School of Computing
FACULTY OF ENGINEERING

Exploring Competitive Artificial Intelligence in Strategy Games: Samurai

Jake Scaife

Submitted in accordance with the requirements for the degree of
BSc Computer Science (Artificial Intelligence)

2017/2018

40 Credits

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Project Report	PDF	Minerva (02/05/18)
Project Report	Physical Copy (2x)	SSO (02/05/18)
Software	GitLab Repository	Supervisor, Assessor (02/05/18)

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.

(Signature of Student) _____

Summary

This report describes the process taken during my final year project to design, implement and improve an artificial player capable of playing the board game Samurai. The artificial player will be created with limited intelligence at first and over the course of this project, I will explore different methods of allowing the artificial player to learn or apply more complex strategies to maximise how competitive it can be when playing against a human player.

Samurai offers a number of challenges, due to the complexity of the game and the large number of unique game states, which make the game interesting to study. Due to these challenges, a portion of this report is dedicated to exploring the representation of Samurai in software: such as enforcing rules and storing game states. From there, we will investigate iterative improvements of heuristic state evaluation in an attempt to provide the best possible set of heuristics which an artificial player can use to produce the most intelligent move possible, from any position within the game. These final set of heuristics hope to provide an artificial player the ability of defeating a human player and could potentially be adapted to games outside of the scope of this project.

Acknowledgements

I would like to thank Dr. Brandon Bennett for supervising my project, for giving me this opportunity to explore an area of Computer Science that I am passionate about and for pointing me in the right direction.

Contents

Chapter 1

Introduction

Strategy games and competitive games in general offer a number of challenges to artificial intelligence that can also be seen in a variety of different fields, as well as our everyday lives. Even before the introduction of high-performance computing, numerous notable attempts, both successful and unsuccessful, were made to develop computer programs which were capable of imitating a human player at competitive strategy games, such as Chess[12]. Since then, a wide array of classic board games have been replicated digitally with the intention of constructing intelligent algorithms that are able to defeat their human counterparts. Until relatively recently, it has been a major challenge for computer scientists to develop artificial players for strategic board games due to major limitations in hardware available to them. However, since some of the classic board games such as Chess, Checkers and Go have been considered 'solved' or have artificial players capable of beating the best human players, there has been an increase in research into new ways of constructing artificial players as well as adapting the existing tried and tested techniques for new and more complex competitive games.

1.1 Report Outline

This report is split into six main chapters which are further split into relevent subsections. Each chapter describes a part of the iterative process I will carry out over the timespan of this project.

1. Chapter one is an introduction to the project and will describe the problem that this project wishes to solve. The chapter will also describe the minimum deliverables that this project aims to achieve and the methodology that I will use to successfully achieve those objectives.

2. Chapter two describes the background research conducted during this project in an attempt to ensure the project is as successful is meeting it's objectives as possible. Due to the exploratory nature of the project the background research will be limited. However, some research will be done into existing game theory and AI techniques as well as the game which is the focus of this project.

3. In chapter three I will go over the implementation of the game in software. Details in this chapter include the technologies used to represent the game as well as the graphical interface created.

4. Chapter four covers the creation of artificial players and the artificial intelligence techniques explored when constructing them. This section will discuss state representation, move choice and strategies.

5. The testing phase of the project will be discussed in chapter five. This chapter will go into detail regarding how the representations were tested and how the artificial players

were improved based upon previous results.

6. The final chapter, chapter 6, will provide a conclusion to the report, detailing how successful the project was at meeting it's original aims and objectives. It will also include a personal reflection and any ideas of future projects expanding on this one.

1.2 The Problem

The scope of this project is to recreate the 1998 strategy board game Samurai digitally, allowing for the exploration of developing an artificial player capable of imitating or even surpassing a human player. Although relatively simple for a human player to understand the rules of the game and develop simple strategies, Samurai is difficult to master due to the complexity of the game, the difficult to track win conditions and the large number of options available to the player. These attributes make Samurai a particularly interesting strategy game to study as an artificial player with even limited intelligence could surprise a human player with unexpected plays and allows for a variety of techniques to be applied and tested when constructing the artificial player.

Despite Samurai winning multiple awards and having a passionate following, no notable attempts have been made since the game's conception to recreate the game digitally; I have also been unable to find any research that has been conducted into constructing artificial players for this game. For this reason, I hope to contribute some research over the span of this project into both representing the game digitally in an efficient way and also constructing artificial players that are competent at playing the game against a human player.

1.3 Project Aim

The aim of this project is to contribute meaningful research into the digital representation of the board game Samurai and the construction of an artificial player which can make intelligent moves equal to, or above, the skill level of a typical human player. During this project I will investigate tried and tested methods of constructing advanced artificial players in well known board games and attempt to implement and evaluate those methods for the game Samurai.

The objectives for this project are as follows:

- Construct a software representation of the board game Samurai, allowing for a full round to be played as per the rules. (Defined in section 2.3)
- Construct a graphical representation of the game, allowing a user to see a play-by-play of a round of Samurai.
- Construct a simple artificial player capable of playing a round of Samurai from start to finish.
- Investigate ways of improving the artificial player to increase it's chance of victory against a human player, or less advanced artificial player, using artificial intelligence techniques.

1.4 Methodology

When constructing exploratory software it is difficult to predict the amount of time a certain task within the project will require and thus makes allocating time somewhat challenging. To make time allocation easier, I decided to adapt an iterative approach, starting with a minimal, functional piece of software and continuously improving upon it. This reduces the risk of not completing the project objectives (Section 1.3) as a functional deliverable is produced as soon as possible, with later improvements only enhancing functionality rather than implementing. Considering this, I split the project into six parts, the first part being the report deliverable, four parts corresponding to each major objective and a final part allowing for additional testing and enhancements beyond the minimum deliverables. In theory, this would allow the report to be worked on continuously as the project progressed, the main objectives met and then additional contingency time available at the end of the project for further AI improvements or quality of life features.

Due to the exploratory nature of the project, the software deliverable will be created with the consideration that it is for use as a research tool only and not as a commercial tool. For this reason, the amount of time allocated to testing, refactoring and user experience required for a commercial tool will not be allocated for this project and the final software will only be improved in ways which enhance the research potential of the software.

The Gantt chart displayed below (Figure 1.1) illustrates how I plan to allocate time across the project. Due to my commitments to other modules, the majority of the project will take part during the second semester where the workload is not as heavy. The first half will be dedicated to background research and report preparation, with the second half being more focused on developing the software. Each square in the graph represents a week of time dedicated to the project between November 2017 and May 2018, with weeks including Christmas and exam periods not displayed.

