School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCE

Developing an Artificial Intelligence capable of playing the board game Santorini.

Jack Edward Boreham

Submitted in accordance with the requirements for the degree of Computer Science BSc

2019/2020

40 credits

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final report	PDF version	Via Minerva Submission portal (06/05/20)
Source code and Program	Project GitLab URL	Supervisor,Assessor(06/05/20)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Jack Boreham

Summary

This project explores the creation of an artificial player capable of playing the board game *Santorini* with the aim of defeating a novice human player. It is a game with certain features such as a high branching factor which may make traditional AI techniques ineffective, research into traditional and more contemporary techniques are explored whilst establishing the best solution.

Acknowledgements

I would like to thank my supervisor Brandon Bennett for his help throughout this project and his calming advice. I would also like to thank my Parents and Grandparents for without their support and prayers I definitely would not be where I am.

Contents

Chapter 1

Introduction

1.1 The problem

Creating Intelligent agents capable of playing games has been a common desire of computer scientists since even before the field of AI (Artificial Intelligence) was properly established, mathematicians were devising algorithms that would be able to evaluate a the state of a chess board and make an "intelligent" move as early as 1950 [15]. With the arrival of modern computers came the consideration of using "Type A" search methods outlined in Claude Shannon's paper that took a brute force search approach to finding the next move [15]. Quickly it was realised that brute force searching would not be sufficient due to time constraints, so techniques such as Minimax and Heuristic state evaluation [12] were developed to make the algorithms more efficient. It was not until computing power dramatically increased in the subsequent decades more that more complex complete information games such as Checkers, Chess, and most recently Go have had their human grand masters defeated by the best AI adversaries [6]. However as the complexity of the games have increased (Go), the AI techniques required to defeat them have evolved [16]. I plan to look at the game Santorini, a quite new and untested perfect information game with a large branching factor, investigating if traditional AI implementation will be sufficient to create an AI capable of defeating a human or whether or not a newer approach such as Reinforcement learning will be more effective.

1.2 Aims

The aim of this project is to identify and implement a combination of AI techniques that will successfully be able to defeat a novice human player at the board game Santorini.

1.3 Objectives

The intermediate objectives of the project act as a means of breaking the project down into a clearer more manageable challenge, they are outlined as follows.

- Conduct an in depth review of relevant information including games and Artificial Intelligence techniques in games.
- Develop a full human playable version of the board game.
- Implement a reduce size instance of the board game Santorini for trialling and testing AI techniques.
- Develop an agent that can intelligently play the board game.
- Refine and improve the agent to play well enough to beat a novice player of the game.

1.4 Project Plan

The Gantt chart below – Figure 1.1, details the full timeline of the project from the beginning of the 2019/2020 academic year. Up until December the main task of the project has been to conduct background research on the possible AI techniques to be implemented to play the game and the implementation of the board game. After the exam period ending on the 24th of January, implementing the game in code will be the first task that needs to be completed, as until this has happened there will be nothing to develop an AI to play, this task needs to be started promptly as it is a non-trivial task and the time which it may take is somewhat unknown to me, and without it there can be no further progression. By mid-February I expect to have a minimum implementation of the game so testing of techniques can begin. As can be seen in the chart I have the background research continuing up until the end of the implementation as I expect as the project progresses certain avenues of research to become less useful and new ones required to be researched in their place as problems and difficulties may occur during implementation. Once implementation is completed I will begin testing and evaluation as well as beginning the report along side this, this will be an intense period of work during March but a necessary one as I think leaving the report to the very end will cause undue stress if there is a much greater time pressure. I am hoping to have completed a first draft by the 6th of April, this will will give me a 3 week contingency period for my advisor to give me some feedback, to which I should have time to make adjustments based upon and to complete any additional testing I may need to conduct.

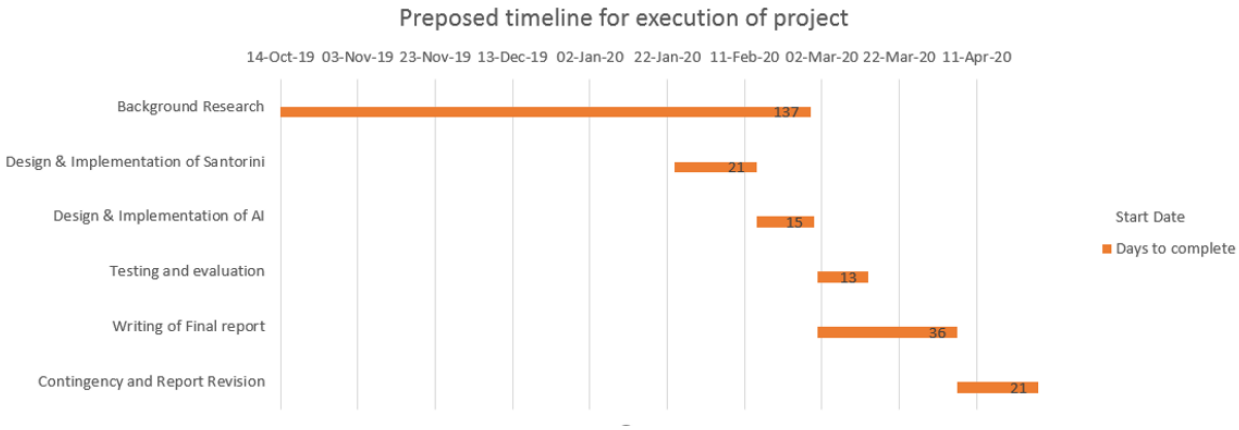

Figure 1.1: Gantt chart detailing project timeline

Also this will give me a chance to step away from the project for a few days before going back and reading it again, reviewing the structure and writing from a fresh perspective before finalising the report and submitting it on the 27th of April.

1.5 Risk Mitigation

Due to the nature of the project, the list of tasks generally requires the previous to be completed before the next one can begin, i.e Implementation cannot begin until adequate research has been conducted, and testing cannot begin until a minimally functional implementation has been successfully completed. This comes with inherent risk as if

implementation is halted for one reason or another it then stops the progress of the rest of the project. This is why I've tried to spread the project over all of the available time of the term as well as factoring in a contingency time at the end, in case any of the project's aspects take longer than anticipated there is a 21 day surplus that can be eaten into if needed.

The main task which carries risk is the implementation of the Game, as without it I cannot progress, I intended to incrementally build the game from a minimal playable implementation to increasing the richness of features if there is time, I will track this and manage it through using GitHub, a version control system. This way I will have a detailed timeline of work flow and can re-visit previous versions of the program should I need to, this also makes creating new features less risky as I can hold a stable version of my game on the master branch and create new feature branches when implementing something additional. However, it is not until this minimal playable implementation is completed that I can begin work on implementing an AI, again I intend to create a minimal functional implementation of an AI capable of playing against an opponent making random moves. Then I will investigate alternative solutions and AI techniques capable of playing to a decent standard.

From a time management perspective, I need to be aware of the workload of my project within the context of the rest of not only my term, but my advisor's term also. As we will have other deadlines that need fulfilling be it coursework for me or marking for them, this will require a good channel of regular communication. I intend to meet weekly with my supervisor to check in with the progress I have made each week and whether or not they think my timescale and scope of the project is still feasible or that I'll need to readjust my goals.

Chapter 2

Background Research

Before the project plan can begin, first a comprehensive understanding of the problem space must be achieved. This way a clear formal definition of what is to be implemented can be outlined along with the best way to go about that. To achieve this, research into the areas of Game theory, Adversarial games, and AI techniques for perfect information games has been conducted.

2.1 Game Theory

Game theory is the field of study of conflict and cooperation. Pioneered by mathematicians such as Emile Borel and Jon Von Neumann in the 1920s, it was later definitively established by the publication of "Theory of Games and Economic Behaviours" by Von Neumann and Economist Oskar Morgenstern [18]. It is the study of modelling situations or "Games" in which two or more parties interact in a strategic manor in which their decisions affect the outcome of the other parties' payoff. These situations are restricted by a set of rules by which the parties have to abide – *Not cheating.*

2.1.1 Definitions

To be able to discuss game theory and its application to different types of games and this project we must first define some of the terms and concepts used in the field to formalise situations. The following are taken from a textbook on the Basics of Game Theory [17].

- Game: Any set of circumstances that has a result dependent on the actions of two or more decision-makers (players).
- Rules: A formalised set of rules which enforce the restrictions of what moves are valid and moves that are not valid. These restrictions facilitate the need for players to make strategic decisions.
- Players: A strategic decision maker within the context of the game.
- State: A particular layout of a game's components
- Move: a decision made by a player in a given state.
- Not Cheating: Playing the game strictly adhering to the rules. A game cannot be modelled if the players do not adhere to the rules.
- Outcome: Outcomes are evaluated based on one or more moves made within the game.
- Payoff: The pay-out a player receives from arriving at a particular outcome. The optimal outcome which each player is after is winning the game.

- Rational Player: A player that makes decisions optimally in their self interest to maximise payoff, whilst being aware that other players are aiming to do the same.

2.1.2 Strategy

In a game, a move or decision is known as a strategy, each player has a of a set pure strategies, these are the moves available to them at when it is their turn to play, each of which will have a quantifiable payoff depending on the strategy selected by their opponent [5]. For example in Rock, Paper, Scissors (RPS) each player has the strategy set (Rock,Paper,Scissors) to pick from, each of these is a pure strategy so if a player played a single pure strategy e.g rock, they would pick rock with certainty for each play of the game. This would quickly be countered by a pure strategy of their opponent playing paper.

However, a player can assign a probability to each of the pure strategies in their strategy set to create a **mixed strategy**, this generally will out perform the use of one pure strategy [5]. So in the example of RPS a player who plays a mixed strategy of (rock = 0.33, paper=0.34, scissors = 0.33) is less likely to be dominated by their opponent as their moves are not predictable. In a mixed strategy the probabilities of each pure strategy must cumulatively sum to 1.

