConstructor class which allowed loose coupling between any service needed by a BoardGame and the constructor of the BoardGame.

## 5.16 Services

Services exist on both the client and server side and both intend to make changes to the model. As discussed previously in this chapter client-side services do this by sending

```
BoardGame game = boardCreationService.createBoardGame(moveDto.game);
```

*Figure 5-16 Example of server-side service call*

requests to the server, the server-side services have access to the actual implementation of the model though and so can call methods to mutate the model.

### 5.16.1 Client

On the client-side services exist through the duration of the tab that the app is open in, unlike other components. These services serve as the true version of the model that is passed from the server and all other components use a system of events to communicate with them

#### 5.16.1.1 Requirements

The requirements of clients side services are simple, they must:

- Maintain the model
- Provide methods to mutate the model
- Provide events for when the model is updated

The difference between this and a controller is this does not interact with any event from the view, it does not interpret click events or menu choices. Instead it provides the controllers with a way to communicate with the server, which in turn will mutate the model.

#### 5.16.1.2 Design

To decouple the services of the client from the web a lower level of service was created called the ApiService. This service wraps the HttpClient that is provided by Angular.

```
constructor(private http: HttpClient) {}

get<T>(endPoint: string): Observable<T> {
    return this.http.get<T>(this.getURL() + endPoint);
}

post<T>(endPoint: string, data: any): Observable<T> {
    return this.http.post<T>(this.getURL() + endPoint, data, this.getOptions());
}
```

Figure 5-17 Wrapping of HttpClient object

Doing this isolates any change that Angular might implement on the HttpClient so only this class would need to be updated. Moreover, it allows reusability of this code as multiple services make HTTP requests. The classes which use this service are the GameService and the UserService, both of which have the ApiService injected into them.

#### 5.16.1.3 Implementation

The services on the client-side consist of two classes, the GameService and the UserService. As services in Angular exist for the duration of the application and are singletons by design they hold information that be requested for by controllers. The GameService holds all the information that the controllers would need to process user actions correctly, such as the state of the board, the available moves and the current player. The

```
curPlayer: Player;
players: Array<Player>;
board: Array<Array<BoardGame>>;
availableMoves: Array<Move>;
```

Figure 5-18 Game Services public fields

UserService is mostly the same except it stores information regarding the current logged in user. These classes are not meant to be overly large, as they are not meant to have much functionality in them. The UserService for example only has thirty-nine lines of code, as it is mainly designed to forward information to the server's controllers.

#### 5.16.2 Server

The services on the server are where the vast majority of the games logic is processed. They exist in the business layer of the servers layered model along with the game models.

##### 5.16.2.1 Requirements

The services are the where most of the code in this project exists. They provide the functionality that interacts with the model and the logic that formats the HTTP responses. These classes have the biggest requirement set of any in the project. They are:

- Let both players make moves
- Decide who the winner is
- Have the moves made by both AI and human players rated
- Have a mechanism for persisting the move data
- Flatten the response time for AI players
- Create the DTO that will be sent to the client
- Provide a mechanism for creating new users
- Provide a mechanism for retrieving existing users
- Provide a mechanism for updating existing users
- Provide a service for creating random numbers
- Provide a system for ranking moves
- Provide a mechanism for creating Model objects from the JSON received from the web

##### 5.16.2.2 Design

The services all exist in a simple inheritance with only an interface and the concrete class. This is because Asp.Net frameworks dependency injection requires injected classes implement an interface and that interface is injected. The concrete class will be

created by the runtime. Doing this decouples the implementation of the service from every class that uses it, as nowhere in the code that was developed is a new instance of a service created. This allows future developers to create new services which fulfil the promise of the interface. They can then change the mapping in the Asp.Net container which will then supply the new class. As mentioned previously the services exist in the business layer of the server, this means that they are used by the Api layer to mutate the models passed in from the web, while also having access the data access layer. Because of this these services are the largest classes in this project. There are multiple services in this project which exist only because TDD demands they do. Services such as the RandomService which wraps the Random class. This class is well tested, which would be substantially harder to do if every class which required random numbers simply had an instance of the random class.

#### *5.16.2.3 Implementation*

The main service class in the server are the GameService and the UserService, mirroring the client-side services. These services though are far more substantial as they provide all the mechanisms needed to process the requests that come from the client. They are injected into the Controllers and are called to process all the requests that arrive from the web. The GameService interface only exposes two methods, processMove and rateMove. These methods are used by the controller to have the service process or rate a move that arrived from the web. The UserService is mostly the same providing methods for creating, updating and getting users.

#### *5.16.2.4 Maintenance*

In the original implementation of the GameService the AI player was given the board which it then made its move on and returned. This system was found to be an issue for two reasons; AI players developed in the future could simply cheat and mutate the board however they wished as it was passed by reference, AI response times depended on how much processing they did before making a move, the RandomAI responded very quickly while the MCTS AI responded after a couple seconds and lastly the amount of time taken for an AI move was effectively doubled as to rate a move the MCTS algorithm was used, so AI which already performed this action would look to the user to take twice as long than AI which did not. To rectify this the MCTS algorithm was

called before the AI were. They were then passed the board they were given and the tree built by the MCTS. This flattened the response time as the amount of time AI took to process the nodes they were given is minimal, while also only needing the MCTS algorithm to be ran once on the game. Furthermore, it meant that when rating AI moves the scores that the AI saw are identical to whatever will be stored. This means that the representation of the AI in the database accurately reflects how the AI intends to play.

### 5.17 Providers

Providers are class which exist exclusively in the server as their primary objective it so provide the application with access to a datastore. The provider lives in the data access layer of the server and abstracts the type of data store used by the system.

#### 5.17.1 Design

The provider consists of only one class and one interface. This is because the data access layer, though vitally important to the operation of the application is not overly complex. The interface is used to abstract the implementation away from the business layer, so at any time in the future a new

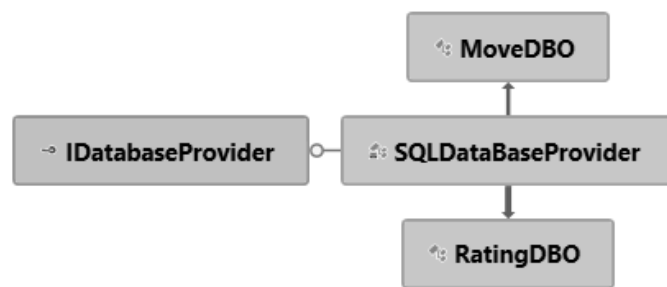


Figure 5-19 Data access layer UML

provider can be implemented which accesses some other form of data storage (No-SQL or Blob). As well as the provider there are a couple classes which serve to encapsulate related data these are known as DataBase Objects (DBOs). The DBOs allow the provider to pass data to the upper layers of the server without exposing the implementation of the storage.

### 5.17.2 Implementation

The `SQLDataBaseProvider` is the sole class that interacts with the database. It does this by instantiating connections with the azure database and performing queries on the desired tables to get the data requested for by the upper layers. The `IDatabaseProvider` does expose the Move and Rating DBOs it does this to allow the implementation to hide the rows it receives from the database while still communicating the data received.

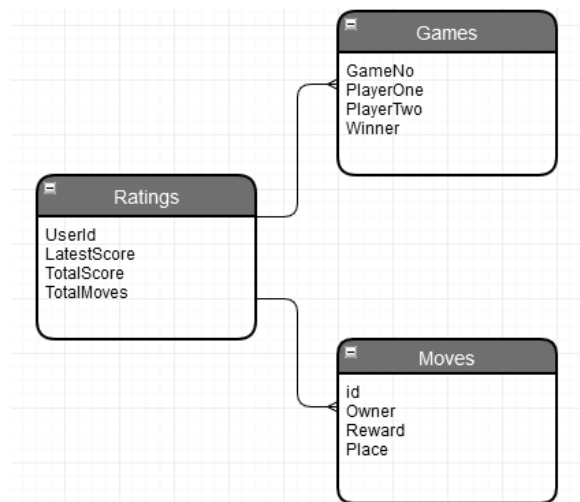


Figure 5-20 Database diagram

As a security feature the connection string required by the database required by the database to query tables is not stored in the implementation. Instead the string is stored in a JSON file which is read during the run time to get the connection string. The advantage of this is that the string can be update when the app is pushed to production, so the connection string is never exposed on the git server or locally. The DBOs are data objects, which is normally a code smell, they were designed like this to avoid multiple calls to the database, which is a slow operation. As data objects the DBOs do not have any methods defined, only public fields. The database is made up of three tables; the ratings table which holds the UserIds for all users and some rating information, the Games table, which holds information on games played, and the moves table, which holds a record of moves made by users.

## 6 Validation and Verification

---

To verify and validate the server and the client similar methods and techniques were used however the tools differed. The Microsoft testing library and Jasmine were both used extensively to develop the two hundred and forty tests which were vital in ensuring the correctness of the product while also allowing for quick bug identification and correction. There are one hundred and eighty tests on the server as it encapsulates most of the functionality and sixty on the client. UI testing is possible, with tools like Selenium, however, this was deemed as too time consuming as tweaks to the UI would cause tests to fail resulting in failed builds. Test cases were not created for this project, this was done as there is no single test that encapsulates enough functionality that a use case is applicable.

### 6.1 Testing Strategy

*“Perfection is Achieved Not When There Is Nothing More to Add, But When There Is Nothing Left to Take Away” Antoine de Saint-Exupery*

In this project Test Driven Development (TDD) was heavily leveraged to ensure a well-tested and maintainable product was produced. The TDD cycle follows a four-step process that allows the developer to be confident that the code does as it is described.