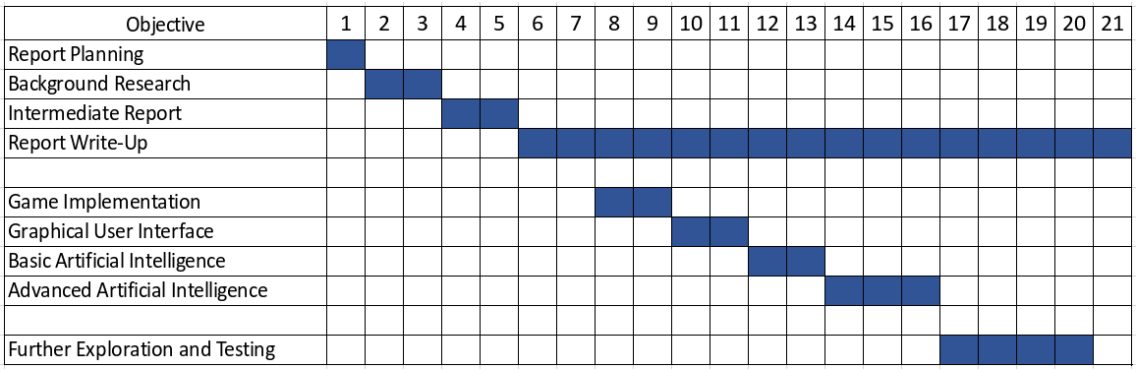

Figure 1.1: Project time allocation

Chapter 2

Background Research

The challenges faced when constructing artificial players are prominent in a number of fields that are essential to making progress in our every day lives. Due to the huge number of applications, both artificial intelligence and game theory have been the subject of a wide array of exploratory research which could be invaluable in deciding the most practical, but also meaningful, approach to meeting the objectives of this project. Despite this project concerning exploratory software, it is important to first build a knowledge base that we can expand on throughout this project so that we can maximise the impact and reduce time spend on redundant research. For this reason, in this chapter we will discuss some of the existing literature regarding game theory, artificial intelligence and also build an understanding of Samurai so that we can approach solving the objectives of this project in the most informed way possible.

2.1 Game Theory

The outcome of conflict between two decision making intelligent agents, whether acting cooperatively to achieve the same outcome or competitively to achieve different outcomes, is an essential study in a number of disciplines; including economics[3], politics[11] and biology[**?**], as well as it's precedence in strategy games and computer science[6]. Game theory details the mathematical models which can be used to represent these conflicts, allowing an actor to predict or observe potential outcomes of said conflicts depending on the decisions taken by an agent within the conflict. These models can be used to generate optimal strategies for particular conflicts assuming that agents will act rationally in an attempt to gain the best possible outcome.

The conflicts modelled by game theory come in a wide range of types and representations that can be adapted to different situations. We can explore these models and representations in an attempt to find the best possible model to use when constructing an artificial agent for Samurai.

2.1.1 Terminology

Until this point, we have used general terms to describe components that make up a typical strategic game. However, when discussing game theory, it is important to understand that models studied in game theory can be used in situations where ambiguous terminology could cause confusion. Thus, in order to discuss game theory and remove any possible ambiguity, we must define exactly what a game is, as well as the components that make up a game, within the scope of this report. Although a number of definitions exist, we will use the following terminology[8].

First of all, it is important to state that the word *game* used in the context of *game theory*

references a situation where a set of independent agents agree to carry out moves based upon an agreed upon rule set. Simply put, a *game* is described by a set of rules.

When the game is *played*, or a *play* is performed, we are referring to a single instance of the game. A *play* is a series of moves which follow the rules of the game until an ending or winning condition is encountered; the *play* is complete.

A *state* within a play describes the current allocation of game resources during a play. Each *state* is unique to a game but a *state* can be repeated multiple times throughout a play.

A play consists of a series of *moves*, each of which describe the action performed that results in a transition of states. At each state, a decision, or many decisions, are made which result in a state transition. These decisions are referred to as a *move*. Each move produces an *outcome*, which is typically in the form of another state. The *payoff* of a move refers to the value of the outcome produced by a move.

A *strategy* details which move should be made depending on the current state. A winning *strategy* typically aims to maximise the payoff of a series of moves.

Finally, a *player* is an agent involved in a play which makes decisions on which move to make at any particular state within the play; in other words, a *player* enacts a strategy. We assume that a *player* would act predictably with the goal of maximising their individual payoff.

2.1.2 Game Types

Game theory details a range of ways to categorise games which allows us to build models which suit the characteristics of the game in question. Being able to define what *type* of game we are dealing with, allows us to better suit the strategies we apply to the game and determine the intelligence approach we use for constructing an artificial player. We will discuss some of the game types that are common in game theory that may be applicable to this project.

Competitive

Competitive games consist of multiple players, each competing for a favourable outcome for themselves or for their team. This can be in the form of defeating another player or gaining a higher payoff than another player. A game can be both competitive and cooperative if the game allows players to work together to defeat a common adversary and reach a win condition without having to betray the cooperative partner[2].

Perfect Information

In order for a game to have perfect information, a player must have knowledge of the current state of the play, the current state of each of the players and a knowledge of all previous moves made throughout the play. If this is not possible, a game has Imperfect Information[5]. A typical game with perfect information includes Chess, as all states are visible from the game

board. A game with imperfect information includes poker, as a player does not have knowledge of another players hand.

Simultaneous and Sequential

Two characteristics a majority of games possess is having either simultaneous or sequential move order[1]. If a game has sequential move order, players take turns making moves, allowing a player to make decisions based on an opponents previous moves. If a game has simultaneous move order, all players moves are applied at the same time, meaning a player has no knowledge of the move an opponent is going to make.

Zero-Sum

In many competitive games, the players are competing for an objective or other goal that has to be won over or taken from another player. If when gaining an objective advantage over a player, the other player receives a disadvantage of the same size, then the game can be considered zero-sum. For example in Chess if a piece is captured, the capturer gains an advantage of that piece whereas the captured gains a disadvantage of the same piece[7].

Many-Player

Almost all board games have a finite number of players and can be considered many-player or n-player games. The number of players can range from single player (1 players) to any finite number of players (n players)[9]. Competitive games are typically modelled as two player games, regardless of the number of players. Games consisting of many players can be simplified into two groups: allies and enemies, or players with complimentary goals and players with adversary goals.

2.2 Artificial Intelligence

Artificial Intelligence has been used to solve a variety of different games with varying level of complexity using a variety of different techniques. Here we will explore some of the common artificial intelligence and machine learning techniques and discuss whether or not they are appropriate for the project.

2.2.1 Heuristic Evaluation

Heuristic evaluation is a commonly used technique in calculating the payoff of moves within games. A heuristic function sums weighted observations effecting the current state of a play and uses those observations to predict the value of a move. This allows an artificial player to choose the move with the highest possible payoff that can be made from the current state. Heuristic evaluation is popular in complex games where the state space is large and a game tree would require massive amounts of space, focusing on current observations allows the values to be calculated quickly but at the cost of less complex strategy.

This method is a sensible choice when considering the game Samurai. Due to having imperfect information, it makes sense to determine the best possible move based upon the current state of the game rather than making assumptions about unpredictable opponents. Additionally, the large number of combinations of hands that can be placed in a large number of playable locations makes the number of unique states incredibly high, making it slow and inefficient to build game trees or state-action values for use in minimax and reinforcement learning.

For this method to work, we would have to perform tests on different heuristics and the weighted values placed on those functions, to determine the best possible strategy given any board state. Alternatively, we could use hyper-heuristics as detailed in section 2.2.4.