2.1.3 Game Classifications and Notation

Zero-sum and Non Zero-sum games
A Zero-sum game or constant sum game is one in which the gain in payoff to one play is equal to the loss of the other players in the game [4]. In most strategy games the aim of the game is to win, in the case of two player games, If player 1 wins and is assigned a score of +1, then their single opponent can be assigned a score of -1. If the game is a draw both players score 0, in any eventuality the outcome is zero [13]. For non zero-sum games in the case of our two player scenario the winning of player 1 doesn't necessarily dictate the losing of player 2, players may win at the same time or lose at the same time resulting in scores that can potentially be above or below zero. Games such as roulette fall into the category of non zero-sum games.

Perfect and Imperfect information games
Perfect information games are ones in which all the information of the game state is available to all players of the game. The players do not necessarily know what the other will do, but they are aware of all the previous moves made, the possible moves available to their opponent, and the subsequent states that may occur as a result of each of those moves [12]. In these games each player has exactly the same amount of knowledge as their opponents, games such as Chess, Checker and Go are all examples of Perfect information games. Santorini is an example of a perfect information game.

Imperfect information games are those in which you cannot possibly predict the potential move of an opponent as there is an unknown set of information only your opponent can see or a random quantity that is not revealed until the player makes their move [12]. Such as the roll of the dice or the drawing of a card, games such as backgammon or bridge have imperfect information.

Sequential and Simultaneous Games

Sequential games are those that are played by players executing consecutive turns, a players turn cannot begin until the previous player's turn has ended. Simultaneous games, sometimes known as *matrix games* are those in which players make their move at the same time without communicating, so neither has any information on their opponents move, for example Rock paper scissors, or the *Prisoner dilemma* (without communication) meaning that all simultaneous games are imperfect information games as each player cannot know what the other is going to do [17].

Game representation

When modelling games, the representation of the game state and the strategies available to each player is key in understanding and analysing the game and the players [5]. For non-cooperative games there are two types of representation, the *normal form* - for simultaneous games, and the extensive form - for sequential games. Co-operative game representations have not been considered as they are not pertinent to the project.

The extensive form is a tree representation of a game as can be seen in 2.2, the root node represents the current state of the game and the branches are the decisions available to the active player which result in subsequent states/nodes. Each level represents a players move thus the depth of the tree represents the number of moves in the game. The values at the leaves are called **terminal states**, these occur when there are no more available moves in the game and represent the payoff for that sequence of moves, the extensive form is more informative as it contains the history of moves played in the game as well as the current state, this is not applicable to simultaneous games.

The Normal form or *matrix representation* of a game is a matrix containing all the possible strategy combinations between the all of the players of the game and their respective payoffs [4] see Fig 2.1, the number of players is represented by the dimension of the . In Fig 2.1 if player I selects strategy j and player II selects strategy k then the payoff to player I will be a_{j_k} and payoff to player II -a_{j_k}. This representation makes it clear that a single player's decision is not the only one affecting the outcome of the game [5].

	Player II			
Player I	Strategy 1	Strategy 2	...	Strategy m
Strategy 1	a_{11}	a_{12}	$\cdots$	a_{1m}
Strategy 2	a_{21}	a_{22}		a_{2m}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Strategy n	a_{n1}	a_{n2}	$\cdots$	a_{nm}

Figure 2.1: A matrix or normal form representation of a two person zero-sum simultaneous game. [4]

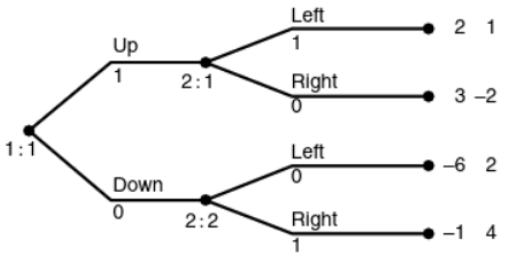

Figure 2.2: An extensive form representation of an abstract sequential game. [4]

One, Two and N player games

In most strategy games the aim is to win, when modelling a game it is simpler to have fewer players in the game, as more parties are involved each can affect the pay off of all of the other players, causing the number of strategies that need to be considered to increase and thus

representing the game can quickly become very complex [17]. Although, one player games are generally not considered by game theory as they usually either have an obvious winning strategy if there are no external factors determining the outcome of the game, or the game is defined by a stochastic event such as in Roulette, there is no certain winning strategy.

Two player *zero-sum* adversarial games are the most straightforward to model as the payoff to one player is the negative pay off of the other, this makes assessing which strategy to pick easier, although strategies may have a poor immediate payoff but have a greater long term payoff for example, trading material for a potentially better board position in chess. This category will be the focus of the proejct as Satorini is a two player zero sum game.

When it comes to N-player games, that is games with three or more players, new difficulties and other considerations arise. The number of combinations of strategies increases and also the potential for *coalitions* (two or more players making decisions together to increase their mutual pay-off at the expense of the others in the game) is introduced [17].

Cooperative and Non-cooperative games
In non-cooperative games each player is making decisions purely for their own payoff and there is no alliance or agreement they can make with another opponent which will mutually benefit them in terms of the goal of the game. This is the case for all two player games in which the aim is to win, and thus defeat the other person.
In a cooperative game any subset of players may form a *coalition*, thus all cooperative games must have three or more players, the coalition then aims to make decisions that will achieve a payoff that benefits each member of the coalition more than they would have been able to acting on their own [4]. Achieving a fair distribution of the total payoff however, is non-trivial as it is difficult to formally define *fair*.

Stochastic and Deterministic games
A game is said to be deterministic if, for a given move in a given state the result will be exactly the same every time that move is executed. Each move in a deterministic game has a pre-determined result for all the possible moves in all of the possible game states [12]. In contrast a game is Stochastic if there exists an element of uncertainty or chance for a given move in a particular state, like the roll of dice or taking a card at random from a deck, this affects the outcome of a given state meaning one state can have multiple different outcomes depending on the stochastic element of the move [12]. Games such as Backgammon and Monopoly are Stochastic games, with Chess and Checkers being the most famous deterministic games, Santorini being a wholly deterministic game.

2.1.4 Game Trees

A game tree is the extensive form (see Chapter 2.1) representation of an entire game with the starting state as the root node, this allows the game to be mapped out into every possible state until it reaches it's *terminal states*, they are then assigned a utility value based on the outcome of the game to the current player, which may be win draw or loss. Game tree's are useful as they can be searched to find the maximum payoff for a player, and by looking at the the sequence of moves inform the player how to play; however, searching the game tree is often not

trivial. The size and therefore search-ability of the game tree depends on two things, the **depth** and the **branching factor**. The depth of the game tree is the number of total number of moves played in the game. The branching factor is the average number of moves available to a player at a state in the game. For Chess the average number of moves per turn is approximately 35 with the number of turns in an average game being approximately 70, Giving a game tree of 70^35 possible states, Santorini has an average game length of 18 moves and each with approximately 55 options giving a game tree of 1.10×10^{69} nodes a game tree of a similar magnitude to Chess, both much too large to brute force search in a reasonable time especially on standard hardware like that available.

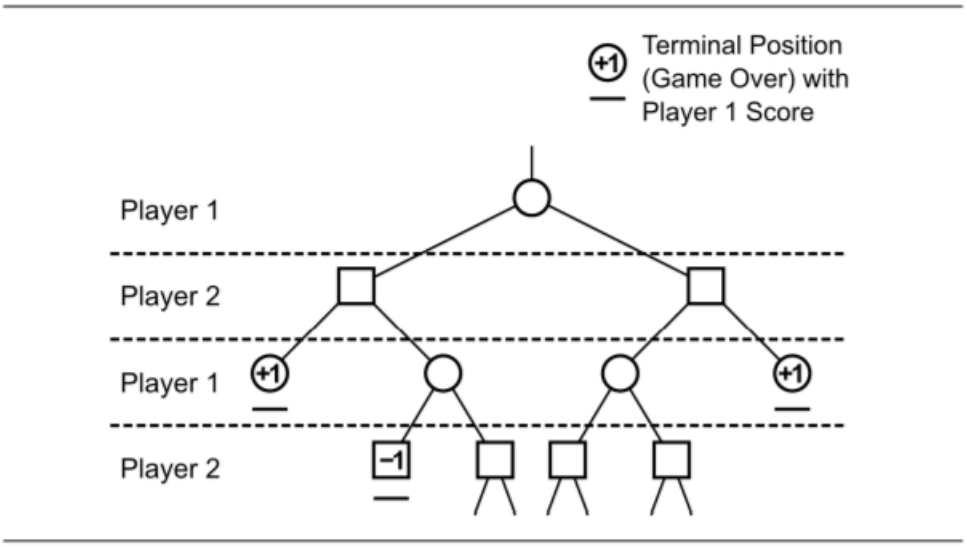

Figure 2.3: A game tree showing players' and terminal moves [12]

2.1.5 'Solved' Games

There are a collection of games have the potential to be *'solved',* this means that for a state in the game given that both of the players are playing perfectly (making the move with the highest possible payoff every time) then the outcome or *value* of the game - win, draw, or loss can be predicted with certainty. These games are those classified as two player, zero-sum perfect information games [8]. There exist three tiers of solution [1, 10]:

- **Strongly solved:** the optimal move can be found from every state in the game.
- **Weakly solved:** the value of the starting state (win, draw or loss) is known as well as a strategy for the first player to realise that state with certainty.
- **Ultra-weakly solved:** the value of the starting state (win draw or loss) is known.