- Write a failing test
- Write code to pass the test
- Run all tests
- Refactor

The reason a test should be written before the code is fairly simple, a test that has never failed cannot be trusted. After code has been written to satisfy the test all tests in the project are run. This is to ensure that this piece of functionality did not affect the behaviour of any other component. Lastly refactoring is simply to allow the code to make sense. It also helps reinforce some core coding principles that were believed be of great importance, for example, it requires only code that satisfies tests to be written and so enforces the “You aren’t gonna need it” principle. As development progressed however, and time became less abundant, the TDD structure as not followed as closely,



as a result code coverage sits at 71% in the server, mostly due to untested code in couple of the AI classes. Excluding the AI brings coverage up to a respectable 89%. Moreover,

Symbol	Coverage (%) ▲	Uncovered/Total Stmts.
▲ [icon] Total	71%	354/1228
▲ [icon] UltimateTicTacToe	71%	354/1228
▶ [icon] UltimateTicTacToe.Models.Game.Players	19%	257/317
▶ [icon] UltimateTicTacToe.Models.MCTS	74%	27/103
▶ [icon] Models	80%	1/5
▶ [icon] UltimateTicTacToe.Controllers	81%	8/43
▶ [icon] UltimateTicTacToe.Models.Game	90%	18/183
▶ [icon] UltimateTicTacToe.Services	92%	31/387
▶ [icon] UltimateTicTacToe.Models.Game.WinCheck	94%	12/190

Figure 6-1 server-side test coverage

in this project tests where seen as documentation. This is preferred over traditional documentation as the tests are infallible, if designed well. The test must pass before the code will be committed to master and so they can be seen as the true behaviour of classes in the project. However, as also demonstrated in this project parts of the system are untested, and so lack documentation. There was every intention to test the code that had been written hastily but as it was complete and there was more work to do these tests were never written. Tests can still be written after the class has been otherwise completed, in instances of a bug where all tests are correct, but

```
it("Will not throw an exception if lastmove is null", () => {
    try {
        comp.getLastMove(0, 0);
    } catch {
        fail();
    }
});
```

Figure 6-2 Example of test written to catch a bug

some piece of functionality still does not behave as expected a test may be written with the input that caused the error. This will allow the developer to see where the code fails and correct it more easily.

## 6.2 Jasmine

Jasmine provides two roles. It is a testing framework that allows javascript and typescript tests to be run in the browser, and it is a mocking framework. The structure of tests in the client is simple, every component exists in its own folder along with a test class (spec.ts). This allows future developers to quickly and easily find components, along with all their

tests and check if everything works as described. Jasmine dictates a “It will” style in naming tests as the method used in the test is simply called “It” which then takes a string argument as the name of the test and an anonymous function which is the test. As

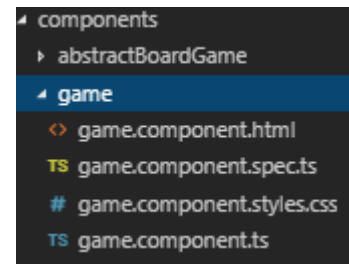


Figure 6-5 Angular file structure

```
it("Will set the over lay when a move is made", () => {  
    comp.moveMade(new Move());  
    expect(comp.overlayVisable).toBe(true);  
});
```

Figure 6-4 Jasmine test

demonstrated below the tests are intended to be concise. They describe exactly one piece of functionality each, this is demonstrated by the fact that each test has only one “expect” call. This is important as it ensures that if a test ever fails future developers will know exactly which piece of functionality has not behaved as expected. As a mocking framework jasmine does two things, it can create mock objects and it can mock

```
it("Will call get on the Http client", () => {  
    const mockHttp = jasmine.createSpyObj("HttpClient", ["get"]);  
    service = new ApiService(mockHttp);  
    spyOn(service, "getURL").and.returnValue("mockUrl");  
    service.get("/some endpoint");  
    expect(mockHttp.get).toHaveBeenCalled("mockUrl/some endpoint");  
});
```

Figure 6-3 Example of mocking in Jasmine

out calls to methods in the class under test.

## 6.3 Mocking

Mocking is a well-established testing practice that allows developers to isolate the code they are testing by “mocking” other pieces of functionality that is not under test. In this project two mocking libraries were used, jasmine for the client and Moq for the server. Both provide functionality that allows the developers to isolate testing code as well as test that interfaces were called with the correct arguments, the correct number of times etc. Mocking is a powerful tool when used in combination with DI as it abstracts away

how components are created. Using a well-tested DI framework, like the ones used in this project, mocking can allow for classes to be completely isolated from their dependencies so unit tests can focus on one discrete piece of functionality in the class. Moreover, mocking influenced how some of the project was designed, for example, the MCTSService has a NodeConstructorService injected into it, this was done to allow the creation of nodes to be thoroughly tested, which would have been more difficult had that functionality been hidden inside the MCTSService.

#### 6.4 Microsoft Unit Testing

On the server Microsoft Unit Testing was used along with Moq. Moq is a mocking library that provides functionality that allows mocking of other interfaces, unlike jasmine, it cannot mock methods in the class under test. However, this is not a major issue as only public methods should be tested as they are the only methods other classes should be interacting with. Mocking works in much the same way as in Jasmine, however, this does mean that some design decisions were made to accommodate this. As discussed previously an example of this can be seen in the NodeCreationService used by the MCTSService. Looking at the tests it is easy to see why this decision was made, even if the code to construct a node is not complex it would have been much

```
[TestMethod]
public void WillGetNodeCreationServiceToCreateRootNode()
{
    Mock<BoardGame> mockGame = new Mock<BoardGame>();
    Mock<INode> mockNode = new Mock<INode>();
    mockNode.Setup(expression: x => x.isLeaf()).Returns(value: true);
    mockNode.Setup(expression: x => x.getVisits()).Returns(value: 0);
    Mock<INodeCreationService> mockService = new Mock<INodeCreationService>();
    mockService.Setup(expression: x => x.createNode(mockGame.Object, It.IsAny<PlayerColour>()))
        .Returns(mockNode.Object)
        .Verifiable();
    service = new MonteCarloService(mockService.Object);
    service.process(new Mock<BoardGame>().Object, colour: 0);
    mockService.Verify();
}
```

Figure 6-6 Example of mocking in MOQ

harder to test had it have been in the MCTSService. In the figure below it can be seen that the test demands only that the correct game (mockGame.Object) is passed to the node creation service, there will be another test for the player colour.

## 6.5 AI testing

There were two main questions to answer around the performance of the AI. How well did they adapt and were the adaptive AI more or less fun than the non-adaptive. For the former the testing method included running fifty games against one of the adaptive AI and any other AI. Originally there was only one adaptive AI, the Good Aad, which simply made moves whose reward was close to the average reward of the opponent. However, after the first round of testing it was decided that a second implementation of an adaptive AI would be implemented. This implementation would use the average placement of the opponent and take a move within two standard deviations of that then randomly choose a move in that range. Moreover, some checks were made to both implementations, such as choosing the winning move if it was an option. This change was made as during the game an average move is not the winning move, so the adaptive AI would delay the game until the opponent won. These rounds of games were also used to quantify how fun the adaptive AI was using the methods proposed by Tan, C. H., Tan, K. C. & Tay, A. The AI were also tested with human players. This testing consisted of having the player play some non adaptive then adaptive AI and record how much fun they had. They were also asked how hard they felt the AI player, this was done to confirm the work done by Chanel, G. & R. C. & B. M. & P. T. in that overly hard or easy AI should be less fun. It was intended for an external AI to be used in the project as a better benchmark of how well the AI performed. The AI chosen for this was implemented by the OFekFoundation (TheOFekFoundation, n.d.). Due to integration issues the AI implemented in this project did not play the OFekFoundation AI. This means that the AI in this project do not have any measure of how well they play the game.

## 7 Results and Evaluation

The intention of this project was to develop an adaptive AI system which was primarily fun to play against. To evaluate the success of the project in this regard users were asked to evaluate the AI in a single game. In this section the results of a six hundred and fifty games will be reviewed and discussed to assess the success of the adaptive AI in creating a fun game as well as some observations made about the game in general.

### 7.1 The AI

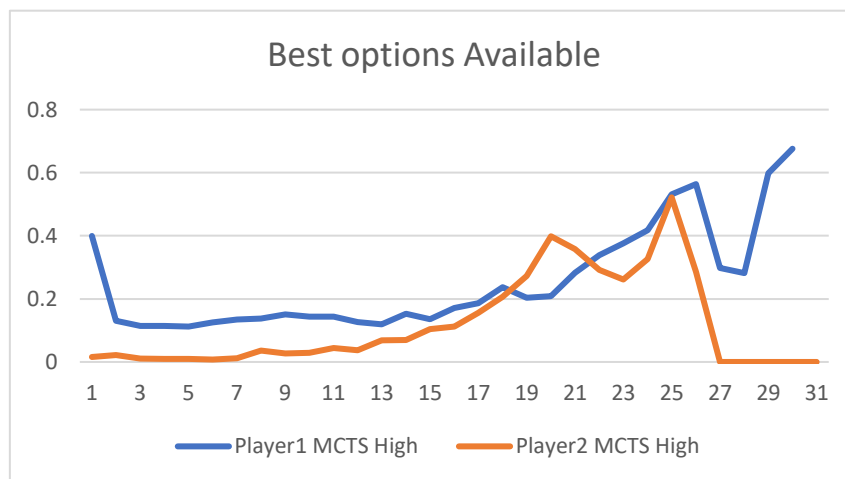


Figure 7-1 Average optimal reward for player one and two

As discussed previously, as there are no external rulers used to measure the performance of the AI developed. Instead the highest and lowest score achievable were logged for fifty games to produce an average progression of an average game. As seen in the figure above the first move seems to have a massive advantage, however, the implementation of MCTS only took three thousand samples of the game space, so the accuracy of this

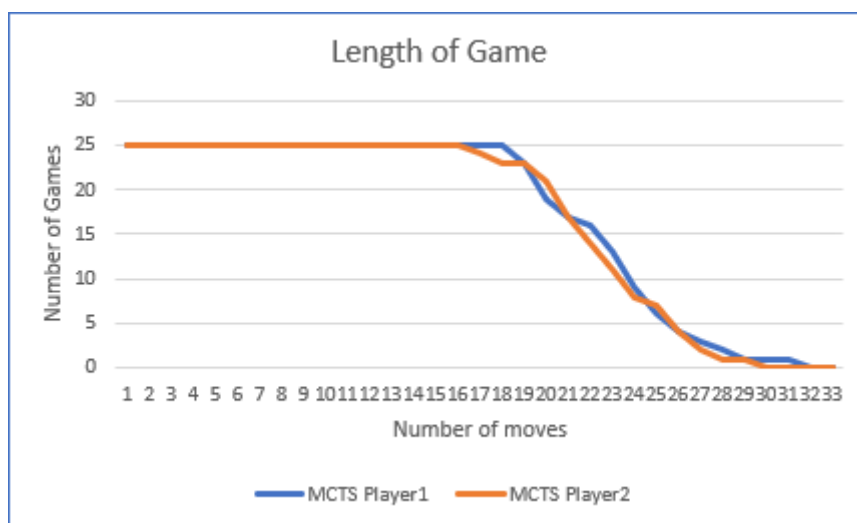


Figure 7-2 Average length of game for player one and two

measurement could be insufficient. The opening move has eighty-one options so if the tree is built symmetrically each child of the root is only visited thirty-seven times. Obviously, MCTS will not build the tree like this but, it may still not gain an accurate picture of the game state as there are so many options. The longest game played when playing optimally was thirty-two moves and the shortest was sixteen moves. In the figure below the odds of winning or losing in the average game converge before becoming farther apart in the later parts of the game. This observation lead to the design decision to weight the average move against the latest move. The original weighting was set at 60% to 40% but this was changed as more experiments were conducted. The