2.2.2 Minimax

Minimax is popular in two-player games with relatively low number of possible moves in each state. The process involves maximising the possible payoff from a sequential game by selecting the best possible move and assuming an opponent will also make their best possible move. By limiting the moves available to an opponent, we can minimise the loss from a worst case game[10].

Minimax could potentially be valuable when improving the intelligence of an artificial player in Samurai. The value of each move can be calculated using a heuristic function and the best possible payoff selected from those moves. The problem with this approach is that Samurai is an imperfect information game and the minimax function would not have knowledge of the other players hand and thus would have to predict which move an opponent would play. The damage limitation would be minimal due to this lack of knowledge. This significantly reduces the effectiveness of the minimax function and would likely see little-to-no improvement in intelligence from heuristics alone with an increase in computation time and problem complexity.

2.2.3 Reinforcement Learning

Rather than providing the artificial player with a series of rules (heuristics) it can use to determine the value of a particular move, it is possible to allow the artificial player to learn their own rules based upon a predefined set of wanted or unwanted outcomes[4] By allowing the artificial player to add or remove weight from a particular move based upon whether or not it lead to a favourable outcome, it is able, over a large number of plays, to construct an optimal strategy (often referred to as a policy) for winning the game.

The difficulty with this method is that the player would need to store values for making each possible move in each possible state, otherwise known as a state-action function. Due to the large number of unique states in Samurai and the large number of possible actions within those states, simply the storage required to store this function, the computational time required to search it and the number of plays required to populate it would be unreasonable without significant simplification of the game. One simplification could involve ignoring the moves made by an opponent and only assigning values to placing a particular tile at a certain location. This

however would be an extremely naive approach and would likely result in worse results than logic based intelligence.

2.2.4 Hyper-Heuristics

One way that reinforcement learning could be applied to the problem is by instead attempting to learn the best combination of heuristics for a particular state or the entire play rather than for each move. This would massively reduce the space and computational time required for learning while also potentially improving the effect of our heuristic function by optimising the payoff of different heuristic weighting. Using this method would allow the artificial player to modify the heuristic functions based upon their success each time a play is made, rather than manually setting the heuristics and running tests.

This technique could be very valuable to improving the artificial player and will likely be explored during the 'further exploration' aspect of the project, to further refine a competent artificial player.

2.3 Samurai

There are a number of reasons why Samurai is the topic of this project, including the complexity of the game and lack of current research into the game. Before we can begin development of the game, we must explore the rules of Samurai and to what extent this project will implement the game.

2.3.1 Setup

A play of Samurai is performed on a provided game board, which holds the perceivable current state of the game. The game board consists of a varying number of hexagonal tiles of which the majority are playable locations and the remaining locations are objective cities. The number of hexagonal locations depends on the number of participating players; a minimum of two players and a maximum of four players, with the board size increasing with the number of players.

At the start of the play, there are three main processes before a move can be made. The first process includes distributing objective pieces among the cities on the game board. In a two-player version of Samurai, 7 of each piece: Buddha, Helmets and Rice, are spread among the cities. A city can contain either 1, 2 or 3 pieces depending on it's size and may not contain duplicate pieces. These pieces can either be sequentially distributed by the players, one at a time, or can be allocated at random.

The second process is providing each of the players with twenty playable tiles, the tiles include:

- 1x 2-Buddha
- 1x 3-Buddha
- 1x 4-Buddha

- 1x 2-Helmet
- 1x 3-Helmet
- 1x 4-Helmet
- 1x 2-Rice
- 1x 3-Rice
- 1x 4-Rice
- 2x 1-Samurai
- 2x 2-Samurai
- 1x 3-Samurai
- 2x 1-Ship
- 1x 2-Ship
- 1x Ronin
- 1x Tile Exchange
- 1x Piece Exchange

These tiles make up the player's deck. A player's deck is private and can not be seen by other players. An explanation of the tiles can be found in section 2.3.2.

Finally, five of the tiles in each players deck are removed and placed into the players hand. This can be done randomly or the player can be allowed to choose depending on the rules in play. The tiles that make up the players hand are the only tiles that are available to be placed on the game board, tiles in the deck are not playable. The hand is also kept secret from other players.

2.3.2 Rules

A player is chosen at random to make the first move, this player can place any of the tiles in their hand on an unoccupied playable location on the game board. If the tile is a ship type, the tile must be placed on water. There are two types of tiles that a player can place: slow tiles and fast tiles. Fast tiles include the tile exchange, piece exchange, ronin and ships. All the other tiles are slow tiles. The player can place a maximum of one tile per move and a minimum of zero tiles per move. Placing a slow tile on the board adds one to the total tiles placed in the move, placing a fast tile does not increase the total tiles placed. Thus, a maximum of one slow tile can be placed and any number of fast tiles can be placed.

Tile exchange tiles can only be placed in playable locations where a tile belonging to the same player has already been placed. The player then selects a second unoccupied location. This move places the tile exchange tile in the location which was previously occupied and moves the tile that previously occupied the location to a new location.

Piece exchange tiles can not be placed on playable tiles but allow the player to switch the pieces which occupy two cities. For example a Buddha piece in city one could be switched with a Rice piece in city two. The piece exchange can only be applied to cities which are not already won and cannot move pieces into cities where there would be duplicate pieces in a single city.

Each tile has an associated influence which it applies to city pieces that are adjacent to the location a tile is placed in. For example a 3-Rice tile applies 0 influence to adjacent Buddha pieces, 0 influence to adjacent helmet pieces and 3 influence to adjacent rice pieces. When all of the adjacent playable land tiles surrounding a city are occupied, the city is considered surrounded and the player with the most influence applied to a piece within that city wins that piece. The player who wins the piece does not have to be the player who placed the tile to surround the city.

Once a player has placed the tiles they wish to use in a move, the same number of tiles the player removed from their hand are replenished from the players deck. The next player can then make their move continuing from the previous player.

2.3.3 Win Conditions

A play of Samurai ends when all of the cities have their adjacent land tiles occupied. When this happens, there are no more pieces to win and thus no further purpose in continuing the play.

A winner is calculated by the amount of pieces they have captured from cities on the game board. If a player has won more of a specific piece than any other player, they have won the support of the caste represented by that piece. If a player wins the support of two castes, they automatically win the play. If players capture the same number of pieces of the same type, the support for that caste is tied and not won by any player. The player who has won the support of the most castes wins the play. If two or more players have won the same number of castes, the winner is the player who has won the most pieces which are not pieces representing the caste that they won the support of. If this number is still equal, a winner can be found from the total number of pieces captured.

2.3.4 Project Scope

To be able to dedicate more project time and research into building an artificial player for Samurai, the scope of this project will not cover all different combinations of playing Samurai. There are number of different game modes available, starting conditions, win conditions and variable number of players available to choose from that would make the scope of the project far too wide. Numerous approaches to artificial intelligence would need to be considered for the different game modes and testing the skill of artificial players across all of the game modes would require a significant amount of testing to confirm any results. Additionally, the implementation time would be significantly increased for the game representation, meaning less time spent exploring artificial intelligence.