The difficulty of solving a game has previously been tied to the *state-space and game-tree complexity* [19]. However, this has been challenged [8] as there are games such as go-moku and renju with very large state-space and game-tree complexities that have been solved [1]. More recently in 2007 Checkers became the most complex game to be solved after 18 years of analysis and relentless computation by more than 100 computers [14].

2.2 Artificial Intelligence approaches

The following chapter includes research into the potential AI approaches that will be considered for implementation during the project, these methods may have different levels of success depending on the type of game they are applied to. This will include looking at older more "traditional" AI methods for adversarial games such as MiniMax and newer ones such as Monte Carlo tree search.

2.2.1 MiniMax

First outlined by Jon Von Neumann in 1928 the algorithm is applicable to zero-sum, perfect information games. The algorithm is designed to play optimally from a given state provided that the opponent is also playing the best possible move available to them. It does this by recursively minimising the opponents maximum payoff [13]. The game is represented generally in a game tree, as demonstrated in Figure 2.4 detailing a partial game tree for a game of noughts and crosses.
The idea is the algorithm performs a depth first search from the root node – in this case the player playing as crosses (X) to a terminal state. A terminal state is one in which either the player has won the game, lost the game or there are no more available moves so it results in a draw. In a two player zero-sum game like noughts and crosses, these states are evaluated by a *utility function* which could translate to +1 for a winning state, -1 for a losing state and 0 for a drawing terminal state. The intermediate nodes are all the possible interim game states, each consecutive level being a possible move by the opposite player. The utility value of the terminal states is then recursively passed back up the tree with the caller – the root node player selecting the maximum possible outcome and the opponent selecting the minimum. Resulting in a maximum for the root node caller [13]. Minimax will be a good starting point I believe when trying to obtain initial strategies and information on the reduced size implementation of the game, as it will be applicable to the much smaller game tree. The branching factor of the game is too high for the full game implementation.

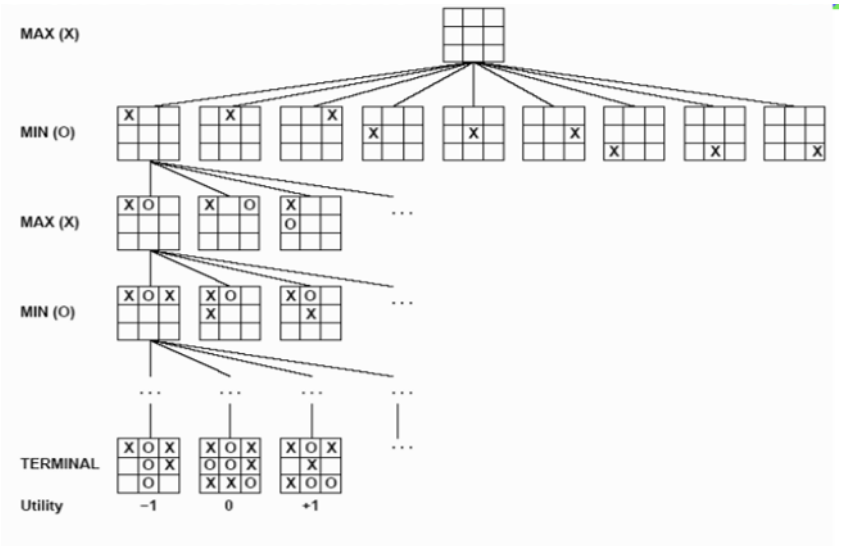

Figure 2.4: A Mini Max partial game tree for a game of noughts and crosses. [9]

2.2.2 Heuristic Evaluation Function.

In a lot of cases performing a full depth search of the game tree is not feasible, it is too large, so performing a full search takes an unreasonable amount of time, even with alpha beta pruning. Instead in most cases it is computationally faster and can be as effective to specify a limit to the depth of the search and evaluate each of the states at that depth with an estimated value of that state's utility [15] this is sometimes known as a *cut-off test*. This estimation is calculated using a **Heuristic evaluation function**; this is a function that cannot say with certainty the utility value of a state but can give a good estimate based on the features of that particular state [13]. A good evaluation function will be able to assign with high accuracy values to intermediate states that translate to their correct utility value and game outcome. An example of how this might be implemented with Minimax is shown in the psuedo code in Figure 2.5.

```
function minimax(board: Board,
                 player: id,
                 maxDepth: int,
                 currentDepth: int) -> (float, Move):
    # Check if we're done recursing.
    if board.isGameOver() or currentDepth == maxDepth:
        return board.evaluate(player), null

    # Otherwise bubble up values from below.
    bestMove: Move = null
    if board.currentPlayer() == player:
        bestScore: float = -INFINITY
    else:
        bestScore: float = INFINITY

    # Go through each move.
    for move in board.getMoves():
        newBoard: Board = board.makeMove(move)

        # Recurse.
        currentScore, currentMove = minimax(
            newBoard, player, maxDepth, currentDepth+1)

        # Update the best score.
        if board.currentPlayer() == player:
            if currentScore > bestScore:
                bestScore = currentScore
                bestMove = move
        else:
            if currentScore < bestScore:
                bestScore = currentScore
                bestMove = move

    # Return the score and the best move.
    return bestScore, bestMove
```

Figure 2.5: Minimax psuedocode with a cut-off test evaluation. [12]

The identification and weighting of those features is what dictates whether or not a function will be a good evaluation function. This may be related to the pieces the each player has on the board or the way their pieces are structured. For instance in Chess many features are considered, each piece is assigned a value pawns being the least valuable the queen being the

most, this is called material value, the difference in material value between two players is one indicator of who might win. However, the structure of the pieces on the board can dictate the true Utility of the state, as there are situations in which someone with less material value may be able to imminently win the game, making certain piece structures a feature of higher importance or weighting. This can be represented as a function known as a linear weighted function as in Figure 2.6, the f_i being a feature of the state and w_i the weighting.

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s) \ ,$$

Figure 2.6: Weighted evaluation function for a state s [13]

Important features are not always obvious and the subtle differences that make similar states have very different utility values is something that is established through thorough studying and analysis of games.

2.2.3 Alpha Beta pruning

Alpha Beta pruning is a method that truncates the game tree by storing a best value for the MAX player so far - *Alpha* and a best value for the MIN player so far - *Beta*. These values are updated as the minimax depth first searches, and if any path has a score that is less than the current score of *Alpha* for the MAX player or *Beta* for the MIN player then the recursive calls are halted and the path is pruned from the search tree [13]. This results in faster more efficient searching of smaller game trees.

2.2.4 Lookup Tables.

One way of improving a heuristic evaluation function is to use a combination of heuristics in conjunction with a database of known advantageous states or a set combination of moves. These are known as *lookup tables* [13]. Some game states often have a vast number of possible moves but there exists a *Policy* which can ensure a win for the current player, these policies are easily executed by computers and can recognise and force wins in certain situations. This method may be applicable to Santorini as there are certain piece configurations that can indicate imminent wins which might be evaluated as insignificant if pieces are considered individually.

2.2.5 Machine Learning

Machine learning is the application of an algorithm that is capable of learning and in some way improving at the task it has been set to do without explicitly being programmed to behave differently. This is a contrasting approach to other techniques in AI in which we aim to program logic and decision making rules which allow agents to play “intelligently”. There are three main types of machine learning algorithms [11]:

Supervised machine learning algorithms – these algorithms are trained on a known, classified data-sets in which the inputs given to the algorithm have a correct output pair. They develop a method based on the classified data-set to analyse new data and select what it interprets to be

an appropriate classification. This classification can then be compared against the correct classification by the algorithm which can then in turn make adjustments accordingly.

Unsupervised algorithms – conversely are trained on data-set in which the data is not classified, these algorithms usually uncover hidden common features from a given data-set. It does not necessarily find a "correct output" but interprets common patterns that may not initially have been easy to spot, and then groups them together showing clusters of similar points. Both of the aforementioned techniques are not suitable for game playing agents as a supervised algorithm would be required to know all the outputs to every move, and a unsupervised algorithm would only group specific moves together without there being any context in the grand scheme of the game – evaluating payoffs.

Reinforcement learning algorithms - lie in between supervised and unsupervised learning, it is a method in which the algorithm searches the state space attempting actions available to it within its specified environment and receives penalties or rewards, but is not instructed explicitly how to improve. Through this it very generally searches possible methods by trial and error which results in it attempting almost all available strategies and finding the ideal one as It continues to learn. This often results in unconventional strategies that have optimal performance, this technique has been hugely successful for other perfect information games such as Go [6].

2.2.6 Montecarlo Tree Search - MCTS

Montecarlo Tree Search in it's pure form is a search method in which the algorithm randomly attempts full random *play-outs* of the game from the available set of moves [12]. The result of these play-outs are recorded and assigned to the initial move they stemmed from essentially discovering the evaluation function as it plays. MCTS is an iterative search technique that improves with time, as with more plays of the game the more statistical data the algorithm has on each move it has previously tried. From this it can give a better estimate if that move is likely to yield a win based on the win/loss record of previous plays.

This approach requires no knowledge of the game other than the rules, which means it is a viable technique for games in which knowledge acquisition is not easy. It has also been proven to be effective in games with large search spaces and high branching factors[3]. This does not come without an overhead though, as there is no predefined knowledge required, for MCTS to be successful it obviously has to perform an extremely intensive repeated search process. There also are MCTS variations such as MCTS-minimax hybrid algorithms which use a shallow depth minimax searches giving more informed starting point to the random search procedure [3]. And more topically MCTS with deep neural networks which has proven unparalleled in the game of Go [16].

Using this research paired with a detailed look into the game of Santorini and it's rules in the next chapter, the most appropriate technique will be selected. Keeping in my the limited resources that will be available to me during this project.