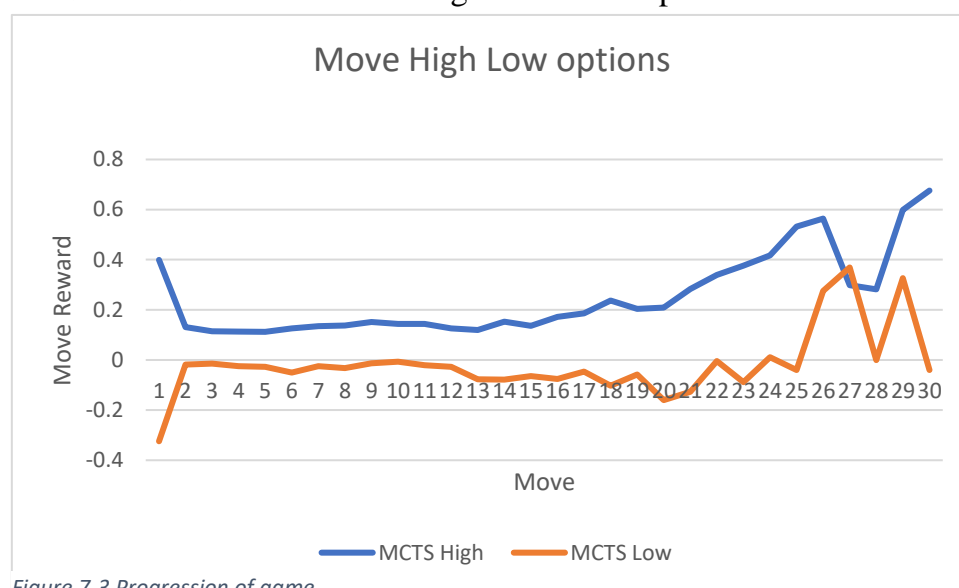


Figure 7-3 Progression of game

initial Good Dad performed poorly against the MCTS player, even though the Good Dad adapted the MCTS play style well, in a set of fifty games it only won twice. The second set of games had the adaptive AI weight average to latest moves at 80% to 20%, this proved to be ineffective at achieving the desired behaviour. The resulting AI did not win a single game against the MCTS player, drawing seven times and losing the rest. The fact that the average move made is not a winning move, or indeed a very good move, forced another change to be made to the adaptive AI. The next iteration had a small piece of logic added that dictated that if a winning move was available. Moreover, the adaptive was not behaving as intended. The MCTS player always picked the move that maximised its reward, but the adaptive AI would every so often choose a move

which was not optimal. This is simply because that move would have been exceptionally good and out with the norm, the MCTS player would have taken advantage of the situation, however. To combat this, two changes were made. Firstly, a small piece of code was added to the adaptive AI that would force it to make a winning move if ever given the opportunity, this proved to also be ineffective as although it did

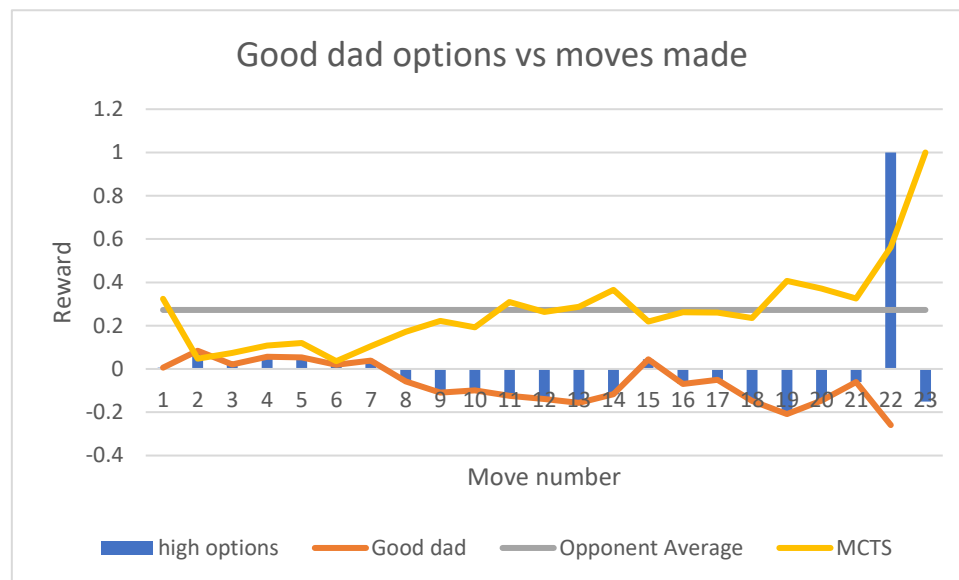


Figure 7-4 Graph showing Good dad not choosing optimal move

increase the win rate, it did not minimise the difference with the new Good Dad only winning seven games out of fifty. Secondly, a whole new adaptive AI was implemented, this one took the average place of the moves the opponent made, in the case of the MCTS player this was always first, and randomly selected a move with a placement within two standard deviations of the average this AI was named Good dad v2. This resulted in an adaptive AI which consistently played the best move when

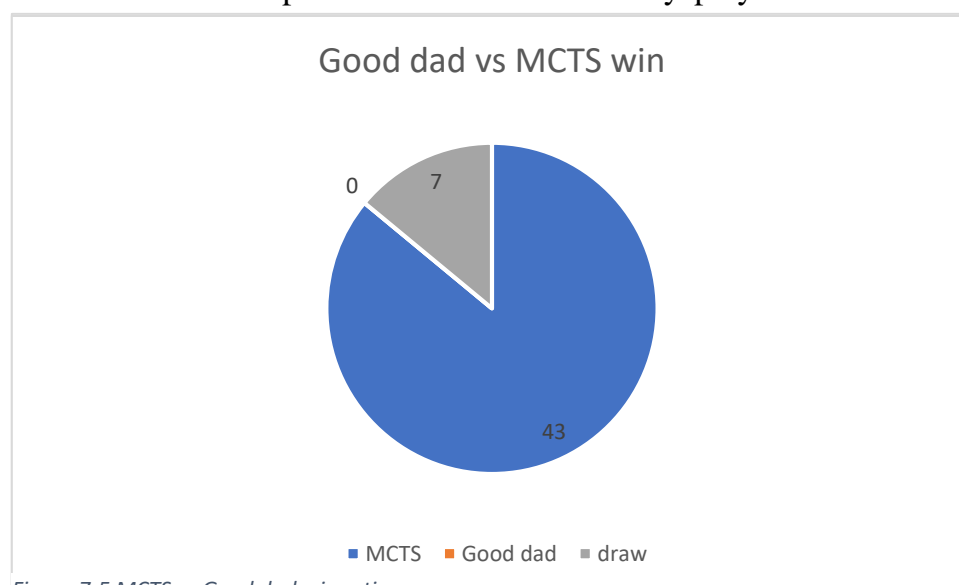


Figure 7-5 MCTS vs Good dad win ratio

playing the MCTS player. The initial implementation of the adaptive AI still did not perform at a reasonable standard as it was routinely beat by almost every other AI implemented in the system. The only AI which the 80-20 balanced AI performed well against was the RandomAI which had no decision-making process and so did not pick winning moves. Even though this system may have been able to beat the RandomAi at a consistent rate it still did not produce a fun game as defined by Tan, C. H., Tan, K. C.

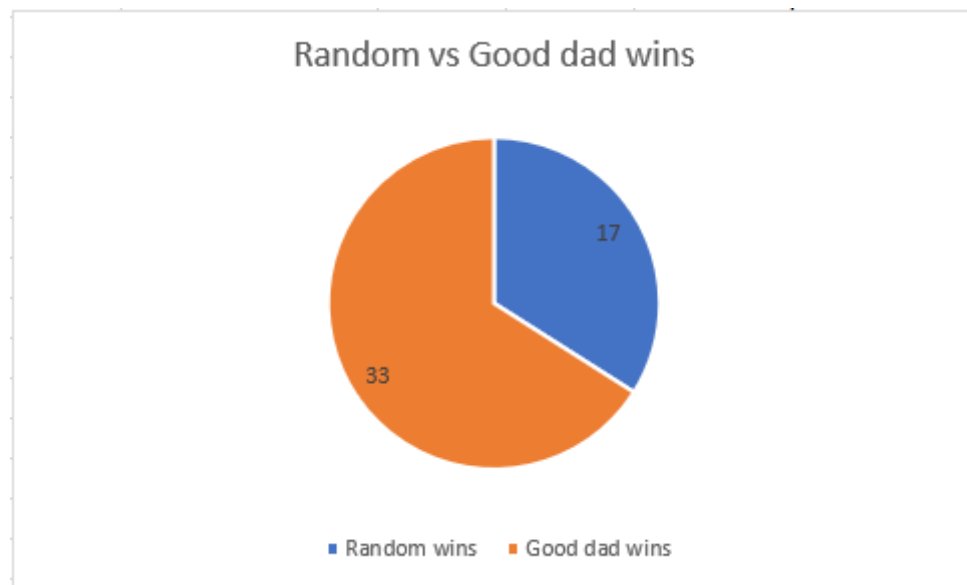


Figure 7-6 Random vs Good dad win ratio

& Tay, A. as the difference of wins and losses for the random player are quite high at sixteen, which is just one away from the number of wins the RandomAI had in total. The Good dad v2 performed at a much higher standard and adapted its' behaviour to match the MCTS and RandomAIs much more readily. Though it still did beat the RandomAI at a dispraportiontly high rate, it encountered more draws which is to be