With this in mind, I have chosen to base the Samurai implementation on a two player version

of the game. This massively decreases the number of unique states and removes a large portion of the unpredictability present in the moves of opponents: an artificial player would only have to counter the moves of one player rather than three. In chapter six I discuss the possibility of future work and would be interested in seeing further research into Samurai with additional players with the hope that findings in this report would scale.

Additionally, the game setup covered in this project will be distributed at random rather than allowing players to distribute pieces and select hands. By making this random, it reduces the complexity of the problem without causing adverse effects to the intelligence of an artificial player. If the player is able to place pieces, it is likely to result in common predictable game starts and if a player is able to select their hand, it is likely that starting hands of opponents will be predictable. For these reasons, artificial players created with random game starts will be at minimal disadvantage when playing with a chosen game start and justifies the huge reduction in implementation complexity.

Finally the game mode that I will implement in this project is domination game mode. The two modes available, classic and domination, both have similar rules and win conditions but play out slightly differently. Domination is by far the most popular game mode and the more interesting to explore.

2.4 Summary

From the background research discussed within this chapter, it is possible to describe Samurai as a competitive, sequential game with imperfect information. The game is modelled as a 2-player, n-player game that is zero-sum. For the purpose of this project, we will explore the use of heuristic evaluation for the basis of the artificial player with the possibility of exploration into hyper-heuristics.

- Samurai is competitive: Players compete to capture the most city pieces in order to gain the support of castes.
- Samurai is sequential: Players take turn making their moves, with the opponents previous moves visible to them.
- Samurai has imperfect information: Players do not know the current state of another players hand.
- Samurai is zero-sum: A piece captured by one player translates into an exact disadvantage to the remaining players.

Chapter 3

Game Implementation

The first half of this project concerns designing and implementing the game of Samurai digitally, allowing a full play of the game to be performed as set out by the rules in section 2.3.2. In addition to this, a graphical interface should be created to allow the play to be seen move-by-move by the user. These major implementation aspects have to be performed before any research can be conducted into building an artificial player.

3.1 Language

The first choice when considering implementing the game in software is what language and technologies are available to solve the problem. Due to the nature of the project, the weight placed on technology choice is not as high as if the end software was required to be of commercial standard, but still needs to be considered to ensure the implementation is suitable for research.

Due to the fact we will be working with user interfaces, it seemed sensible to consider object oriented languages, giving the added benefit of modular and abstract code. C++ is a good choice due to it's heavily documented Qt libraries for constructing graphical user interfaces and the low level nature of the language allows for efficient implementation, potentially allowing for faster or more efficient artificial player testing. Java is also a good choice, a compiled language with a number of options for constructing graphical user interfaces and a strong focus on object oriented design. The language I will be using for this project however is python, particularly python 3. Python 3 allows for object oriented approach with good support for graphical user interfaces. In this case, PyQt5 is likely the best option for the graphical user interface due to it's extensive documentation and operating system native design. The main advantage of python is it's simplistic syntax, allowing for less time spent on the implementation aspect of the software and more time spent on the actual research. Java is substantially more verbose and C++ is a much more complex language; if the project was designed for commercial standard I would more likely use Java with JavaFX instead.

3.2 Class Design

Graphical user interfaces and games in general are very good candidates for object oriented design due to their object based modular nature rather than procedural. For this reason, I will detail some of the classes that will be required to build our application so that a full play of Samurai can be performed.

The first and most important class to consider is a main Game class. This class will handle the game setup, the main loop, and the checking of win conditions throughout a play of the game. This class is responsible for keeping track of the state of the game, including the playable tiles,

cities and the players involved in the game. Some of the main methods in the Game class include updating the status of cities, checking if they are surrounded and allocating pieces to a player who have won them. The class should contain a main game loop, which loops until a win condition is met, allowing players to place tiles on the game board sequentially. This of course means the class also needs a method for checking win conditions, this would check the state of the game and determine is the game should end, if so a winner is found. The Game class should keep instances of each Player so that it can communicate with them as well as the game board, or representations of the game board so that it can communicate with it. A player is a part of a game. A game board is part of a game.

Another major class is the Player, this can be an abstract class which needs to be specifically implemented/extended by other players, but holds the main common functionality of a player. This class should setup the deck and hand of the player, with a method to ensure the hand is always supplied with tiles from the deck. The player should keep track of its score and influence so that it can make plays based upon the current state of the game. The player should also store an instance of the game so that it can fetch information about the game and also communicate directly with the game. The player should have a method which can be called by the main game loop which returns the move that the player wishes to make. For this, the player should have a method for getting the possible actions in a state depending on their current hand tiles. It should also have a method for choosing which of these actions should be chosen, depending on the player.

The Player class should be implemented specifically, with specialisations including a random player, which overrides the method for choosing an action, replacing the functionality with a random move selection. This basic artificial player shows no intelligence but allows a play through of the game to be made and allows us to test the implementation works as expected.

Another main aspect of the game are tiles. There are playable tiles (locations) and tiles which can be placed by the player. These tiles can be represented with the same class, each tile should store it's type, location on the game board and the tiles owner. It would also be possible for only the tiles on the game board to be represented this way, with players simply taking ownership of these tiles and applying a value to them equal to the tile they wish to place. In this case, it is also important to store what terrain the tile is located on with respect to the game board, either land or water tile.

Cities will also need to be represented in the game. Since they have different attributes and functions to basic playable tiles, they should be modelled in their own class. Cities should also store their location on the game map as well as the pieces that have been assigned to the city. It would also be useful for a method or attribute which describes if the city is surrounded or not.

3.3 Graphical Interface

The complexity of the game makes it important to be able to visualise the current state of the game. For this reason, we will be creating a graphical interface that displays the game board with text representations of cities and tiles placed. The graphical interface will be produced using PyQt5 and the main implementation will be structured around a game board class.

The user interface at this stage of the project does not require user input, although the user should be able to continue the game to the next players move using input from the terminal, to keep the implementation simple. The main complexity in the user interface is recreating the hexagonal grid pattern this game uses. To do this, a hexagon class can be created which constructs a polygon from six points distributed evenly around a circle. These hexagons can then be iteratively overlayed to create our game board. Each of the hexagons will be coloured depending on their type, cities being coloured grey, land being a sand colour and the water being blue. Only locations which are playable will have a hexagon outline around them, with unplayable locations being rendered but without the outline. We are using the 2 player game board which is a 14x13 grid including none playable locations. Constructing the game board in a square grid helps with both positioning of the hexagons as well as being able to use an X,Y coordinate system when referring to a specific location.

The locations within the game board are rendered based upon a seperate map, represented by an array of numeric codes. These numeric codes represent the type of location: non-playable, water, land, city etc. The game board class simply iterates over this map, assigning the hexagon with the same coordinates as the item of the array the type represented in the array. This potentially allows for expansion to larger maps or allows the map to be very easily changed, the game board could be changed drastically by simply changing the values in the map array.