Chapter 3

Santorini

For this project It was decided to that the Strategy board game Santorini would be a good candidate for exploration. It is a two to four player game; the game has a basic form and can also be played with the addition of 'Gods' which enforce extra rules that alter the potential set of moves for each player. However, these rules can quickly imbalance the game making the prospect of developing individual AI strategies for each God pairings an endeavour that was beyond the reasonable expectation of this project. For this reason and for relative simplicity within the scope of this project I will be looking at implementing the two player basic version of the game. The reason the game was selected is that the rules are simple, yet the game requires strategic consideration, each player has two builders which are aiming to ascend to the 3rd level of a building and are only capable of making one universal move, the limited set of variables means that developing a strategy is not immediately obvious, from extensive playing of the game it seems there are certain strategies that can be employed which have specific counter strategies, this makes the strategy of the game rapidly evolve turn by turn. This led to the question, would a minimax implementation with a general heuristic state evaluation be a sufficient AI or would a more advanced technique be required to create an 'Intelligent player'.

3.1 Setup and Rules

Santorini is a strategy game which is played on a 5x5 board, each player has two pieces called "builders", who's allegiance is denoted by their colour - two blue and two red. The aim of the game is to manoeuvre and build with your builders to be the first person to ascend with their builder to a level 3 building. A player also loses if they are unable to move when it is their turn an Example game state can be seen in 3.1.

Figure 3.1: Example late stage of a game. [7]

Before the game can begin, the Blue player places both of their builders onto the board, then the Red player responds placing their pieces on two unoccupied spaces on the board. Blue then moves first. Each move consists of two parts:

- Part 1 - A builder may move one square in any direction, provided the space is not occupied by another builder and they adhere to the movement rules.*
- Part 2 - Place a building level on an adjacent unoccupied space (including diagonal)

*A player may use a builder to build on any level but can only move to a space on the same level or to a level of one above their current level – I,e 0 to 1, 1 to 2, or 2 to 3. A builder may move down any number of levels. I.e 2 to 0. Players may also build "Domes" which can be placed on a level 3 building in order to stop an opponent from ascending to level 3, essentially removing that grid position from the game [2].

3.2 Strategy

Firstly, selection of piece positions on the empty board, there are a few variations here and it's not very clear if any give an obvious advantage see 3.2. Being within the central 9 squares gives most versatility allowing you to react to opponents moves quickly but also move across the board to build an attack. In contrast the corners and walls can be used as an effective means of blocking, stopping the opponent from getting close enough to block.

Figure 3.2: Example opening positioning of pieces. [7]

As the game is quite open and straight forward this means strategies are constantly evolving so it is rare that a player employs one strategy for a game and then successfully executes it. This means strategies have to be re-evaluated after every move, as momentum in the game can change very quickly.
Strategies Include:

- Gain and hold height: climbing up levels when possible is obviously key but also making sure not to step down unless absolutely necessary, height is easily lost and hard to regain.
- Keep the opponent down: keeping the opponent on the ground by building multiple level 2 buildings.
- Trapping an opposing builder: block an opponents builder in a corner or against the side of the board, surrounded by buildings 2 levels higher than them or by buildings with Domes on them, this removes them from the game leaving a 2 vs 1 see.

- Using a builder as a blocker: using the builder to occupy a space the opponent needs to block you is a strong tactic.
- Edges and corners: using the edges and corners of the board to reduce the number of ways opponents can approach to block.

3.3 Classification

Using the terminology researched and defined in section 2.1.3, Santorini can be classified as a game with the following characteristics:

- **Perfect information**: All information in the game is available to both players as is the history of their moves. Each player is aware of all the possible moves their builder can make next.
- **Two Player**: The game can be played with up to four players, but this is actually two teams of two so the most commonly played version and the one we are analysing for this project is the two player version.
- **Non-Cooperative**: The game is played by two players both aiming to win, there is no cooperation.
- **Sequential**: Moves are taken consecutively, each player has to wait for their opponents turn to end before theirs begins.
- **Deterministic**: All moves in the game are defined and set out and their availability is only influenced by previous moves made by players in the game. There is no element of chance or randomness in the game.
- **Zero-sum**: , The outcome of the game or total payoff is always 0, if we score +1 and -1 for a loss. In Santorini there is always a winner and a loser so the total payoff is always 0.

3.4 Approach

From the research conducted in Chapter 2 along with the classification of the game as a result of that research It seemed suitable to pursue an AI implementation based upon *minimax* in conjunction with a *heuristic state evaluation function* for *game trees* of a limited depth. Ideally the AI will be able to look several moves into the future and give an accurate estimation as to which of the states will likely lead to a winning terminal state within a reasonable execution time enabling it to select the best move possible at a given state of the game. Another option could have been to attempt an AI implementation that incorporated *machine learning*, something like **MCTS** or a reward based *reinforcement learning algorithm* which would learn to play optimally over time without much alteration. However, the feeling was that MCTS would not be feasible with the hardware available and a machine learning approach would be too much of a "black box", which would give less insight into the strategies and intricacies of Santorini, the idea being that developing the heuristic evaluation function would reveal key features of the game that need to be considered to play well.

The heuristic evaluation function will consist of a linear weighted equation of the form discussed in section 2.2.2. This will be a collection of game features and corresponding weights, as the material in Santorini is equal and usually doesn't change for both players, the features will most likely be a selection of positioning characteristics of the builders and buildings. Players can In theory block in their opponent as in 3.3 however, this very rarely occurs unless the opponent is either inexperienced or has made a blunder. Features to be investigated and considered are likely; the height of the pieces and the height of the opponents pieces, looking for states that facilitate elevating pieces along with states that reduce the mobility of the opponent, and perhaps more specific situations such as forks for winning states - much like in chess, or opportunities to deny a player from being able to execute a winning move.

Figure 3.3: Example of blocking an opponent piece creating a 2v1 builder advantage. [7]

A potential issue with this method is that I am no expert in the game of Santorini, and although I have played the game quite extensively, a lot of the move selection feels quite intuitive rather than methodical, meaning formalising promising features so far hasn't been straightforward. To try and identify some of these features as well as test minimax on it's own before adding state evaluation, a reduced board size version of Santorini will be implemented - a 2x2 board with only one builder. This is in the pursuit of finding terminal states which might have distinguishable characteristics that could be useful as features in the heuristic state evaluation function.

But, before the heuristic evaluation function and minimax can be developed firstly a simple AI implementation will be required to assure the game can successfully be played by a Human against the AI. This will be a kind of *Primitive strategy* and the most basic decision making an agent can make is picking randomly, it follows to improve this to a simple *greedy strategy* one that picks a move based on a single criteria or heuristic. This will most likely be selecting the move that gains the most height for a player. Then once a playable primitive AI has been achieved the more appropriate minimax with heuristic evaluation function or *advanced strategy* can be implemented. The function will be iteratively designed adding and combining distinct sets of features with varying weights. Another consideration will be how far ahead the algorithm will be able to look - the max depth of the tree and how it will be used and potentially changed given the context of the game - early, middle or late game. Then, playing the different iterations of the AIs against one another, this will hopefully identify the most

effective strategy.

Finally the AI will be tested, initially this was supposed to be against a human novice player, but in light of the current circumstances (see appendix B COVID), testing will instead be carried out by playing the AI against the Novice AI computer difficulty in the Mobile application version of Santorini [7]. Although not ideal this should help ascertain the degree of success of the project.

Chapter 4

Game Design and Implementation

This chapter aims to detail the design choices that have been made in creating the human playable Santorini game program as well as the AI, the reasoning for these choices and to what degree they have been successful.

4.1 Design and Language selection

The first choice to be made was which language would be most appropriate for the implementation of the Game. From simple analysis of the game and looking at the moves available to each player it became quickly obvious that Santorini has a huge branching factor with each move for a player having 40-70 possible distinct choices. With a typical game lasting around 18 turns this would yield an estimated game tree size of approximately $55^{18} = 1.01 \times 10^{69}$) nodes. Considering the size of the game tree a language such as C or C++ may have been a faster option in terms of processing speed which would have been desirable for the computation of the AI algorithms, especially creation and search of the game tree. But with low level languages such as C and C++ memory has to be manually allocated and collected, for a program that would be using complex data structures including recursive trees this was be an avoidable source of errors, ones which would no doubt consume already limited time. For this reason I turned my search to higher level languages like Python and Java both of which automatically allocate and collect memory.

From a paradigm perspective, the game features and rules as outlined in Chapter 3 made it a fairly straightforward decision to use an Object Orientated approach, as the components of the game are structured and interconnected in a way which nicely lends itself to object orientation rather than a procedural programming style. Finally I settled on Java as it had automated memory handling, it enforces object orientation, and is has extensive documentation and online support unlike newer alternatives such as Kotlin. The consideration of picking Java over a language such as Python was ultimately down to my experience programming in Java is greater than in Python, this decision was made to reduce the number of new technologies and concepts being learned as the project would provide enough difficulty without unnecessary additional potential sticking points.

4.2 Program Structure

As was previously touched upon the game consists of several main physical components those mentioned in Chapter 3; the board, the builders, and the buildings, these would form the fundamental basis of my program. Along with these would need to be an implementation of two players and an Instance of the Game to hold all of the components and the appropriate methods for them to interact. This lead me to the high level structure of there being a *SantoriniGame* class, which would consist of two *Players* (also a class) as well as the game

board which would be a structure made up of instances of a *Tile* class. Each player would have two builder objects with which they would share a colour representing their Alliance, a red team and a blue team. Both teams will enter starting locations for their pieces on the board via co-ordinates in the form (X,Y) and the game will commence. The game will then follow a simple loop pattern of, player enters the desired builder, then enters the desired location move, check move validity - throwing error if invalid, update builder location, player then enters desired build location which is checked and executed, then play is passed to the other player. Once a move is made the game is checked to see if the move just played was a winning one. Each class will contain the relevant methods for pieces to move and interact within accordance of the game rules and perimeter of the board.