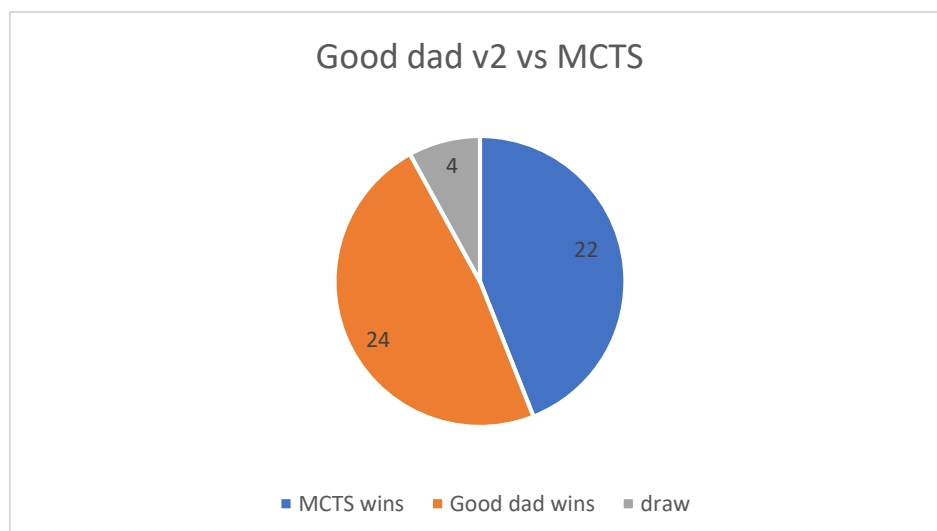


Figure 7-7 MCTS vs Good dad v2 win ratio



expected when the opponent does not play to win. Both AI had issue with the MinefieldAI though. This AI was designed to play moves that would win it sub boards by placing tiles in straight lines that were not interrupted by the opponents tiles. The assessment of why this might be is the MCTS algorithm may opt to send the opponent to a sub board that gives the opponent lots of options, as the roll out is random there is a chance the opponent does not pick the winning move, and so that move may be given a placement that is inaccurate. Or the Minefield AI play is so poor that the adaptive AI simply makes bad moves the entire game, not building its position while the MinefieldAI did just that. The standard deviation of the MineField is quite large at nine, meaning the Good dad v2 was basically playing randomly. Additionally, the Pearsons correlation coefficient for the reward of the Minefield AI against the Random AI was a 0.67, which indicates a strong correlation, and the coefficient for the move placements of these two AI was a 0.45, which indicates moderate correlation (Schober, et al., 2018). So, by the measures that both the adaptive AI implementations were taking indicate that the MineField and the Random AI have similar play styles so playing both of them in a

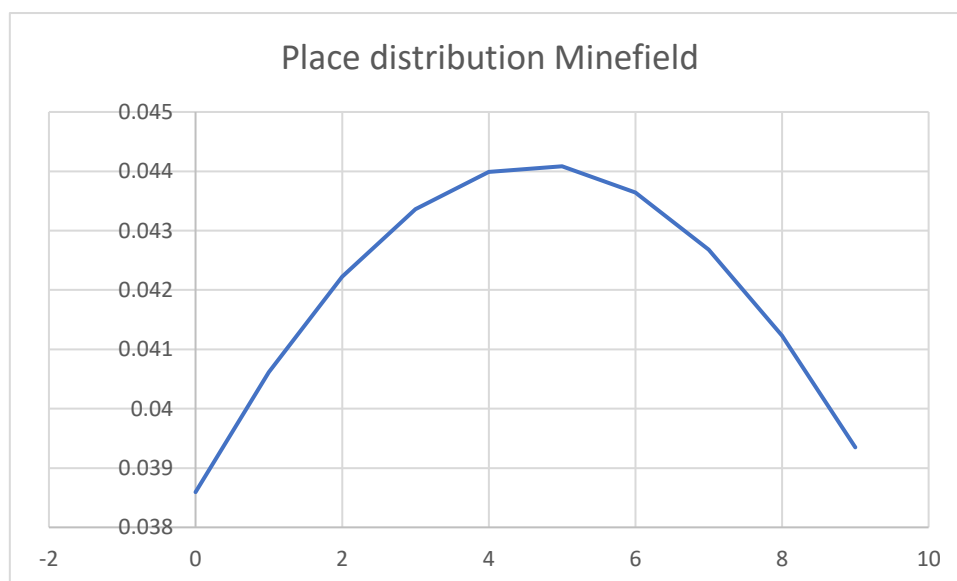


Figure 7-8 distribution of reward for Minefield AI

similar way is reasonable. However, this would explain how the MineFieldAI routinely out plays the adaptive AI solutions beating both Good dad and Good dad v2 with ease. To measure how adaptive the AI were the rating of both the adaptive and non adaptive AI were recorded. To calculate how well the AI adapted its game play the ratings of both the adaptive AI and the non adaptive AI's moves were recorded over a series of

games. When playing the MCTS player the Good Dad adapted its behaviour well on average. As it tended to play a close game, in terms of the reward both players were getting, until the final few moves of the game where the MCTS player would take a very advantageous move and the Good dad would struggle to recover. This result is similar for all match ups of adaptive and non adaptive AI.

## 7.2 Further Improvement

Though the adaptive AI did adapt quite well it did not play well against certain types of opponents, the Minefield AI being the most notable. To help correct this a heuristic could be introduced during the rollout stage of the MCTS. This could help the adaptive AI from making moves which give the opponent more options, but a game winning option. Furthermore, the code which forces the adaptive AI to pick a winning move could also be developed. Currently the solution simply checks for any move which gives a 100% chance of winning. The threshold value for a move which is too good not to

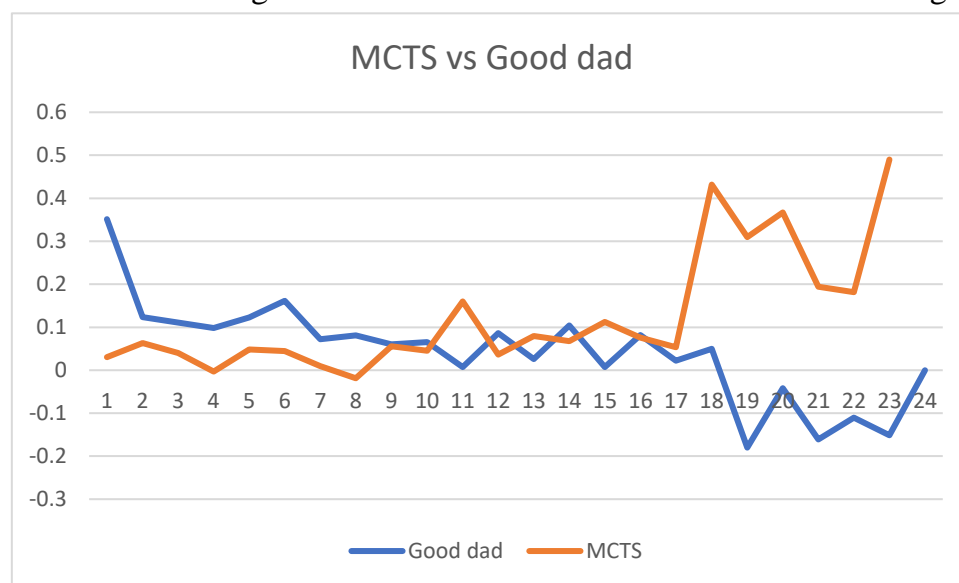


Figure 7-9 Reward for Good dad and MCTS during a game

take could be adjusted depending on the skill of the opponent, creating a system which will only make massively advantageous moves if the opponent would have also made that move.

### 7.3 User Evaluation

The user evaluation of the adaptive AI revealed some results that seem to contradict what is said about player enjoyment and difficulty by Chanel, G. & R. C. & B. M. & P. T., however, this discrepancy could have a reasonable solution. As can be seen in

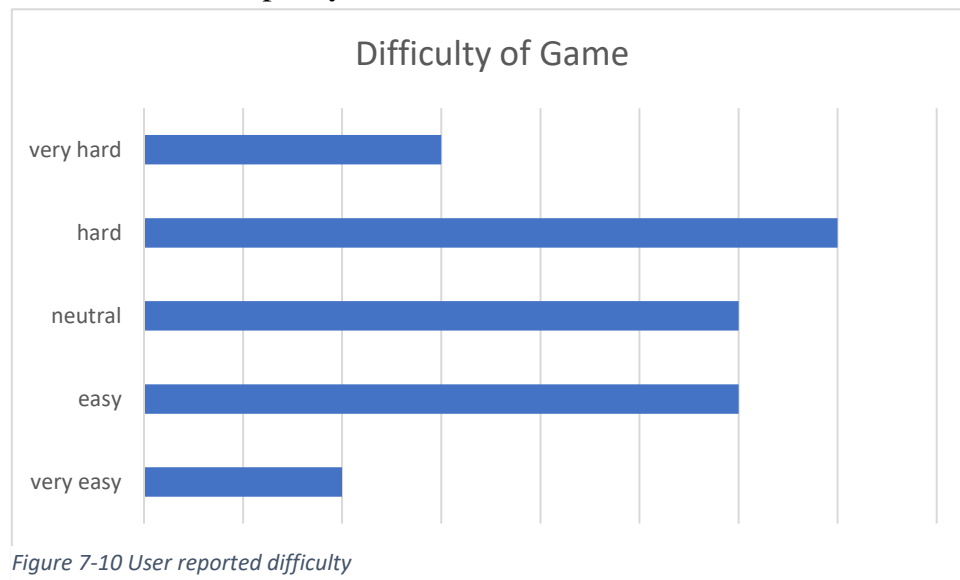
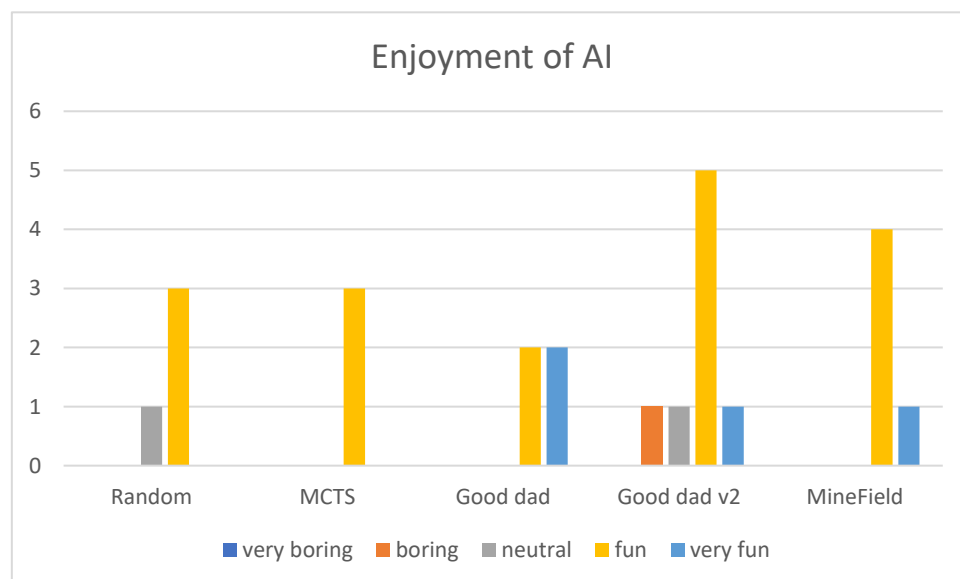


figure 7-10 users enjoyed the game in general. Out of twenty four games only one was found to be boring, this game was played against the Good dad v2. This data does not seem to correlate with the difficulty the users reported the game. This would seem to suggest that the enjoyment of a game is not correlated with the difficulty experienced



by the user. This result however, may be skewed by the fact that this is a novel game to the user, and the experience of learning it may have resulted in a more enjoyable game. Moreover, as there are no external rulers by which the performance of the AI can be

measured, and due to the users inexperience with the game, their percieved difficulty may also be somewhat inaccurate. Interestingly, the Minefield AI, though not rated any more enjoyable to play against than any other AI, received comments that noted its play style. Though users did not feel tense as intended they did feel baited, and that the AI