The game board also needs to handle the placement of tiles on the map by players. When a player takes ownership of a location, the tile is assigned a colour associated with that player (red or green). The hexagon also gains a text descriptor so that the user of the application can see the tile that has been placed. Cities also have text descriptors which indicate the pieces within the city: B - Buddha, H - Helmet, R - Rice. An example of the interface can be seen in figure 3.1

3.4 Implementation

When implementing the game in python I tried to follow the design as closely as possible, refactoring the design when necessary. The game class as expected holds the current state of the game. This is implemented using two arrays: one array contains all of the playable tiles, or tiles objects which can be assigned values. The second array contains a series of city objects, each representing a city on the map. The state of the game is determined by these two arrays as the current state of any play can be determined by the state of the cities and the state of the playable tiles. At the game setup, these lists are populated using the game map detailed in

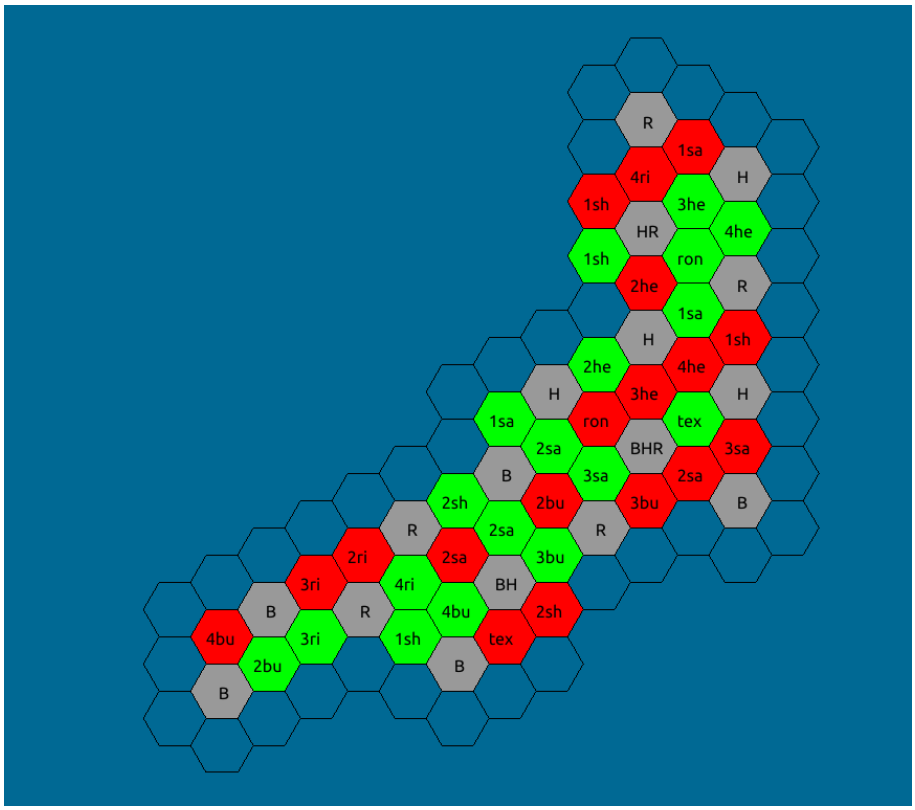

Figure 3.1: Graphical representation of a finished play

section 3.3 by methods within the game class. Once the state of the game has been initialised, the graphical user interface is drawn/updated and the pieces allocated to cities. The pieces are allocated to cities using a simple random number generator, allocating a piece depending on the number that is produced.

Once the game is setup, the main game loop starts and is provided with two player objects. The player objects are created independently of the game so that players can exist between multiple games (best of 3, etc). A player is selected at random to go first and are prompted for a move. Each type of player inherits a generic method detailed in section 3.2 which generates all the possible actions given the current state of the game board and the players current hand. The rules of the game are enforced through this method, with illegal moves not being generated as possible actions. A player may only choose one of the possible moves and thus must abide by the rules of the game.

The possible actions are generated by combining each possible tile that can be placed with each possible location it can be placed in, with locations being items in the game's playable location array. A move is represented by the value of the tile followed by the location the tile should be placed in. An example move includes [2, 15] which would represent placing a 3-Samurai tile in the 15th location in the array. Tile exchange tile and piece exchange tile possible moves are calculated separately, with tile exchange tile being represented with an extra destination location (example [15, 1, 20]). The first value is still the value of the tile, the first location is the tile to move and the second location where to move the tile to. Actions generated for tile

exchange pieces are only done so on friendly tiles. Similarly, piece exchange tiles can only be activated on cities and actions are generated for all possible cities which can accept a trade without causing duplicate pieces on one city.

When all of the possible actions in the players current state have been generated, it is up to the player to choose which move to make. For the purpose of this implementation, a random player was created which simply takes all actions and chooses one at random with no further intelligence. Depending on the action taken, if the player is still able to place tiles then they are prompted for an additional action, until the player can no longer place a tile. The player can of course choose not to place additional tiles.

These moves are then passed back to the game loop, which updates the state of the game and communicates with the graphical interface causing an update. The state of all of the cities is checked, looping through cities which are not surrounded and determining if they are still not surrounded. If all of the adjacent tiles are occupied then the city is flagged as surrounded and the pieces allocated to the appropriate player. If all of the playable locations which are land are filled, the game is ended. The winner is then calculated using the win conditions detailed in section 2.3.3.

Chapter 4

Artificial Intelligence

Now that the rules of the game have been implemented, we can move onto constructing an artificial player. The method that will be used as described in section 1.4 will be an iterative approach, first starting with the most basic player and improving it's intelligence based upon results from testing. All of the types of player discussed in this chapter are specialisations of the existing Player class, each incorporating different methods of selecting a move from a list of generated possible moves.

4.1 Irrational Player

At the beginning of a players turn, each move that is both possible and valid according to the rules of Samurai are calculated based upon the current state of the board and the tiles the player currently has access to in their hand. Since the player has access to all of the possible moves, the player simply needs to choose one of those moves based upon its value.

To start with, we need to prove that there exists a strategy which is superiour to irrational, random play. In such a play, a player will not consider the payoff of a particular move and will treat each move equally; essentialy playing at random. If a strategy exists that can beat random play then we can explore the heuristics required to enact such a strategy. Otherwise, a different approach would be required for creating an intelligent player. In theory, due to the strategic competitive nature of Samurai, a player which is irrational and does not make moves in an attempt to win the game should be easily defeated by a player who considers the payoff of a particular move but also should win roughly fifty percent of games played against another irrational player.

To implement this player we can treat each of the possible moves equally, using python's inbuilt random library to make a random choice from each of the moves. This player would almost certainly be hopeless against a human player or any rational computer player, but is an essential step in shaping the development of a more advanced player and ensuring the software implementation works as expected. It should also be noted that an irrational player, although not trying to win, is also not actively trying to lose, it will be interesting to see during testing how the irrational player holds up against some of the more basic heuristics and will help us determine if a particular heuristic is worth implementing depending on it's ability to add rationality to a particular move.