I first decided to abstract the board into the idea of it being a 5x5x4 structure, that consists of 4 levels 0-3 each of a 5x5 tile grid. So the board is a 3-D array of Tile objects, of the form Tile[4][5][5]. Each Tile has three fundamental properties, represented by boolean variables. 1. isOccupiedWithBuilding, 2. isOccupiedWithRedBuilder, 3.isOccupiedWithBlueBuilder. When the board is initialised all properties of all tiles are set to false. This structure was chosen as it encodes the location of the players pieces into the board which means almost all of the required information for the evaluation of states is stored in the board, reference to the builderPieces is not required. On top of this it also offers a simple solution to the printing of the board state through the use of 3 nested for loops. This approach may have been over-complicated as in hindsight it may have been simpler to implement the board as a 2-D structure with an associated number for the level at which the occupying feature is at, this may have been able to use less memory as the 3-D data structure will have incurred a potentially avoidable memory overhead.

It is in the Player and BuilderPiece classes that the methods required to handle geographic and building moves are implemented. Each BuilderPiece object has a set of possible moves that correspond to the points of a compass, N, NE, E, SE, S, SW, W, NW, these are then filtered down to legal moves, if a player enters an illegal move an Exception is thrown. A player then enters the pair of compass points for the move and build, the board is updated, the game then checks if the move has won the game for the current player terminating if it has, if not play passed to the opponent. This process was adjusted for the AI so each move-build pair was treated as a single entity for ease of use with minimax.

4.3 User Interface and Game Representation.

The User interface design for Santorini is quite straightforward as the board and pieces are not too complex to represent. The game is played via a basic text based command line interface, with each build level of the game represented by a 5x5 grid of empty tiles represented by empty brackets [ ], when a player places their BuilderPiece on a tile the bracket becomes filled with their initial e.g [J] and if they place a building it is the tile becomes a [+] indicating the player should look up to the next level to see the tile availability. To avoid forcing the player to make moves with (X,Y) co-ordinates on a zero indexed board which could easily cause confusion, the points of the compass seemed like a more intuitive option as each part of the move consists of

selecting an immediately adjacent tile. As mentioned in the previous section this means that provided it is known which player's turn all the information that is needed to evaluate the game state is the game Board.

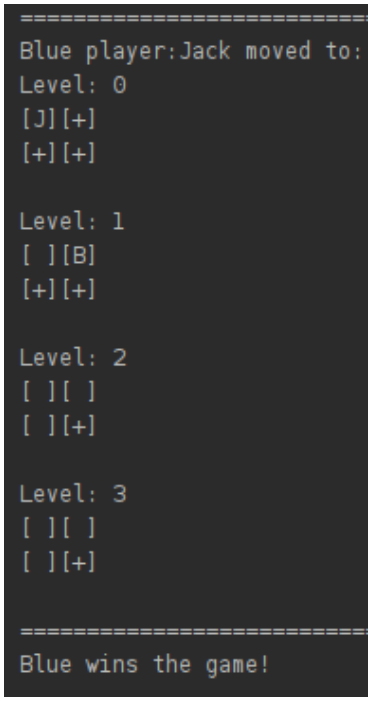

Figure 4.1: A 2x2 game text representation.

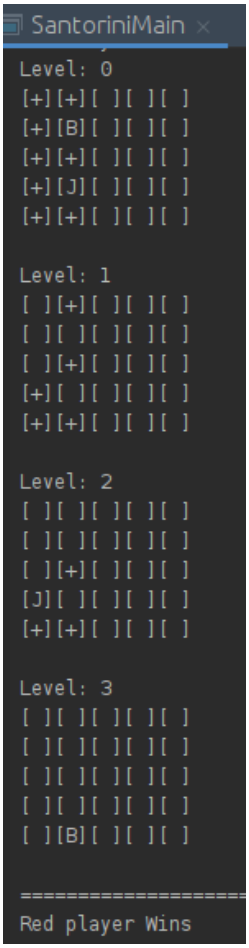

Figure 4.2: A 5x5 game text representation.

The display of the board state is not as humanly intuitive as the 3D animated interface shown in Chapter 3 but, for the purpose of this project to have prioritised creating a detailed graphical interface like that would have been a mismanagement of the time available to me. As the priority and focus of the project is analysis and development of the AI the basic text representation was completely sufficient.

Chapter 5

Developing the Artificial Player

Once the game was fully functional and playable by two humans, work could commence on creating an AI, starting at the most basic goal - making a legal move given a state, then step by step improving it to reach a more complex AI with improved performance. This chapter outlines the incremental progress of that process.

5.1 Finding and representing the available moves

Before any kind of move selection could be implemented, first I needed to calculate all of the possible move combinations for both of the AI player's pieces, each having up to 8 possible geographical moves and each of those new locations on average having 3-6 possible build locations. This at the beginning of the game being as large as 70 unique combinations of geographical moves followed by a build. Rather than represent each as a tuple of moves, for example (Move - NE, Build - W) as a human player would make them in the game, it made more sense to create an *ArrayList* of final board states that could be reached by executing each of the possible move combinations from the current state. This is achieved in a function called getStates which takes the current board state and the active player creates Deep copies of each and then creates and returns an ArrayList of 3D Tile arrays (board states). It is from this list that a decision can be made as to which state the AI decides to move to. The actual game board is then updated to the board state picked from the possible list. As the moves are made on copies of the board, the position of the AI player's builder pieces are then retrospectively updated from the new board state.

5.2 Primitive AI implementation

These are the most simple pure strategies possible to play Santorini, they consider making only the next move with no consideration of opponents moves or future benefit to the player. They are only intended as a starting point which should expose flaws in simple strategy, confirming the need for more detailed consideration if we are to achieve a decent standard of AI.

Random Selection

The most basic approach to decision making is to make an arbitrary or random selection from a collection of available possible options. It made sense that this would be the starting point for the AI's move selection process. In terms of Santorini this simply means picking one game state from the list of calculated next possible game states. This obviously does not factor in any calculation of payoff to either player, so the chances of the AI making a string of decisions that actually might result in them consecutively gaining height with a builder to level 3 very unlikely. Although not very useful for the primary decision making method, random move selection is a helpful tool when deciding between moves that result in an equal payoff and must be considered during the testing phase to ensure that when playing many repetitions of games

between two AIs, in the case of states at a certain point in the game being evaluated to the same payoff the AI does not select the first one every time which would result in the same deterministic game being repeated. In practice, random selection performed very poorly unsurprisingly and is definitely not a good option for a whole game strategy but, it has valid use and should not be entirely disregarded.

Greedy - highest level

One step further than just randomly picking the next move is picking the move based on obtaining the highest immediate payoff based on a single heuristic, for Santorini where getting to level 3 is the goal, this would most naively be considered moving up to a higher level. This however, would quickly be confronted with a set of states in which there is not option to move up higher and once again the decision of which state to chose would be left to chance. The greedy decision AI outperformed the random decision AI when played against each other but very inefficiently, with the games taking far more turns than an average game.

The exploration of these rudimentary strategies highlight the need for many aspects of the game state to be considered if the AI is to make a well informed move, not limited to the features of the AI's pieces but also considering how the opponent might be looking to play as a rational player also aiming to maximise their possible payoff and win the game.

5.3 Advanced AI implementation

For the AI to progress to any decent standard of play it would need to consider far more than just the best heuristic to pick the next immediate move. This meant identifying key features of the game state which indicate a players weakness or dominance in a given position and formalising these in the evaluation function, then using this in conjunction with minimax to make an intelligent move. Once a collection of features had been refined. The process of varying the weights of each component of the function to try and identify an optimal evaluation function could begin.

5.3.1 MiniMax

The backbone and main focus for the advanced strategies as previously discussed is the minimax function, at a high level pure minimax takes the current board state and the player trying to maximise their payoff and from performs a depth first search of the whole game tree passing back up the utility values from each of the terminal states of the tree. This is unfeasible for the full game of Santorini due to the previously mentioned high branching factor. Instead, minimax was first implemented on a reduced size version a 2x2 board with one builder, and then once functioning it was adapted to include the required cut-off testing with a depth limit that was needed for it to be usable in the full version of the game. The initial algorithm was implemented based on the research conducted in Chapter 2, see Figure 2.5, it takes the maximising player and the current board state, creating a list of possible moves from that state (as outlined in section 5.1), performs each of the moves and recursively calls minimax on each of the resulting states until the terminal states are reached. It then feeds the utility value back

up the tree to the original call by the current player and returns a move that that would lead to the highest utility value for the maximising player.
For the full board implementation the additional information of the maximum depth needed to be passed, this is the depth which once reached would cause the heuristic function to be called and return the integer value estimation by the heuristic function of the current state, this value is then passed back up to the caller instead of the utility value. The move with the maximum payoff can then be selected from all of the states passed back. For this to be possible an initial feasible Depth limit needed to be established so it could be decided when to perform the cut-off test and apply the heuristic function. To begin with the heuristic function was just set to return an arbitrary value and tested at each depth to see how long the minimax algorithm would take to return. The results can be seen in table 5.1.

Depth of Tree	Number of Nodes	Time to search (ms)
0	70	19
1	2898	383
2	101430	4561
3	3448620	170270
4	1.52×10^8	>2 Hours

Table 5.1: A table with the explosion of nodes at each depth of the game tree from the starting position of the game.

If testing two implementations of minimax using different heuristic functions against each other the AIs need to be able to make move choices within a reasonable time. For an average game for 2 AIs playing with Depth limit 2 the game takes 2-4 minutes, If only one AI is then set to depth limit 3 and look one move further, each game then can take up to 30 minutes. Given that each strategy will be played against each of the others 100 times, the time implication of depth limit 3 make testing infeasible so the initial depth limit was selected to be 2.