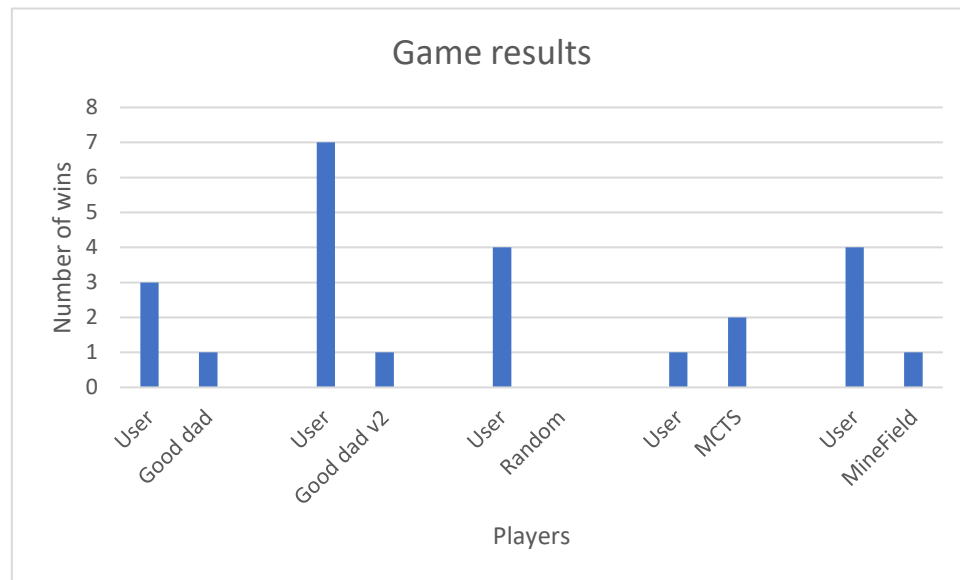


Figure 7-12 User game results

was more intelligent than some of the others. One user even reported the Minefield player as “clever” though it had one of the simplest heuristics out of all the AIs implemented. One user commented “I felt almost like it was trying to distract me,” and “It led me to do moves that allowed the opponent to win the subsequent [sub boards]”. This is not the intended effect of the MinefieldAI but does show that this AI seem to

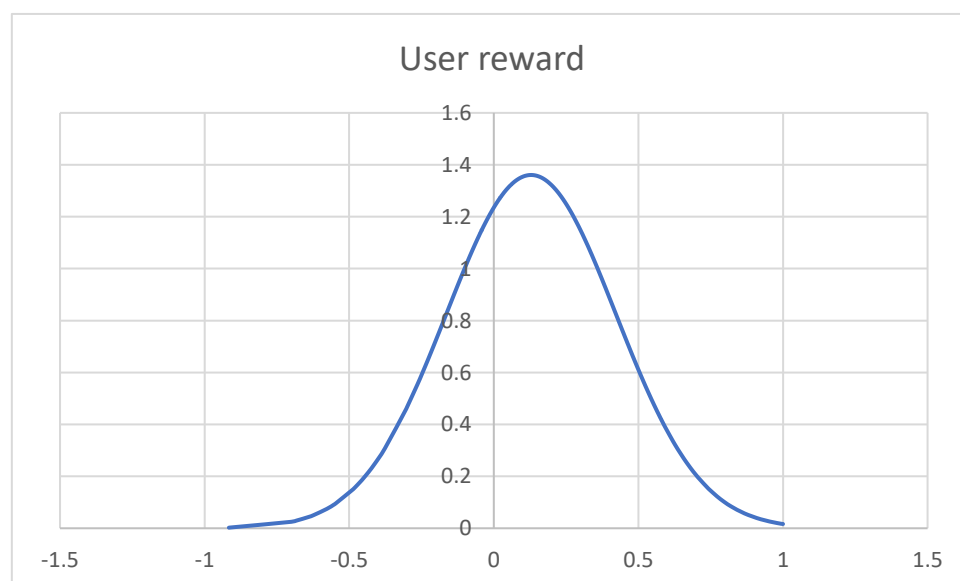


Figure 7-13 Distribution of users reward

illicit an emotional response from the player, even if it was not the response intended. Moreover, users commented on the AI seeming inability to win sub boards. One user commented about the MCTS AI that “It turned down two moves that would have won a sub-board.” The participants often got too invested in winning sub boards even though it may not have taken them any closer to winning the game. However, in general users seemed to play at a generally good standard, on average playing moves that were in the fifty fifth percentile. The users also were accurate in assessing their own performance, as they consistently reported that they performed well during the game. Overall the users consistently beat the AI opponents, the only AI which managed to have a possitive win loss ratio is the MCTS AI but it only played three games.

## 8 Summary and Conclusion

This chapter will summarise the work executed during this project. Additionally, it will identify issues with the implementation in certain sub systems of the codebase and address the success of the Good Dad AI. Potential future work will also be described to improve the current implementation in terms of performance as well as create more correct implementations of certain sub systems.

### 8.1 Summary

The objective of the project was singular to begin with: develop an AI which was more enjoyable to play against than traditional AI. The success of this objective is hard to measure as results seem to be skewed towards a more fun game than anticipated. However, the Minefield AI seemed to be successful in its objective to play on the user's emotions and so seemed to produce a more enjoyable game. However, as development progressed the main objective got more granular.

Objective	Success
<b>Select a game, it should be quick to play and learn</b>	Ultimate tic tac toe as it is familiar but still novel and quick to play.
<b>Develop the game</b>	The choice was made to implement UTTT as it would give greater control over the model
<b>Traditional AI</b>	There are two traditional AI implemented but as there are no outside ruler
<b>Evaluation</b>	Evaluating the user and AI moves does not work as intended however, this is an issue with MCTS
<b>Adaptive AI</b>	Two adaptive AI were implemented with similar results, though their methods were quite different
<b>AI Evaluation</b>	The AI performed well when playing users. Though the novelty of the game

	may have resulted in a more fun game than it would have otherwise been.
--	---

## 8.2 Game Implementation

The selection and implementation of the game were carried out successfully. A game of UT TT takes between five and ten minutes and while the game is much more complicated than its basic format, it still did not pose any issues for users to learn. The implementation of the game was carried out in stages, as new functionality was needed from the system it was added accordingly, the same can be said for every part of the system. Of all games considered the reasons that UT TT was chosen are that it is quick to play, this was desirable as if games took more than ten to fifteen minutes participants in the survey may start getting bored due to the amount of time spent and it is quick to learn. Given that the game is a variation of the class users were quick to pick up the fundamentals, but the game still had the strategic depth required to make it interesting to play. The implementation of the AI did not follow as clean a development cycle as initially intended. The first iteration of the AI makeMove method simply took the board and mutated it. This was problematic as AI could simply mutate a board to a winning state for them and win, secondly it gave users information about how much processing the AI was doing. The less complex, and hopefully easier, AI returned their moves nearly instantly while the more complex AI took significantly longer. Users could reasonably guess at the complexity of the AI if their response times were not flattened. This was done by performing the MCTS algorithm for every AI and simply having them return the move they thought was best. This change involved significant development in the Player interface but otherwise was quick. However, it did raise an issue for the single responsibility principle as the GameService was now performing MCTS. This was acceptable however as the benefits that the system gained by this arrangement outweighed the negatives that breaking the principle brought.

## 8.3 Good dad Summary

There were two competing ideas at the beginning of this project. One was to use the average reward of an opponent to adjust the play of the AI and the second was to use the average move placement. Initially the former was implemented as it was reasoned that the placement of moves might not track well with the reward gained. For example,

a user might consistently pick the second-best move, but the first best might be significantly better. For this reason, the average reward was chosen as it was being more consistent between moves. However, as experiments have shown this AI did not behave as intended as the game progressed. For this reason, Good Dad v2 was implemented which used a slightly more sophisticated method that was still based on the average move placement gained by the opponent. This AI better mirrored the AI opponent it played against, though it still struggled to beat the Minefield AI. This was found to be caused by the fact the Minefield AI played, according to the measures taken by the AI, to play in a similar way to a Random AI.

Evaluating the adaptive AI did not produce the results that matched the hypothesis, that games which the players are matched in skill are more enjoyable. The users overwhelmingly thought the game was enjoyable despite the intended difficulty of their AI opponent. The Minefield AI however did gain some interesting comments as its unique play style was noticed by users. Use of an external ruler to help create a more accurate picture of how well the AI in this project perform would have been very useful in evaluating actual skill of the AI.

#### 8.4 Future work

Going forward there is some work that would help improve the quality of the product produced in the project. Firstly, in regards with the MCTS algorithm used there are two issues. The easier to address is that the algorithm biases the first nodes it visits from a child. If two child nodes have the same value the first one visited will be the one that is expanded every time. This could be corrected by adding a small random value to the reward when it is called so the algorithm would not be biased in this way. The second is a bit more cumbersome to address. That is the fact that the same game state might exist twice in the tree. This is an issue as it over represents that game state and may impact the algorithms ability to make decisions. To correct this when a node is expanded if any of the resulting game states already exist in the tree it should not create a new node but instead should get a reference to the node that already exists in the tree. Moreover, as the game state is quite large at  $1.8 \times 10^{78}$  (Bredyn McCombs, 2018) cutting down the tree would be massively useful. By implementing the methods such as rotating



and mirroring the board (George & Janoski, 2016) the potential search space could be cut down significantly. Additionally, the decision to implement the AI and the game logic on the server side was made so that the user's own device did not impact on the performance of the AI, even if the MCTS AI performed poorly it performed equally poorly on every user's device. This seems to have been a misstep as letting the AI run on the client side would have gained some significant advantages. It would have allowed the MCTS algorithm to be ran while the user is making a move, this could be done with the current system however this would need two different implementations of the MCTS algorithm and for the tree structure to be sent to the server. Doing this would allow the AI to make a better-informed decision. Moreover, running the game solely on the client side would mean that the moves could be saved with a reference to the game. This would make data collection a lot easier as moves could be grouped together and sent to the server to be stored with a reference to the game that they were played in. Another option is also available for an adaptive AI system, a neural network. A neural network system could be built to analyse moves made by an opponent and produce the move they would have made. Due to time constraints this AI, though it would have been interesting, was never implemented. Lastly the implementation of the adaptive AI is not dependent on the game in this project. Extending more games to meet the interface and testing how the AI systems developed in this project perform would be of great interest. In terms of the experimentation, the conclusion that the game being novel could be easily tested with a longer study. Participants could be asked to play for a series of games, 10 or more, so the experience becomes less novel and the gameplay provided by the AI opponent becomes the focal point not the game itself.