4.2 Basic Player

The aim of building a basic player hopes to show that by adding simple rationality to the moves made, simple improvements in strategy can significantly improve win percentages against a player who chooses moves irrationally. The basic player represents a player who

understands the rules of the game and how to use those rules to make moves which result in a payoff for the player, instead of choosing a move at random. With this player we hope to also show that competence in playing the game can be improved by adding additional heuristics as well as experimenting with their influence on the final payoff. A basic player will not necessarily consider the moves of their opponent in a strategy and will simply aim to collect the highest possible payoff with minimal knowledge of the game state.

The approach that we will take to calculating the payoff of each move is by associating a numeric value to each tile in each playable location. At the beginning of each move, the player will assess the current state of the board and determine the payoff of playing each possible tile in each possible location. This results in an array of dimensions (playable area * unique tiles) or in our case (74 * 17). This area can decrease as the play progresses due to locations being occupied and tiles being used, meaning that values for those tiles and locations not needing to be calculated. With these values calculated, each possible move then has an associated value; the move with the highest numeric value has the highest payoff and is chosen as the players move. If multiple moves have the same value then one of those moves is chosen at random.

The obvious first step when applying heuristics to mimic a beginner player is assigning values to moves which will add influence to a city in hope of capturing pieces from that city. The main aim of the game is to win the support of castes using those pieces and so the objective of a beginner should be to capture the pieces. This means that for example placing helmet tiles next to a city which only has a Buddha piece to capture should have a lower associated value than placing that same helmet tile next to a city with a helmet piece. The second of those options contributes payoff to the player by moving them closer to winning the game and thus should have a higher value associated with them. This value can also be changed depending on the amount of influence a tile enacts on the city, with a four-influence tile having a higher value than a 3-influence tile due to it's higher probability of winning the piece.

If a city is surrounded, all adjacent tiles are immediately given negative values with a negative value being attributed to moves which would cause a disadvantage to the player. This allows the player to avoid situations where placing ships next to cities which have already been captured would cause the player to waste a tile, thus reducing it's chance of victory. All moves start with a negative value and values are only added to moves that are directly in the interest of the player.

These simple heuristics maximise the probability that each move made by the player will be rational, improving the players chances of victory. More heuristics will need to be added and refined before the player is capable of playing competently, however this first step aims to prove the value of developing a strategy for playing Samurai.

4.3 Improving Heuristics

The best way to improve upon the simple heuristics are to add more functions which influence the payoff of a particular move. This involves discovering new ways to improve the current strategy. Each time a new heuristic is added or an existing one modified so that the win percentage of the new experimental artificial player increased and the win percentage of the baseline player decreased, the causing heuristic is likely improving the strategy of the player. The experimental player can then become the new baseline player and the process repeated.

The first factor that builds a good strategy is the ability to calculate the payoff of a particular move based upon the moves made by an opponent. In Samurai, it is important to consider the influence both yourself and enemy players are enacting on a city, to give you the best chance of both using your pieces efficiently but also not wasting pieces. For this reason, I added an additional heuristic which reduced the predicted payoff of a move if the difference in influence between the player and their opponent is too great. The value of placing a tile adjacent to a city that has a large amount of influence either by an enemy or by your own tiles is decreased by half with this new heuristic as the chances of a tile placed there being wasted is considerable. This results in better informed moves due to taking into consideration enemy plays but also creates a better balance when capturing pieces. In order to win the game, it is not how much influence but how many castes are won that is most important, so spending additional valuable tiles securing a city that is uncontested results in bad strategies.

Another major heuristic which improved the win rate of the artificial player significantly is adding weight to a move if that move would cause a city to be surrounded. It is common in Samurai to surround cities as quickly as possible, especially if you have an influence advantage over the city. Surrounding cities results in pieces won, which provides you with an advantage and your opponent a disadvantage. At first I added additional weight to locations which are adjacent to cities based upon the number of tiles required to surround them. If a city only required one more placed tile to surround the city, that tile would have maximum weight, with cities requiring more tiles to surrounded them having proportionally less. I was able to observe faster piece taking with this heuristic, although it did seem to hurt in the late-game phase of a play due to a player spending their valuable, high-influence tiles surrounding cities they are already certain to win.

To further improve on the previous point, I added complexity to this heuristic, only adding positive weight if the player will win the pieces within the city if the city is to be surrounded. The weight of a move if the opponent was likely to win the pieces when the city was surrounded was reduced, forcing the opponent to spend their own tiles surrounding a city if they wish to win the pieces, making them weaker when it comes to contested cities. Additionally, I attempted to add or remove weight depending on the tile used to surround the city. Using a high value tile to finish surrounding a city when it could be surrounded with a lower value tile is a poor move and would cause weakness in the late game, so prioritising lower value tiles when the influence is not important helped increase win statistics.

When implementing the artificial player, considering heuristics for the special pieces is not easy due to the complexity of the rules regarding these pieces and their special use cases. It is rare that much use can come from the special pieces if used randomly but can be powerful if used correctly. Unfortunately these special cases are difficult to predict. I was unable to produce a heuristic that made effective use of the piece exchange, however the tile exchange implementation did help improve win statistics. I calculated the payoff of using a tile exchange by maximising the value of a piece being moved to a new location and minimised the value of the piece in the location it is currently in. This would provoke situations where tiles for example around surrounded cities would be moved to locations that could be highly beneficial to the player and could potentially turn the tide of the play.

Once the base heuristics were determined, I was able to experiment with different combinations of weightings and observe how they effected win conditions. It is clear from the experiments in chapter 5 that many of the heuristics are not independent and require fine tuning to produce the best possible strategy. This would certainly be a case where the use of hyper-heuristics would be valuable to progressing the player further.

Chapter 5

Implementation Testing

To ensure that the implementation worked as expected, the rules of the game were enforced and to improve the artificial player, it is important to run periodic tests throughout the project. This chapter will discuss the observational and mathematical tests performed on the implementation and the results of games between different artificial players.

5.1 Game Implementation

Testing the implementation of the games and rule set is essential before any progress can be made into the artificial player. Without a good foundation, the player would likely develop faults or provide inconstant results. I used the random player described in section 4.1 to carry out a full play of the game. I analysed each of the moves made on the graphical interface, ensuring that each move was within the rules. Once I had determined that the moves being made were valid, I used the software to perform 1000 plays of the game. After some minor fixes, the software was able to perform all 1000 plays without any errors or issues. I used print statements to display the scores of each player and their influence at the end of each play and was able to calculate their score from the graphical interface and compare it to the printed version. From these scores, I was able to calculate the winner and also compare it to a printed version. At first of course, there were some issues, however these could be easily identified and fixed with the methods described. The graphical user interface while being developed could be tested visually and compared with expected traditional game boards.

The user input supplied to the application in the form of command line arguments and terminal input was tested using boundary cases. The input is kept as simple as possible and have clear instructions on how to progress. If incorrect command line arguments are passed to the application, a helpful prompt is displayed to assist the user. Plays of the game are progressed by providing user input of any form, although the user is prompted to press the enter key. Plays can be skipped by entering S and then pressing enter. This allows plays to be viewed both move-by-move or many plays at a time, making evaluation of win percentages more efficient for the user.