5.3.2 Reduced Board MiniMax

Initially the thought process was that if I implemented minimax on a reduced board size version of the game, the game tree would be small enough to execute an entire tree search producing a collection of terminal states from which certain features of the winning player's board structure may be extracted and adapted for the features of the heuristic state evaluation function. Although the algorithm successfully reached the terminal states for the smaller boards, as can be seen in Figure 4.1 in Section 4.3 the game most often ended in a win by stopping the opponent from being able to make a move. The same outcome was found from playing multiple games on a 3x3 board with one builder also, the board was just too small. Upon closer analysis it was then realised that for the 2x2 board the game was at least *weakly solved* as the player who moves first is guaranteed a win.

This proved to be a frustrating conclusion that offered little insight into developing a successful heuristic evaluation function, as in the 5x5 board full implementation with two builders each it is almost impossible to defeat an opponent by blocking them from being able to move, the number of options for moving is simply too great.

5.3.3 Feature Identification

Due to the failings of the reduced board size experiment the features would have to be identified and defined by extensive playing and studying of the game. Using the tactics discussed in section 3.2 a list of positional characteristics that a player might consider when deciding on the best move were collected and formalised. One thing that was kept in mind was applying the *lookup tables* theory discussed in section 2.2.4 to Santorini, as there are certain specific situations which have far greater payoff given a certain context (Feature 4 below), by encoding this it means although the depth limit is limited to 2 it essentially gives minimax an extra turns look ahead, without incurring the enormous overhead cost of calculating another whole level of the game tree. Once finalised, these aimed to provide insight to the quality of the board state:

Feature 1: Combined piece height difference
This is the combined levels at which a players two pieces currently sit, minus the combined levels of the opponents pieces.

Feature 2: Player vertical mobility
This is the number of adjacent tiles to the current player which they can move to, with tiles at a +1 level being more desirable, and tiles of +2 or +3 level which cannot be moved on to being undesirable. The opponent's vertical mobility is subtracted from the current player's vertical mobility.

Feature 3: Centre square control
This is whether or not the current player has a builder on the centre square.

Feature 4: Level 2 threat
This is the number of adjacent +1 level tiles to a player's builder if that builder sits at level 2 as it potentially signifies an imminent win. This needs to be considered separately to vertical mobility as as level 2 it has far greater importance. Especially if there are two or more tiles, this is a fork, only one can be capped resulting in a certain win for the current player. These are positively scored for the current player and equally negatively scored for the opponent.

This produces the final heuristic function:

$$h(s) = w_1 \times HeightDiff + w_2 \times VerticalMob + w_3 \times Centre + w_4 \times lvl2Threat$$

5.3.4 Heuristic state evaluation Function

For the heuristic function to be successful and give an accurate evaluation of a non-terminal state it needs to value the game features to a fine enough granularity that the function would return more than just one or two distinct sets of evaluated states so it is able to differentiate between subtly better or worse positions. For this to be possible an appropriate range for the score function and the features outlined in the previous section that make it up had to be set out. It was decided that +1000 would be value of a winning terminal state -1000 for a losing one. Then the individual feature point assignment could be calculated with this in mind.

Feature 1: HeightDiff - This heuristic looks at the all of the builder's positions in the game, assigning a value of 0 points if the builder is at level 0, 40 points if at level 1, 60 points if at

level 2 and 1000 point for a state with the builder at level 3 with it being a terminal state. The combined value of the current player's builders minus the value of the opponents builders is returned.

Feature 2: VerticalMobility - This heuristic looks at each builder and all of their adjacent tiles. It assigns +5 points for a tile +1 level relative to the builder, -10 if +2 levels relative to the builder and -15 for +3 and -20 points. This score is calculated for all builders and the combined score of the opponents score is taken from the AI's combined score. An example of differing Vertical mobilities is demonstrated in states 3 and 4 of figure 5.1

Feature 3: CentreSq - This heuristic places a +10 point score for states with a current player's builder on the centre square. As can be seen in state 5 of figure 5.1.

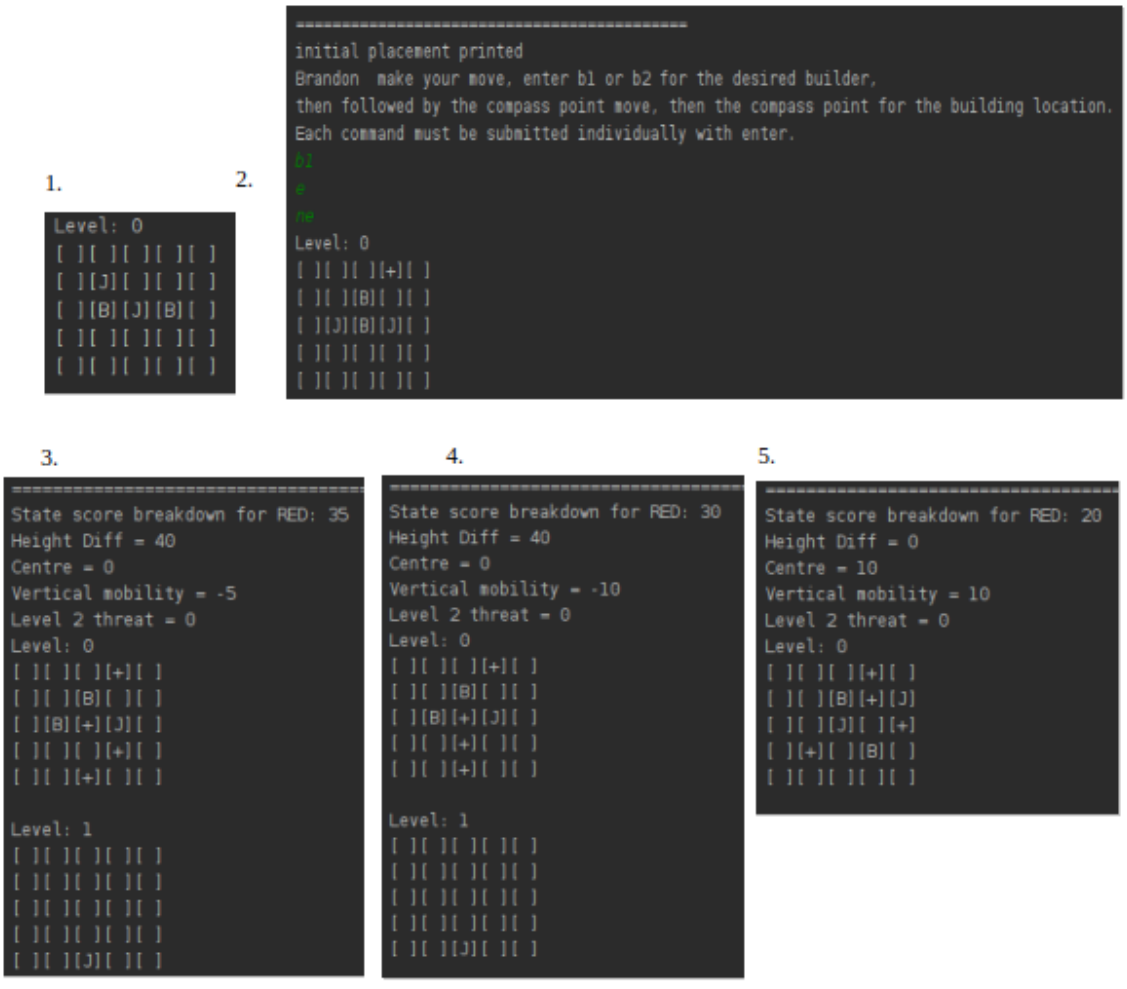

Figure 5.1: 5 board-states, the potential future positions of a starting state with the the score breakdown of each state. B is the blue player and J is the Red player.

Feature 4: Lvl2Threat - This heuristic evaluates a position with two or more +1 level tiles adjacent to a builder which is already on a level 2 tile with +500 points as this will yield a certain win. An example board state is shown in figure 5.2.

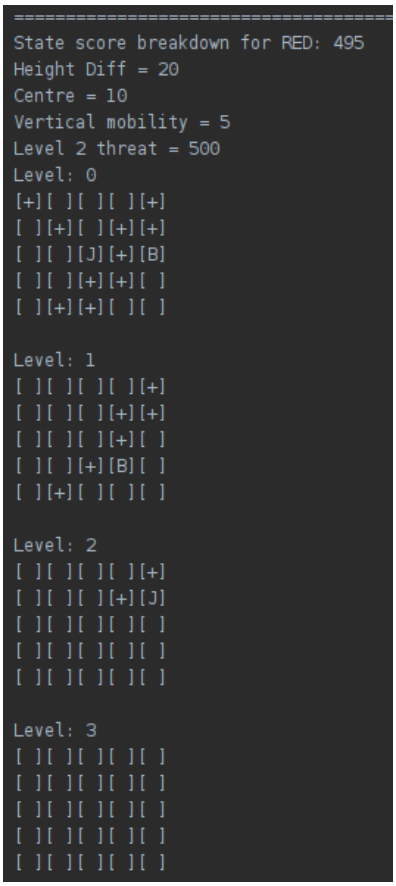

Figure 5.2: Example of feature 4, a fork which will result in a certain win for Red (J)

It was noted that these point assignments already encode an inherent weighting to each of the features, this was unavoidable as they were assigned based on personal judgement with not much statistical data to support them. They were roughly tested so the AI made moves as expected. For example as can be seen in states 3 and 4 of figure 5.1 if only scored on height difference, both would be evenly scored forcing minimax to make a random selection between them; instead the vertical mobility feature differentiates them scoring state 3 as a better position as hoped due to the fact the opponent "B" has fewer options to ascend.