## References

- Bredyn McCombs, C. L., 2018. *Ultimate Tic Tac Toe Search Strategies*. St George, Dixie State Regional Symposium.
- Chanel, G. & R. C. & B. M. & P. T., 2008. *Boredom, Engagement and Anxiety as Indicators for Adaptation to Difficulty in Games*. New York: ACM New York, NY, USA ©2008.
- Donlan, C., 2014. *eurogamer*. [Online]  
Available at: <https://www.eurogamer.net/articles/2014-05-22-middle-earth-shadow-of-mordor-promises-plenty-of-strategising-and-accidental-chaos>  
[Accessed 5 March 2019].
- Eric Freeman, E. F. K. S. B. B., 2004. *Head First Design Patterns*. s.l.: O'Reilly Media.
- George, W. & Janoski, J. E., 2016. *Group Actions on Winning Games of Super Tic-Tac-Toe*. [Online]  
Available at: <https://arxiv.org/pdf/1606.04779.pdf>  
[Accessed 21 03 2019].
- Grant, E. F. & Lardner, R., 1952. *The New Yorker*. [Online]  
Available at: <https://www.newyorker.com/magazine/1952/08/02/it>  
[Accessed 15 March 2019].
- Hao Cui, R. K. A. F. P. T., 2015. *Factored MCTS for Large Scale Stochastic Planning*. Austin, AAAI.
- Hunicke, R. C. V., 2004. *AI for dynamic difficulty adjustment in games*. 2nd ed. s.l.:s.n.
- Jozsef Beck, J. B., 2008. *Combinatorial Games: Tic-Tac-Toe Theory*. Cambridge: Cambridge University Press.
- Kocsis, L. & Szepesvári, C., 2006. Bandit based Monte-Carlo Planning. In: J. Fürnkranz, T. Scheffer & M. Spiliopoulou, eds. *Machine Learning: ECML 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 282-293.
- Lidén, L., 2004. Artificial Stupidity: The Art of Intentional Mistakes. In: *AI Game Programming Wisdom 2*. s.l.:Charles River Media, pp. 41-48.
- Mirna Paula Silva, V. d. N. S. L. C., 2015. *Dynamic Difficulty Adjustment through an Adaptive AI*. Piaui, SBGames.
- Orlin, B., 2013. *mathwithbaddrawings*. [Online]  
Available at: <https://mathwithbaddrawings.com/2013/06/16/ultimate-tic-tac-toe/>  
[Accessed 15 March 2019].
- Schober, P. M. P. M., Boer, C. P. M. & Schwarte, L. A. M. P. M., 2018. Correlation Coefficients Appropriate Use and Interpretation. *Anesthesia & Analgesia* , 126(5), pp. 1736-1768.
- Spronck, P. P. M. S.-K. I. e. a., 2006. Adaptive game AI with dynamic scripting. *Machine Learning*, p. 63\*217.
- Spronck, P., Sprinkhuizen-kuyper, I. & Postma, E., 2004. *Difficulty scaling of game AI*. Ghent, GAME-ON.
- Suddaby, P., 2013. *tutsplus*. [Online]  
Available at: <https://gamedevelopment.tutsplus.com/articles/hard-mode-good-difficulty-versus->

bad-difficulty--gamedev-3596

[Accessed 15 March 2019].

Tan, C. H., Tan, K. C. & Tay, A., 2011. Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based AI. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 3(4), pp. 289-301.

TheOFekFoundation, n.d. <https://www.theofekfoundation.org>. [Online]

Available at: <https://www.theofekfoundation.org/games/UltimateTicTacToe/>

[Accessed 18 March 2019].

Wei-Yuan Hsu, C.-L. K. C.-H. H. I.-C. W., 2019. Solving 7,7,5-game and 8,8,5-game. *ICGA Journal*, 40(3), pp. 246-257.

Wikipedia, 2017. *Wikipedia*. [Online]

Available at: [https://en.wikipedia.org/wiki/File:MCTS\\_\(English\)\\_-\\_Updated\\_2017-11-19.svg](https://en.wikipedia.org/wiki/File:MCTS_(English)_-_Updated_2017-11-19.svg)

[Accessed 22 03 2019].

## Appendix A – User Guide

---

The user guide for installing, running and testing the server and the client.

### Installing Frameworks

There are two frameworks used in this project which must be installed if you wish to run and test the project locally. These frameworks are .Net core for the server and Angular CLI for the client. .Net core can be installed from Microsofts website here: <https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install>. The client requires that NPM is installed to manage node packages. NPM is installed along side node here: <https://www.npmjs.com/get-npm>. You can verify installs by running the dotnet command in your terminal for .Net and the npm command for NPM. Lastly you can install Angular CLI with the command “npm install -g @angular/cli”, you will need to restart terminal to load Angular CLI. You can confirm Angular CLI has installed by running the “ng” command.

### Running the project

Once NPM, Angular CLI and .Net are installed open a command prompt and navigate to the directory which you have saved the project to. Navigate to the UltimateTicTacToe folder. Run the “dotnet restore” command to install the project dependencies for the server. Navigate to the “ClientApp” directory and run the “npm install” command to install the dependencies for the client. Once the installations are complete you can run the client without the server by running “npm start”, this may be useful if you have your own server implemented and want to point this client at it. To run the server, navigate back the UltimateTicTacToe directory. Here you will find the DataAccess directory, in there is a file named “connectionString.json” you must provide a connection string for the SQL database you are using (See Appendix H for SQL setup), if you do not require a connection string or are not using an SQL database please see Appendix B for guidance on updating the DataAccess layer. Once that is done you can navigate back to the UltimateTicTacToe directory. From here run the “dotnet run” command. This may take a few minutes to run but it will start the server on your local host listening to port 5000 if no service is currently on that port. If a service is on that port you can either close it or add the –urls argument to the dotnet command, “dotnet run --urls=https://localhost:[preferred port here]”. Soon the application will launch the client

to what ever port the server is listening on and you can interact with it. To kill the process press “ctrl + c” in the terminal running the server.

### Testing the project

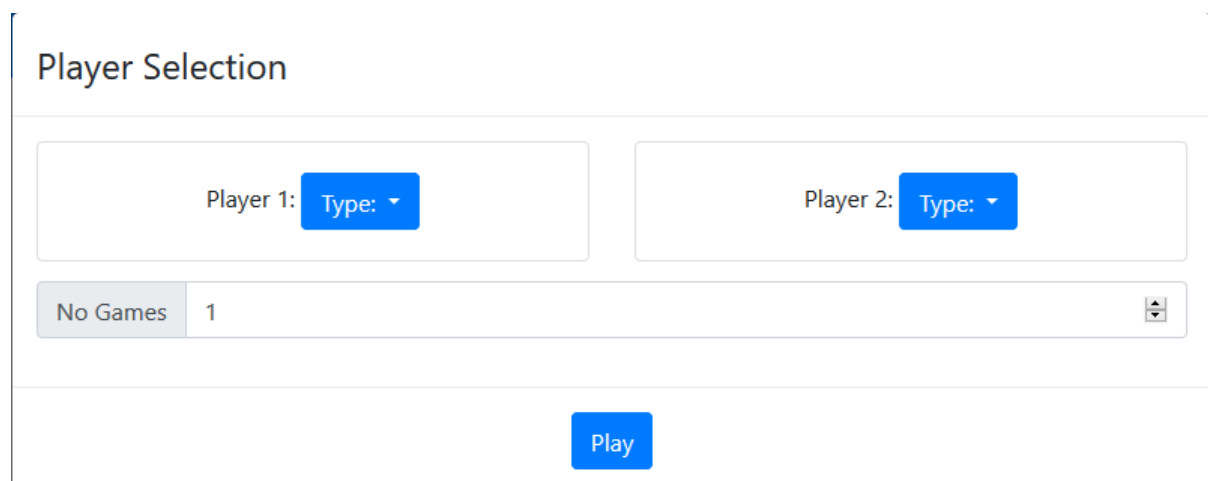
To test the server, open the command prompt and navigate to the directory where you have the project saved. Navigate to the UltimateTicTacToeTests directory. Run the command “dotnet test”, this will run all the tests for the server and print out the results. To test the client, navigate out of the UltimateTicTacToeTests directory and into the UltimateTicTacToe directory. Once there navigate to the ClientApp directory and run the “npm run test” command, this will run the tests once and print the results to terminal. Alternatively, you can run “npm run testDebug” this will open a headless chrome browser and run debug-able tests there.

### Login

There are two interactions on the Login screen, login as a user or create a new user. To login type your name into the text entry field left of the login button then press the login button. A successful login will navigate you to the game screen and a confirmation message will appear in the top right corner of the screen letting you know you’ve logged in. Unsuccessful logins will have you remain in the login screen and an error message will be displayed. To create a new user simply press the button labelled “create user”. This will create and log you in as a new user. Your UserID will be displayed in the login message in the top right corner.