5.2 Testing Basic Player

The first testing of the artificial player was the irrational player. I wanted to carry out some tests using this player to identify a base skill level that could be improved upon. Additionally, testing this player can be used as proof that the game rules are implemented correctly if the random players have roughly fifty percent win rate over a large sample of plays. Throughout the testing, I performed 3 sets of 200 plays and evaluated those plays into a win percentage. an increase in overall win percentage likely represented a positive strategy iteration. The results of testing the random player can be found in figure 5.1.

Player	Test One	Test Two	Test Three	Win Percentage
Random 1	89	101	102	48.6
Random 2	111	99	98	51.4

Table 5.1: Random player vs. Random player

Once this base had been established, testing the player with a single heuristic function was next. This player relied on simple tile association, applying higher weights to tiles placed next to cities which could be influenced by those tiles. This test hoped to prove that a strategy exists which could defeat irrational play as well as developing the first of our heuristics. The test results can be seen in figure 5.3. The test shows that although the strategy is simple, rational play defeats the irrational player almost every time, proving our hypothesis. Since this has been established, we can explore additional heuristics and modifying existing heuristics in a hope to improve the artificial player's strategies.

Player	Test One	Test Two	Test Three	Win Percentage
Random	1	1	2	0.7
Experimental	199	199	198	99.3

Table 5.2: Random player vs. Tile Association

5.3 Improving Heuristics

Since our hypothesis has been established, we can focus on testing additional heuristics and observing their outcome. Since the basic experimental player increased the win percentage significantly, this player becomes our new baseline player (base). The next step in the development iteration involved the exploration of heuristics to weight tiles based on a cities current influencers: both friendly and enemy. The results of this test can be seen in figure 5.4. These simple heuristics provided the player with an insight into how to react to moves made in previous game states, rather than simply analysing the current one. This heuristic provided a roughly 15 percent increase in win rate for the experimental player and thus becomes our new base player.

Player	Test One	Test Two	Test Three	Win Percentage
Base	73	57	70	33.3
Experimental	127	143	130	66.7

Table 5.3: Base player vs. City Influence Heuristic

Further tests were performed on different combinations of these heuristics and little progress was found. It is likely that these heuristics provide a base increase in win condition simply from consideration rather than the amount of weight placed upon them. The alternative is that the heuristics are not independent and require fine tuning for maximum effect. Tests performed

with a number of different combinations provided results around the average of fifty percent win rate not favouring any particular player. Thus, these combinations were not included into the base player.

The next successful heuristics tests involved the addition of heuristics aimed at adding weight to placing tiles which result in city captures. This test showed the importance of prioritising moves which resulted in early game advantages and the winning of pieces over simply gathering influence. The first tests with this heuristic can be seen in figure 5.4 and resulted in a roughly 15 percent increase in win rates for the experimental player. Again, this experimental player became the base. Further testing was done in an attempt to prioritise low value pieces when surrounding cities that are certain wins as well as avoiding cities that are certain losses. These tests did not require a new heuristic and instead improved on the current heuristic associated with capturing cities. The results for that test can be found in figure 5.5 and the findings show an increase in roughly 10 percent, becoming the new base player.

Player	Test One	Test Two	Test Three	Win Percentage
Base	75	66	74	35.8
Experimental	125	134	126	64.2

Table 5.4: Base player vs. City Capture Heuristic

Player	Test One	Test Two	Test Three	Win Percentage
Base	89	84	80	42.2
Experimental	111	116	120	57.8

Table 5.5: Base player vs. Advanced City Capture Heuristic

The final notable test performed to improve the artificial player was with the introduction of strategy for using the tile exchange tile. All future tests provided minimal gain in intelligence, with the tile exchange providing a 15 percent increase in win rate. This increase was unexpected but understandable given the power of the tile. A player who uses the tile exchange tile effectively would essentially be able to use a major piece twice, remaining effective into the late game and receiving an advantage over a player who uses the tile exchange irrationally. The results for tests with a player making maximum use of the tile exchange against a player using the tile randomly can be seen in figure 5.6. As the intelligence of the agent increased, the returns from additional heuristics and modified heuristics becomes increasingly small, with only very minor changes in win rate. Another method of recording the intelligence of each move would have to be produced to provide more accurate tests. Due to the element of chance within the game, from the hand that is randomly allocated, these small improvements are offset by a game lost due to a poor starting hand, etc.

Player	Test One	Test Two	Test Three	Win Percentage
Base	70	66	63	34.8
Experimental	130	134	127	65.2

Table 5.6: Base player vs. Tile Exchange Heuristic

Chapter 6

Conclusion

This project has touched on a number of aspects in game theory and artificial intelligence that has allowed for the construction of both a digital representation of Samurai but also the building and improvement of an artificial player that is competent at playing the game. For us to judge the effectiveness of the project, we will see which of our initial project objectives have been met. Additionally, I will analyse my performance throughout the project in a personal reflection, discussing my own strengths and weaknesses as well as my opinions on the project. Towards the end we will discuss the possibility of future research outside of the scope of this project.

6.1 Project Outcome

The purpose of this project was to successfully explore the four objectives that were defined in section 1.3. In order to determine the accomplishment of this report, we must discuss if these objectives have been achieved.

The first objective of this project was to create a digital version of the board game Samurai. This objective I believe has been successfully accomplished by this project, the software deliverable is capable of playing a full round of Samurai as defined in the project scope (section 2.3.4) and has been tested to ensure that all of the rules are followed. The game can be played by two independent players, taking it in turns to place tiles on the game board. All of the moves available to the players are determined by the games rule set and thus a player is only able to make moves that are within the rules of the game. The software is able to produce a winner which is accurately calculated from the ending state of the game.

The second objective that this project hoped to fulfil was creating a graphical representation of the board game Samurai. This involved translating the current state of the game and any moves by players onto a GUI. This objective has also been achieved, the game can be observed through a graphical representation: allowing a user of the software to progress through a play move-by-move observing the difference in state at each move throughout a play. The graphical representation replicates the style used on the board game with a hexagonal grid, highlighting tiles which have been placed by a particular player. Each tile and city within the game board has an appropriate descriptor identifying the tile or pieces that are placed at the location. Although the interface is simple, it is functional and invaluable for using the software as a research tool, allowing the user to see which moves are being made by each player.

Developing a simple artificial player for the game was the third goal of this project. In order to test that the game representation was functional a naive player was built which chose moves at random. This implementation although simple proved a number of important aspects about

the game, including that a strategy exists that allows for a consistently better payoff than random play. On top of this, a player was produced which chose moves based on simple heuristics and showed a degree of intelligence, favouring moves which would likely result in a positive payoff for the player. This player was tested and later refined but showed the foundations for an artificial player for the game Samurai.

The final objective set out by this project was exploring advanced artificial players that would be competent at defeating a player that makes rational moves in order to maximise their payoff. Although limited in extent of the techniques used to fulfil this objective, the initial players were improved substantially, to a point where the players with more advanced heuristics are able to win against the base heuristics player almost every time. This project does fulfil this objective but there are certainly grounds for more research which we will discuss in section 6.3.

6.2 Personal Reflection

Overall I believe that this has been a positive end to the project, with the objectives all being satisfied, although I do believe there was plenty of room for improvement and had the project been better managed, the possibility of a more refined artificial player.