Another consideration is the weightings need to account for relative scores between two move choices for example a builder would be expected to move up a level rather than surround themselves with lots of +1 levels. They would need to be surrounded by 8 +1 Squares before it becomes a random decision between moving up a level and taking that surrounded position. That position is very unlikely to occur so the AI will almost always move up a level. However, any misjudged discrepancy should be made negligible in the testing phase of the heuristics by varying the weights.

5.4 Summary

After initial use of the heuristic function, it was clear that a function that was capable of making *rational* decisions when playing Santorini had been achieved. When only the HeightDiff heuristic was used with minimax it was beating the Greedy basic AI comfortably. All of the

features could now be tested against each other to obtain evidence to confirm the most successful set of heuristics for the AI. Once the best combination is determined a range of weights will be tested to find the optimal values in the final Heuristic function, this final function will be ready to be tested against the official mobile application AI to ascertain the quality of the final AI implementation.

Chapter 6

Testing and Evaluation

Having all of the features necessary to create the final AI, it was essential to test each of the individual components as they were added to interpret the affect they were having on the quality of the final AI. Starting with the most basic informed AI - the greedy decision making, each subsequent iteration would then be tested against a control AI, seeing if it achieves a dominant win rate. Once the best collection of heuristic features are empirically justified, the weightings of each individual heuristic will be adjusted to find the optimal combination of weights. Once this is established the final AI can then be tested against the mobile application AIs as a replacement test for the human testing phase.

6.1 Heuristic parameter Testing

The testing aimed to evaluate whether or not each modification made to the AI's Heuristic function was positively contributing to it's overall quality. As there is no point scoring in Santorini the method for evaluation is limited to the win percentage an AI iteration achieves. Each of the following combinations has been tested by playing 100 games, the number of moves of each game is also recorded for an added aspect of analysis aiming to provide a little more context to the games. Although a win in less moves is not necessarily a better win, in terms of the AI it does indicate a potentially higher quality of move selection.

Before the testing could begin, a neutral starting configuration of Builder pieces needed to be selected so it is ensured that neither player has a clear advantage. To identify this I started a game between two instances of the "Godlike" AI of the mobile game and recorded the positions they assigned for player 1 and player 2, this was then kept the same for every play see Figure in section . This definitely risked the potential for similar games to occur however, the size of the branching factor meant that there is extremely high variation from the outset. The starting player blue was always be using the AI with fewer Heuristics in it's evaluation function. In each of the following tests the weights of the heuristics were introduced at a value of 1.0.

Test 1 - Greedy Basic AI vs Height Difference Heuristic
This test was 100 plays of the Greedy Basic decision making AI against minimax with depth limit 2 using only the heightDiff (Feature 1) heuristic in the heuristic evaluation function.

$$h(s) = 1.0 \times HeightDiff \text{ vs Greedy Height gain decision making}$$

The outcome was a win rate of 100% in favour of the minimax heuristic AI with an average game length of 27 moves. Although the minimax AI completely dominates the Greedy AI as expected, the length of the game being almost 1.5 times the average game length indicates that it's move selection is far from optimal.

Test 1 proved that the Greedy basic strategy was completely dominated by the most basic heuristic when paired with minimax, this indicated that basic strategies without consideration

for the opponents moves are completely insufficient as an AI strategy in Santorini. For tests after Test 1 the Height Difference heuristic with minimax depth limit 2 - HDH ($h(s) = 1.0 \times HeightDiff$) will act as the control AI for the introduction of each additional feature. HDH is the only available option as the control as none of the other features encourage *rational* decision making on their own in terms of payoff towards achieving the main goal of Santorini.

Test 2 - Height Difference Heuristic vs HeightDiff + VerticalMobility
This test was 100 plays of HDH AI against minimax with depth limit 2 using the heightDiff (Feature 1) heuristic with VerticalMob (Feature 2) in the heuristic evaluation function.

$$\text{HDH vs } h(s) = 1.0 \times HeightDiff + 1.0 \times VerticalMob$$

The outcome was a win rate of 96% in favour of the minimax heuristic AI with an average game length of 16 moves. This was a strong indicator that Vertical mobility plays a valuable part in improving decision making by the AI.

Test 3 - Height Difference Heuristic vs HeightDiff + Centre This test was 100 plays of HDH AI against minimax with depth limit 2 using the heightDiff (Feature 1) heuristic with Centre (Feature 3) in the heuristic evaluation function.

$$\text{HDH vs } h(s) = 1.0 \times HeightDiff + 1.0 \times Centre$$

The outcome was a win rate of 96% in favour of the minimax heuristic AI with an average game length of 17 moves. This was an unexpected result and shows the centre heuristic is significant in improving the decision making of the AI.

Test 4 - Height Difference Heuristic vs HeightDiff + lvl2Threat
This test was 100 plays of HDH AI against minimax with depth limit 2 using the heightDiff (Feature 1) heuristic with lvl2Threat (Feature 4) in the heuristic evaluation function.

$$\text{HDH vs } h(s) = 1.0 \times HeightDiff + 1.0 \times lvl2Threat$$

The outcome was a win rate of 90% in favour of the minimax heuristic AI with an average game length of 17 moves.

Test 5 - Height Difference Heuristic vs HeightDiff + VerticalMob + Centre +lvl2Threat
This test was 100 plays of HDH AI against minimax with depth limit 2 using all of the features together in the heuristic evaluation function.

$$\text{HDH vs } h(s) = 1.0 \times HeightDiff + 1.0 \times VerticalMob + 1.0 \times Centre + 1.0 \times lvl2Threat$$

The outcome was a win rate of 98% in favour of the minimax heuristic AI with an average game length of 18 moves.This makes sense and proves the optimal collection of heuristics is all of them together.

The results of the Heuristic parameter tests against the control AI all established that each of the features improved the decision making of the AI. This was the expected outcome of the

tests as each feature was picked to improve the AI; however, the 96% win rate with the addition of the centre heuristic was higher than expected. This was most likely due to the fact it has a significant affect on evaluating states at the beginning of the game before it became out-weighted; which meant the AI prioritised it in the opening few moves and this set it up to win the game. In contrast the lower than expected win rate of 90% for the level2Threat feature is indicative of the fact the opening selected is less strategic which allows the opponent AI to get ahead early and go on to win even though it could identify the opponent was going to win. A potential limitation of these tests were that there was no available external control AI to bench mark each of the iterations against, they had to be tested against a stripped down version of themselves. So unless features that scored bad states highly were added, the tests were always going to yield clear wins for the heuristic function with more features. Ideally there would have been a way of automating the play between my AI and the mobile game AI to see how each iteration performed against the most basic mobile AI.

6.1.1 Heuristic weight refinement

With the optimal collection of Heuristics for the final AI justified, it remained to establish if the inherent weighting assigned through scoring each feature was accurate or there was a more optimal set of weights that could achieve a higher win percentage or an equal win percentage with a lower number of average moves over 100 games against the control AI. To establish this each weight would be altered and the outcome of wins/losses and moves per game over 100 games would be recorded for each feature weighting. Feature 4 - level2Threat was kept at a weight of 1.0 as it had already been weighted accordingly and if adjusted it could contribute evaluating a state over 1000 points which would exceed the value of a winning state, causing the AI to pick a fork over a win.

The following table contains the weights for:

$$h(s) = w_1 \times HeightDiff + w_2 \times VerticalMob + w_3 \times Centre + w_4 \times lvl2Threat$$

With $w_3 = 1.0$ and $w_4 = 1.0$.

	$w_2 = 1.0$	$w_2 = 1.2$	$w_2 = 1.4$	$w_2 = 1.6$	$w_2 = 1.8$
$w_1 = 1.0$	98/2 (18)	95/5 (20)	90/10 (18)	91/9 (19)	89/11 (19)
$w_1 = 1.2$	94/6 (18)	xxx	93/7 (17)	92/8 (19)	90/10 (18)
$w_1 = 1.4$	93/7 (19)	91/9 (18)	xxx	93/7(17)	88/12 (19)
$w_1 = 1.6$	95/8 (18)	94/6 (17)	95/5 (19)	xxx	91/9 (18)
$w_1 = 1.8$	97/3 (17)	95/5 (17)	90/10 (18)	96/4 (17)	xxx

Table 6.1: Win/loss results for each pair of feature weights for f_1 and f_2 with $w_3 = 1.0$ and $w_4 = 1.0$

From the results in table 6.1 it can be clearly seen that altering the weighting of the heuristics feature 1 and feature 2 even slightly causes a drop in win percentages. Testing all of these again with varying w_3 values is unlikely to produce an outcome which yields any higher than a 98% win rate or a lower average move count. It was deemed this would be a waste of time as each 100 games takes over an hour to run. So tuning commenced with features 1,2,4 fixed at 1.0 and feature 3 altered.

$W_3 = 2.0$

$$h(s) = 1.0 \times HeightDiff + 1.0 \times VerticalMob + 2.0 \times Centre + 1.0 \times lvl2Threat$$

This yielded results of 92% win in an average game length of 17 moves.

$W_3 = 1.5$

$$h(s) = 1.0 \times HeightDiff + 1.0 \times VerticalMob + 1.5 \times Centre + 1.0 \times lvl2Threat$$

This yielded results of 99% win in an average game length of 17 moves. Values of 1.0-1.5 with graduations of 0.1 were tried for w_3, none of which improved on this win rate. From this testing it has been established that the optimal weightings for the Final heuristic evaluation function for the project AI are:

$$w_1 = 1.0, w_2 = 1.0 w_3 = 1.5, w_4 = 1.0$$

6.2 Final Project AI vs Mobile AI Test

Unable to test the final AI against a Human player as first hoped (see appendix B), it would instead have to be tested against the AI on the mobile application. For this I acted as a go between the two machines inputting the moves between my program and the application. My AI would play the lowest difficulty of the mobile AI in a best of 5 games, then if it won it would play the next lowest difficulty. The mobile AIs are categorised as "Novice", "Modest", "Skilled", "Expert" and "Godlike". The wins and the number of moves in each game will be recorded.