### Game Setup

The game setup allows you to select the players you wish to have play a game and the number of games you want them to play. Players can be selected using the drop-down

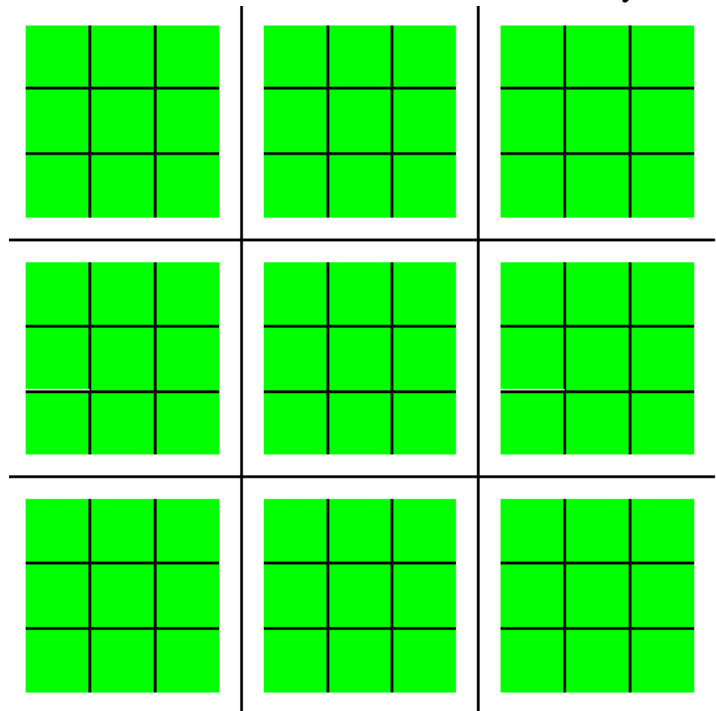


The screenshot shows a web interface titled "Player Selection". It contains two input fields for "Player 1:" and "Player 2:", each with a blue "Type:" dropdown button. Below these is a row with a grey "No Games" button, a text input containing the number "1", and a small up/down arrow icon. At the bottom center is a large blue "Play" button.

menus. The number of games that you wish to queue up can be entered in the “no Games” input field beneath the drop-down menus for player one and player two. To start the queued games press the “play” button.

### Game Play

Once a game has been started you will be greeted with the game screen. If an AI opponent was selected as player one, then the board will have their move already made on it. By convention player one will play blue and player two will play red. The available moves will be denoted by green tiles. The last move made will be highlighted by an orange border. A finished sub board will have a border applied to it. If the sub board has been won then the boarder will be the colour of the winning player, otherwise it will be grey.

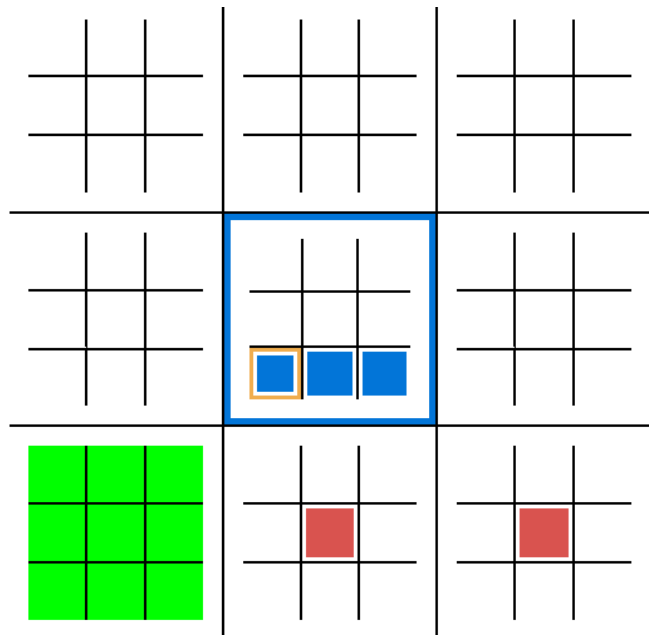


### Making a move

As stated previously, available moves will be highlighted with green. Once you have chosen your move simply click on screen where you wish to make your move. If you are playing an AI opponent, it will make its move automatically. If you are playing another person they can now make their move.

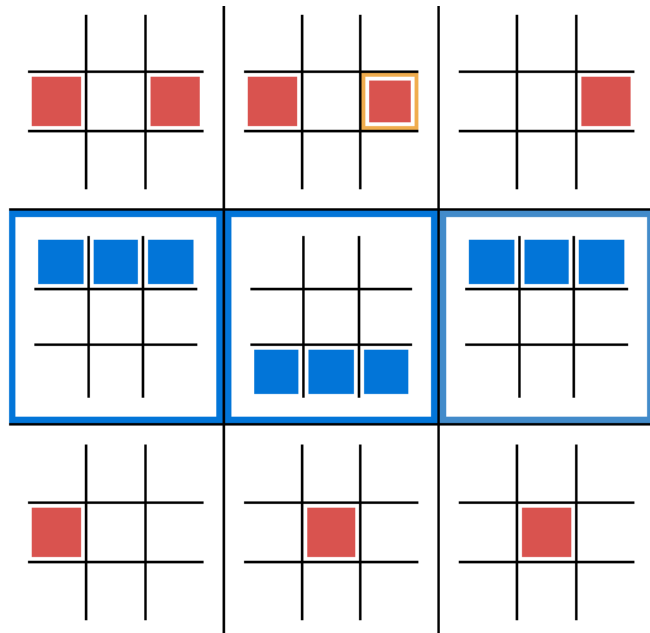
### Winning a sub board

A sub board is won using regular tic tac toe rules. If any player manages to place three of their colour in a row in that sub board then they have won that sub board. If all cells of a sub board are owned by a player then the sub board is drawn.



### Winning a game

Regular tic tac toe rules apply here too. If a player has won three sub boards in a row then the player has won the game. If all sub boards are complete, either won or drawn, but no player has three in a row then the game is drawn.



### Game over screen

Once the queue of games has been completed you will be presented with the game over screen. This screen will tell you the winner of the last game along and present you with the option to play again. Selecting yes will start another game with the same opponent and selecting no will present you with the game setup screen where you can set up a new game.

## Game Over

---

BLUE Has won!!

Would you like to play again:

---



## Appendix B - Maintenance Guide

---

### Guide for adding new functionality to the code base.

#### Players

There are a few steps involved for adding new player to the game. Firstly you must create a concrete class which implements the `AbstractPlayer` interface. Once your implementation is complete you must add a new enumeration to the `PlayerTypes` enum on both the server and the client side. The `AbstractPlayer` and `PlayerTypes` enums can be found at `./UltimateTicTacToe/Models/Players` on the server and `./UltimateTicTacToe/ClientApp/src/app/models/player` for the client. The enumeration on the client side is named `“player.type.enum.ts”`. Your player should have its enumeration placed between the last AI player and HUMAN. Once you have added your player to the enumeration you can create the factory for it. To do this you will want to extend the `AbstractPlayerClassHandler` class in the `services/playercreation` folder. You will need to assign the field `“type”` to the enumeration your player has and the `build player` method should simply return a new instance of your player. Lastly you will need to follow the Chain of Responsibility to the last link and add your new class handler to the end of the chain.

#### Data Access

If you do not want to use a SQL database, or are using one which does not require a connection string then you will need to implement the `IDatabaseProvider` interface. The interface can be found at `./UltimateTicTacToe/DataAccess`. Once you have completed your implementation you will need to give your instance to the container so it can be injected into other parts of the application. To do this find the `Startup.cs` file in the root directory of this solution. There is a method in this class called `ConfigureServices`. You will need to find the line which reads `services.AddSingleton<IDatabaseProvider, SQLDataBaseProvider>()`; and change `“SQLDataBaseProvider”` to your class. When the application is now started your implementation of the `IDatabaseProvider` interface will be loaded.

## View

If you wish to change the view you must not change any of the classes in the “Services” folder of the client. These classes are designed to give events from the server and unless the controllers are changed on the server these services should remain as they are. You can however, add new components to the project by leveraging the Angular framework. Help can be found here: <https://angular.io/>. You cannot change the framework of the view without greater knowledge of the .Net framework and how it interacts with Angular.

## Appendix C – The Stealing Strategy

---

The stealing strategy was invented by John Nash in the 1940s, to show that in the game of Hex that the first player can always win (Jozsef Beck, 2008).

### Example of the strategy

In a game of tic tac toe where the board is any size, we will suppose that player two has a strategy,  $S$ , that guarantees them a win. The first player places an  $X$  in a random position. The second player now takes their turn according to  $S$ . But if they ignore the first random  $X$  that was placed on the board, then the first player is in the same situation that the second player is on their first move, a single enemy piece on the board. The first player can now make a move according to  $S$ , unless that move is where they have already placed an  $X$ . In this case the player can simply place his  $X$  randomly on the board. The result of this is that there will be one  $X$  in a place demanded by  $S$  and one random  $X$  on the board. Player one can continue playing by  $S$  in this manner. Doing this way  $S$  is guaranteed to produce a win, with an additional  $X$  on the board. But as player two has lost this contradicts our previous statement, that  $S$  is a winning strategy. Therefore there is no such winning strategy for player two and tic tac toe is either a forced win for the first player or a draw.

## Appendix D – User Evaluations

The user evaluations of the AI systems were carried out with twelve participants in total. This is a small sample size and so data gathered may not be completely accurate, however, due to time constraints more experiments were not feasible.

### Participant Information Sheet



## Participant Information Sheet for AI Testing of Good Dad AI

**Name of department:** Computer and Information Sciences  
**Title of the study:** AI for Games: Being A Good Dad

### Introduction

Hello my name is Mauro Di Nardo, I am 5<sup>th</sup> year Software Engineering student here at the University of Strathclyde. I am currently undertaking my dissertation project on adaptive AI (Artificial Intelligence) for games. This investigation aims to explore which type of AI opponent people prefer playing against in a board game.

### What is the purpose of this research?

This research aims to find what type of AI people enjoy playing against. Typically, modern games offer a few AI set at static difficulties, these tend not to be very engaging as they are either too easy that they do not provide a challenge, or too difficult to the point where they do not give the player a chance. An adaptive AI is one which changes its play style to match the skill level of the opponent.

In this study I hope to evaluate how much players enjoy playing against the adaptive AI and how their experience with it differs from more typical AI.

### Do you have to take part?

No, participation is entirely voluntary. Withdrawing from this investigation will not impact the way you are treated. You have the right to withdraw your results from the investigation at any point without any justification. If you do decide to withdraw your participation during the study any and all data related to you will be destroyed and not analysed. However, after this study data will be kept anonymous and analysed, your results will not be removable from the analysis.

### What will you do in the project?

During this study you will be asked to play a game of Ultimate Tic Tac Toe against a random selected AI. The rules of the game will be explained before you play, and you can play one game to familiarise yourself with the game. After the game with the anonymous AI you will be asked a few questions about your experience of the game. You will then play as second randomly selected AI and, after a game with it, will be asked again about your experience of the game. Lastly you will be asked to compare the two games. The entire study should take no longer than 30 minutes.

### Why have you been invited to take part?

You have been asked to take part because you are over the age of 18.

The place of useful learning

The University of Strathclyde is a charitable body, registered in Scotland, number SC015263



**What are the potential risks to you in taking part?**

The study requires you to sit in front of a computer and use computer and mouse inputs. The potential risk has been deemed minimal.