This project gave me the opportunity to explore areas of artificial intelligence which I have not touched before, reaching beyond the taught material within my degree programme. Multiple previously unknown techniques were explored which certainly improved my knowledge of the field. Due to this project, I would certainly feel much more comfortable leading another project or further research into artificial intelligence. Furthermore, it gave me the opportunity to work with PyQt, a technology I previously had no experience with. I certainly think that the quality of the report could have been improved and refined from an academic standpoint, but as the first technical report I have produced, provided plenty of opportunity for learning which I'm sure will be invaluable in the future.

One of the major improvement points that I can take away from this project is the importance of time management. Although I did make a project plan at the start of the allocated time, I found that it was often put aside due to external factors such as other modules, exams etc. In future I would consider developing a more dynamic plan with would allow me to better meet deadlines for the module as well as providing plenty of time for background research and the actual development of the software. It is important to carefully analyse the project beforehand and try to predict the amount of time each section will take to avoid situations where there is not enough time at the end of the project to both complete the software deliverable and the report write-up in the detail and quality that I would prefer.

Continuing from that point, a large portion of the project was spent exploring technique which ultimately did not become an aspect of the final project. I spent a considerable amount of time exploring how reinforcement learning could be used without contributing any significant progress to the project. This ultimately left me behind schedule for writing the report and

building the software. This could have been avoided with a better project plan: assessing beforehand in more detail how to mediate risk in research as well as more thorough background research.

The size of the project means that there are a number of valuable learning moments throughout the lifespan of the project that will I will make sure to consider before starting any projects in the future. I think I could certainly have made better use of the resources available to me, meeting with my supervisor more often rather than trying to tackle the project alone. However, I have thoroughly enjoyed the project and will likely continue research in the area past the deadline and outside the scope of this project when possible.

6.3 Future Work

Using the findings from this project it would be interesting to explore how the artificial player generated within the scope of this project could be applied to an extended scope. One of the main areas of future research would be expanding the game representation to a three or four player game. The representation would be easily adapted for the increased size and the rules remain the same. However, it is likely that different combinations and weightings of heuristics would be better suited to Samurai with more players and a bigger board size.

An improvement that I would make to the existing software would be to incorporate a more user friendly graphical interface. The interface could be more useful with the ability to see the weightings given to each location by each player - allowing a user to visualise exactly why a move has been made rather than adjusting heuristics based upon win percentages. It would also be interesting to modify the interface slightly so that a human player would be able to play against the artificial player, further testing the capability of the artificial player. Although I doubt this would improve the functionality of the software as a research tool, it would be interesting to see the ability of the AI against another human player.

Finding heuristics that could produce good strategies for all of the tiles proved to be more difficult than expected. Some tiles such as the special tiles (tile exchange, piece exchange) are difficult to use effectively in play and usually are special cases. These pieces can have a large impact on the game but building a strategy that can use them to their potential is difficult. Further research could look into implementing more advanced heuristics for these pieces to increase the tool set of the artificial player, allowing for more complex strategies. On top of this, the software created in this project treats each tile placed within a move as a seperate entity, assigning values to each tile placed individually. In reality, a player would move multiple tiles in unison for an overall effect. Applying heuristics to groups of tiles rather than single tiles could certainly improve the ability of the artificial player.

Finally, exploration into hyper-heuristics could provide meaningful insights into the combinations of heuristics used to generate values for each move. The current heuristics do not adapt to different situations and are manually adjusted based upon observations. Allowing the

artificial player to modify these heuristics based upon the current state of the game and over a large number of plays would refine the heuristics and provide better strategies for the player to choose from.

References

[1] S. Brocas, Carrillo. The path to equilibrium in sequential and simultaneous games. 2016.

[2] G. C. et al. Cooperative game theory: Basic concepts and computational challenges. *IEEE Intelligent Systems 27*, page 86, 2012.

[3] H. Gintis. Behavioural game theory and contemporary economic theory. *Analyse and Kritik*, pages 25–30, 2005.

[4] m van otterlo. Reinforcement learning and markov decision process. pages 3–42, 2012.

[5] J. Mycielski. Games with perfect information. pages 43–45.

[6] R. Myerson. Game theory: Analysis of conflict. *Harvard University Press*, 1991.

[7] G. Owen. Zero-sum theory. *Game Theory: Third Edition*, page 11, 1995.

[8] E. Prisner. Terminology of game theory. *Game Theory Through Examples*, pages 1–10, 2014.

[9] H. R. Robert Luce. N-player game theory. *Games and Decisions: introduction and critical survey*, 1957.

[10] S. Russell. Artificial intelligence: A modern approach. pages 163–171, 2003.

[11] M. Shubik. Game theory models and methods in political economics. *Handbook of Mathematical Economics*, pages 285–330, 1981.

[12] A. M. Turing. Digital computers applied to games. *'Faster Than Thought' by B. V. Bowden*, pages 286–310, 1953.

Appendices

Appendix A

External Material

This project was hosted on GitLab for the purpose of version control and issue tracking. The GitLab repository can be found at:

https://gitlab.com/sc15js/final-year-project

The repository includes all of the source code as well as instructions on how to build, run and use the software.

This project made use of Python 3 for the software implementation. The Python 3 'PyQt5' library was used to construct the graphical user interface. No other external resources were used.

Appendix B

Ethical Issues Addressed

When conducting research in any field it is important to consider the impact of this research beforehand to ensure that the research is conducted in a responsible manner. Many projects, particularly in computing, due to the handling of sensitive data and consumer trust placed on software, have some ethical issues associated with them. Here I will discuss some of the potential legal, ethical, social and professional issues that could be associated with this project and how they could be prevented.

This project, within the scope defined by this report, does not present any obvious legal issues. Legal issues are often present in software engineering when sensitive data is being stored or when the software itself reaches at the boundaries of what is legal (for example software that encourages piracy, or illicit messaging services). The scope of this project does not concern any of these issues, however if human testers were used in the future, it may be a legal requirement to receive written consent for those testers personal data to be recorded as part of the research. This project does not have that requirement and thus is not effected by legal issues.

Furthermore, the scope of this project does not clearly present any ethical issues that could be perceived negatively by a user or community. Some ethical issues that software engineers may have to consider is the use case of their software, if to software could be easily adapted for nefarious purposes. The software developed in this project does handle artificial intelligence which some may consider a potential future ethical problem; especially how we approach designing artificial intelligence to protect the future of the human race. Fortunately, the scope of this project is limited in that extent and is extremely unlikely to present any issues of that kind.

Another consideration software engineers must make is social issues associated with their software. This project does not include any human elements and so does not need to address any social issues from the research conducted. Future features could be considered for the user interface of the project such as text to speech integration and more considerations for impaired users. However these things would require significant development time and/or deduct from the research potential of the software. Although improvements could be made, there are no obvious issues that this project faces in regard to today's social climate.

From a professional standpoint, the project is stored on a public version control system, allowing others to view, learn and possibly contribute to the project in the future, in the hope of contributing meaningful research to the field. However due to the individual nature of this project, it is unlikely to face any professional issues that could arise from a team project or a project with significant outside influence.