Project final AI vs Mobile "Novice" AI

Game	Moves in Game	Win/Loss for project AI
1	15	Loss
2	20	Win
3	19	Loss
4	16	Win
5	14	Win

Table 6.2: Win/loss results final AI vs Novice mobile AI

Project final AI vs Mobile "Modest" AI

Game	Moves in Game	Win/Loss for project AI
1	16	Win
2	17	Loss
3	26	Win
4	19	Loss
5	13	Loss

Table 6.3: Win/loss results final Project AI vs Modest mobile AI

The above two tables 6.2 and 6.3 show the final project AI to have achieved a 60% win rate against the "novice" mobile AI and 40% against the "modest" AI. These results to some degree validate the fact that an AI capable of beating a human player has been achieved as these AIs will not have been implemented to just act as irrational agents for people playing the game. A

potential flaw in this testing system is that in some circumstances the mobile AIs makes moves that appear un-human, in game 3 especially vs the modest AI the opponent made a string of moves that seemed potentially irrational in terms of the goal of Santorini. It should be noted that the mobile AI most likely uses a similar AI implementation with a set of heuristics. The wins here could just mean that the project AI is specifically effective against the mobile AI but not necessarily against human players.

Chapter 7

Conclusion

The final chapter aims to provide insight into the success of the project with regard to the aims and objectives set out in Chapter 1. It also looks at the specific achievements and challenges encountered in the project process, while reflecting on my own personal experience and lessons learned.

7.1 Objectives and Aims

For this project to be a success, the fundamental objectives and aims set out at the beginning needed to be attempted and achieved. These are reviewed and evaluated:

1. **Perform in depth background research of Games and Game AI**

   The literature review in Chapter 2 explores a range of topics including the necessary Game Theory to understand games formally. Along with this was a range of relevant AI techniques that have been used for similar games. This materials relevance was then analysed with Santorini in mind in chapter 3.

2. **Develop a playable version of Santorini**

   As detailed in Chapter 4 a simple playable implementation of the Santorini was created from scratch, an object orientated solution with a basic text user interface. After the human playable version was functioning the implementation was adapted and extended to be utilised with minimax and heuristic state evaluation.

3. **Implement and test a reduced size instance of the game for analysis**

   In Chapter 5 section 5.3.2 the reduced board size version of Santorini was implemented for analysis; however, this proved was a fruitless experiment. In terms of the objective though it was successful and was an example of methodical work.

4. **Develop an agent that can intelligently play Santorini**

   Chapter 5 documents a methodical and detailed approach to studying and implementing the AI techniques researched in Chapter 2 resulting in heuristic evaluation function, comprising of several different heuristics. Which when used together with minimax proved to be quite effective as an AI player.

5. **Improve the AI to be able to beat a novice player of the game**

   After the best combination of heuristics were justified and refined in section 6.1 they were played and tested against the Mobile AI in section 6.2. As can be seen in the results the final project AI this objective was achieved.

The main objective of this project was to create an artificial player using suitable AI techniques that was capable of playing the game Santorini to the standard of a novice human player. This

objective was successfully met and validated as the submitted program has an AI that has proven that it can beat not only the most basic but also the modest AI implementation in the mobile app version of the game. This proves the main objective has been met. Something I would take from this project is that the necessary techniques required to create these types of game playing AI depend on the Standard of the AI you are aiming to create, this project has proven that established "traditional" techniques when appropriate can yield quite successful results without a huge amount of resources. And though new AI techniques are effective, they are not essential.

7.2 Reflection

To conclude I would say this project has been a success, it is by no means is without it's shortcomings but, overall the Aims of the project have been accomplished. I think one part of this project which was lacking was the quality of the evaluation of the final AI, I struggled to find a quantitative measure by which I could grade my AI, which meant my test results in section 6.1 did not have a relevant graphical representation and were not as insightful as I would have hoped.

One major challenge of the project would be the developing of the playable implementation of Santorini had to be completed before minimax could be attempted which meant the design choices made in the implementing of the game weren't necessarily the most efficient and easiest to adapt when it came to creating a functioning minimax. In particular the creation of the next possible states due to the fact the builder moves are split into 2 parts. The solution was using board states as the result of each move combination which then meant re-allocating builder locations using the state rather than the other way round which was initially implemented. This is an inherent difficulty of working on a project in which you are learning lots of new concepts and theories as you are trying to implement them as you do not always interpret them perfectly straight away. This results in cyclical periods of testing, programming and re-evaluating implementation design. This is a difficult process to assign time to because there is no way of accounting for issues and bugs which don't become obvious until late into the development phase, furthermore when working entirely alone you create a knowledge silo and it can be difficult not having someone to discuss problems with which I think results in an overall longer but probably more rewarding process.

Working on this project has been a truly new and informative learning experience but most notably a challenge, something from the outset I was extremely daunted by. Now at the end, I am pleased to say I am proud of what I have accomplished during the project. From this project I have gained a new appreciation for the importance of in depth planning and research when undertaking a project of this size and detail, as good preparation inevitably saves time and stress in the long run, you might even say it has a greater long-term payoff.

7.3 Future Work

7.3.1 Improving the current solution

One way in which the current AI could be made more efficient and quickly improved rather than consider a completely different approach would be to implement alpha beta pruning with minimax as it would result in a much more efficient tree search and permit the AI to look further into the future of the game.
Further research into lookup tables or designing a lookup table for certain scenarios would also benefit the heuristic function as when the Level 2 threat heuristic was added to the evaluation function the AI was greatly improved. The addition of other such heuristics would no doubt improve the AI.
Another improvement would be to implement a machine learning, neural network to find the exact optimal setting for the weights of the individual heuristics, then the AI would be able to learn and improve as it played more games.

7.3.2 Alternative Solutions

As reasearched in sections 2.2.5 and 2.2.6 an entirely alternative method could be implemented, applicable methods would be something like Monte Carlo Tree Search or a Reinforcement learning method.

7.4 Legal, ethical, social and professional issues

This project was focused around implementing public existing AI techniques that have been extensively studied before. I created the game from scratch using no external Java libraries and as it is for academic research purposes only I do not think it is infringing on any copyright laws from the perspective of the game publisher. The data and information gathered with respect to the game is in no way sensitive and could not be utilised in a malicious manner that I am aware of. Initially there was to be a human testing section but this was replaced with a test against another AI, removing any issues with regard to anonymity or the storing of personal data of test users. Meaning overall, there are no major legal, ethical, social or professional issues within this project in my view. The debate of the evolution of AI and it's ethical and social implications is far beyond the kind explored in this project.

References

[1] L. V. Allis et al. *Searching for solutions in games and artificial intelligence.* Ponsen & Looijen Wageningen, 1994.

[2] N. Author. The rules of santorini, No Date. https://www.boardspace.net/santorini/english/santorini-rules.html.

[3] H. Baier and M. H. Winands. Mcts-minimax hybrids with state evaluations. *Journal of Artificial Intelligence Research*, 62, 2018.

[4] E. N. E. N. Barron. *Game theory an introduction.* Wiley series in operations research and management science. John Wiley |& Sons, Inc., Hoboken, N.J, 2nd ed. edition, 2013.

[5] I. K. Geckil. *Applied game theory and strategic behavior.* CRC Press, Boca Raton, 2009.

[6] E. Gibney. Google ai algorithm masters ancient game of go. nature 529, 445–446, 2016. https://doi.org/10.1038/529445a.

[7] G. Hamilton and Roxley-Games. Santorini, mobile game, 2019. Available on Google Play Store.

[8] M. J. Heule and L. J. Rothkrantz. Solving games: Dependence of applicable solving procedures. *Science of Computer Programming*, 67(1):105–124, 2007.

[9] K. Kask. Set 4: Game-playing., 2016. http://www.ics.uci.edu/ kkask/Fall-2016%20CS271/slides/04-games.pdf, Last accessed on 2020-1-3.

[10] T. R. Lincke. *Exploring the computational limits of large exhaustive search problems.* PhD thesis, ETH Zurich, 2002.

[11] S. Marsland. *Machine learning: an algorithmic perspective.* CRC press, 2015.

[12] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.

[13] S. J. S. J. Russell. *Artificial intelligence : a modern approach.* Pearson Education, Upper Saddle River, N.J. ;, third edition. edition, 2009.

[14] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[15] C. E. Shannon. Programming a computer for playing chess, 1988.

[16] D. Silver and D. Hassibas. Alpghago: Starting from scratch deepmind blog, 2017. https://deepmind.com/blog/article/alphago-zero-starting-scratch.

[17] P. D. Straffin. *Game theory and strategy.* Anneli Lax New Mathematical Library ; 36. Mathematical Association of America, Washington, DC, 1993.

[18] T. Turocy and B. von Stengel. Game theory cdam research report london school of economics, 2001. http://www.cdam.lse.ac.uk/Reports/Files/cdam-2001-09.pdf, Last accessed on 2020-1-5.

[19] H. J. Van Den Herik, J. W. Uiterwijk, and J. Van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.

Appendices

Appendix A

External Material

All source code submitted was written by me using no external libraries. Source code is available at the gitHub URL: https://github.com/j-boreham/santorini-project

Appendix B

COVID-19 Imapact Statement

The change in circumstances due to the COVID-19 pandemic not only directly hindered my project as I was not able to complete the final evaluation of Human testing that I had initially hoped but also definitely affected my ability to work on the project. Without the human testing the evaluation process seemed a bit hollow, which definitely impacted my motivation when this was realised. The work environment and resources available to me at home have not been ideal, especially in comparison to the library. I think the project on the whole has suffered significantly from the circumstances and I feel this project had more potential that I was unable to fully maximise, It has been a frustrating conclusion to the end of my time at Leeds.