**What information is being collected in the project?**

In this project you will be asked to evaluate your experiences with two AI. The results of your games will be recorded, along with how well you played. All information provided will be used to measure the performance of the adaptive AI, and measure what AI you enjoyed playing against. The information will be presented in a table in my final report. There is the potential that direct quotes from your session may be made in the report (These will not refer to you by name).

**Who will have access to the information?**

Information collected during this study will be made pseudo-anonymous, you will be given a numeric ID, and kept confidential. No identifying information will be collected during this study. If you choose to withdraw your participation from this study, it must be done before information has been analysed as results from the analysis cannot be altered. Your information will be automatically recorded and stored in a secure database.

**Where will the information be stored and how long will it be kept for?**

Information about how well you played as well as the results of your game will be automatically recorded by the system. Data will be stored in a secure database. No personal information will be retained after the study is complete and all information collected will be destroyed within 2 years.  
data with no consent form]

**What happens next?**

If you want to participate in this study, please sign and return the consent form overleaf.

If you do not want to participate, thank you for taking the time to read this.

**Researcher contact details:**

Mauro Di Nardo – Strathclyde Computer and Information Science  
LT1420 – Livingston Tower, 26 Richmond Street, Glasgow, G1 1XH

Email – [mauro.di-nardo.2014@uni.strath.ac.uk](mailto:mauro.di-nardo.2014@uni.strath.ac.uk)

**Chief Investigator details:**

Dr. John Levine – Strathclyde Computer and Information Science  
LT1420 – Livingston Tower, 26 Richmond Street, Glasgow, G1 1XH

Telephone – 01415484524      Email – [john.levine@strath.ac.uk](mailto:john.levine@strath.ac.uk)

This research was granted ethical approval by the University of Strathclyde Ethics Committee.

The place of useful learning

The University of Strathclyde is a charitable body, registered in Scotland, number SC015263



If you have any questions/concerns, during or after the research, or wish to contact an independent person to whom any questions may be directed or further information may be sought from, please contact:

Secretary to the University Ethics Committee  
Research & Knowledge Exchange Services  
University of Strathclyde  
Graham Hills Building  
50 George Street  
Glasgow  
G1 1QE

Telephone: 0141 548 3707  
Email: [ethics@strath.ac.uk](mailto:ethics@strath.ac.uk)

The place of useful learning

The University of Strathclyde is a charitable body, registered in Scotland, number SC015263



## Consent Form for AI Testing of Good Dad AI

**Name of department:** Computer and Information Science

**Title of the study:** AI for Games: Being a Good DAD

- I confirm that I have read and understood the Participant Information Sheet for the above project and the researcher has answered any queries to my satisfaction.
- I confirm that I have read and understood the Privacy Notice for Participants in Research Projects and understand how my personal information will be used and what will happen to it (i.e. how it will be stored and for how long).
- I understand that my participation is voluntary and that I am free to withdraw from the project at any time, up to the point of completion, without having to give a reason and without any consequences.
- I understand that I can request the withdrawal from the study of some personal information and that whenever possible researchers will comply with my request. This includes the following personal data:
  - My personal information from transcripts.
- I understand that anonymised data (i.e. data that do not identify me personally) cannot be withdrawn once they have been included in the study.
- I understand that any information recorded in the research will remain confidential and no information that identifies me will be made publicly available.

I consent to being a participant in the project.

(PRINT NAME)

Signature of Participant:

Date:

The place of useful learning

The University of Strathclyde is a charitable body, registered in Scotland, number SC015263

## Participant Questionnaire

### Participant Questionnaire Sheet for AI Testing of Good Dad AI

Computer and Information Science

AI for Games: Being a Good Dad

**Please answer all questions honestly and fully**

#### Game 1:

How enjoyable was the game?	(boring) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (fun)
How difficult did you find the opponent?	(easy) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (hard)
Do you have any comments on the opponent's performance?	
How do you feel you preformed during the game?	(poorly) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (well)

#### Game 2:

How enjoyable was the game?	(boring) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (fun)
How difficult did you find the opponent?	(easy) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (hard)
Do you have any comments on the opponent's performance?	
How do you feel you preformed during the game?	(poorly) <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> (well)



## Evaluation results

These games are in order by which they were played

PlayerOne	PlayerTwo	Winner	How enjoyable was the game?	How difficult did you find the opponent?	How do you feel you performed during the game
MCTS	User	User	fun	easy	well
User	Good dad v2	User	fun	very easy	well
Random	User	User	fun	very easy	well
Good dad v2	User	User	fun	neutral	well
Random	User	User	fun	neutral	average
Good dad	User	User	fun	easy	well
MineField	User	User	fun	hard	poorly
User	Good dad v2	User	neutral	hard	poorly
Random	User	User	neutral	hard	well
User	Good dad v2	Good dad v2	boring	very hard	poorly
MineField	User	User	very fun	hard	average
User	Good dad	User	very fun	easy	well
MCTS	User	MCTS	fun	neutral	average
User	Good dad v2	User	fun	neutral	average
Random	User	User	fun	neutral	well
User	Good dad v2	User	very fun	very hard	well
MineField	User	User	fun	easy	well
Good dad v2	User	User	fun	hard	well
MCTS	User	MCTS	fun	hard	well
User	MineField	User	fun	easy	very well
User	Good dad v2	User	fun	neutral	average
Good dad	User	Good dad	fun	easy	well
MineField	User	MineField	fun	hard	poorly
User	Good dad	User	very fun	very hard	average

These comments come in the same order as the games above.

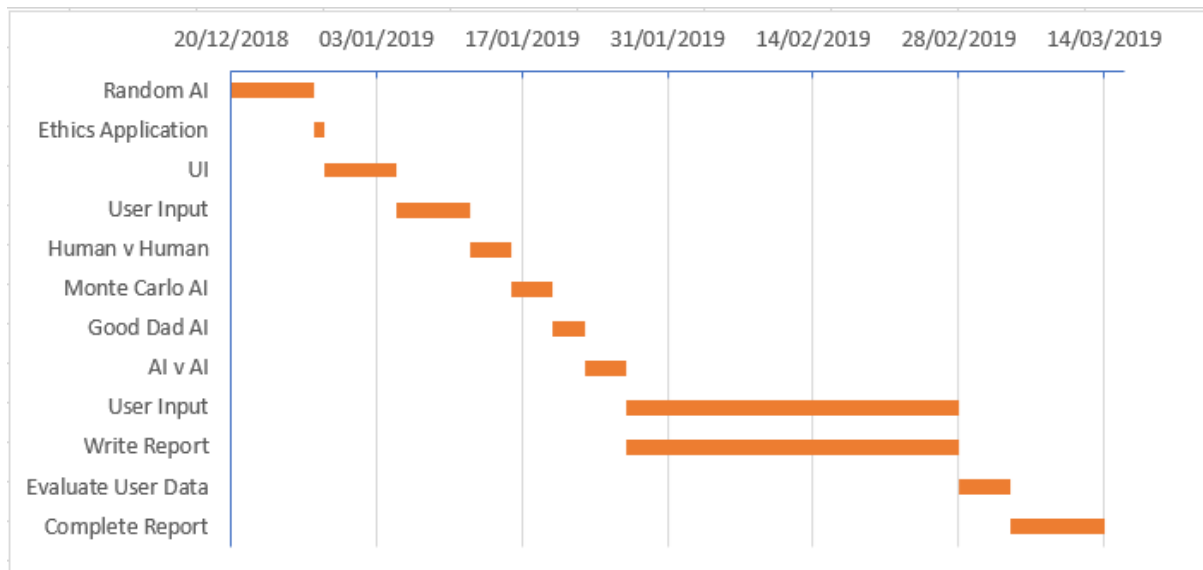
Do you have any comments on the opponents performance2
occasionally the AI made moves i would deem 'poor' in terms of overall game state. The ai was seen to be making a general strategy, but could be easily outplayed
AI could be seen to be making gneral strategies, although on multiple occasions it made poor moves that ended with myself the game.
Didn't play very strategically.
Only seemed to consider the square it was on.
my opponents performance was poor,because it let me win
A couple of obviously incorrect moves made by the opponennt, that they didn't take full advantage of my mistakes made
Felt opponent was making foolish moves a first, however after seeing where they lead me and what options I had left, made me think that the A.I was either 'baiting' me into making the wrong move or operating many more moves ahead than I was, perhaps based off my previous moves made. Felt opponent was attempting only diagonal methods of victory.
Felt though at times i was being set up. However it was quite unpredictable.
Even though i was first (red) etc it felt thouth i was the one chasing the game.
It performed well and knew how to bait me as its opponent, it did this by not giving me a choice between performing an advantageous move and losing a board. It led me to do moves that allowed the opponent to win the subsequent. I felt almost like it was trying to distract me. I feel I won purely based on luck.
The AI wasn't as sneaky, if there was a part of the board that was neglected it moved the game over there. It didn't seem interested in winning.
it did make some move that where tactically sound but also made some major mistakes
the opponent showed stratigical thinking in its moves and showed a overall strategy inn its play making moves that would make it harder for me to win
Opponent was good at setting up their own sub-boards but not at playing aggressively e.g. trying to force me into positions where I have to give up a board or trying to stop me from claiming a board.
Opponent made sure to put me in a sub-board which prevented me from winning a board or blocking off the sub-boards they were building up.
It turned down two moves that would have won a sub-board.
Could have won the game on the last move but did not do so. However, turned down winning a board to prevent me winning a board (I think.)
clever
made it very tricky

## Appendix E – Git commit log

The git commit log for this project can be found here:

<https://github.com/mDinardo08/dissertation/branches>

## Appendix F – Project plan



## Appendix G – Web Application address

The web application served at: <https://ultimatetictactoedissertation.azurewebsites.net>

## Appendix H – SQL

If you wish to create an SQL database and do not wish to change the current DatabaseProvider you can run this SQL to create the database correctly.

Create Table Ratings (UserID Primary Key int, LatestScore float, TotalScore float, TotalMoves int)

Create Table Moves (ID Primary Key int NOT NULL (1, 1), Owner int REFERENCES Ratings(UserID), Reward Float, Place int)

Create Table Games (GameNo int, PlayerOne int REFERENCES Ratings(UserID), PlayerTwo int REFERENCES Ratings(UserID), Winner int REFERENCES Ratings(UserID))

Once completed the current DatabaseProvider will be able to correctly insert data in your database given you have provided a connection string in the connectionString.json.

## Appendix I – Bandit Problem

The multi armed bandit problem is a problem in which some number of decisions must be made between competing choices to maximise gain. Classically this is thought about in terms of One Arm Bandit betting machines, where you have some number of one arm bandits and must attempt to maximise the gain from playing them.