Applying Artificial Intelligence and Machine Learning Techniques to Create Varying Play Style in Artificial Game Opponents

Nicholas John Sephton

Doctor of Engineering

University of York
Computer Science
October 2016

Abstract

Artificial Intelligence is quickly becoming an integral part of the modern world, employed in almost every modern industry we interact with. Whether it be self-drive cars, integration with our web clients or the creation of actual intelligent companions such as Xiaoice[1], artificial intelligence is now an integrated and critical part of our daily existence.

The application of artificial intelligence to games has been explored for several decades, with many agents now competing at a high level in strategic games which prove challenging for human players (e.g. Go [127] and Chess [73]). With artificial intelligence now able to produce strong opponents for many games, we are more concerned with the style of play of artificial agents, rather than simply their strength.

Our work here focusses on the modification of artificial game opponents to create varied playstyle in complex games. We explore several techniques of modifying Monte Carlo Tree Search in an attempt to create different styles of play, thus changing the experience for human opponents playing against them. We also explore improving artificial agent strength, both by investigating parallelization of MCTS and by using Association Rule Mining to predict opponent's choices, thus improving our ability to play well against them.

[1] `http://www.msxiaoice.com/`

Contents

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my academic supervisor Professor Peter I. Cowling for his continuous support of my EngD study and related research, for his patience, motivation, and immense knowledge.

My heartfelt thanks go to all those at Stainless Games for their funding of my research and this opportunity to fulfil my long held desire to pursue a career in research and teaching.

I'd also like to thank the rest of my thesis committee: Dr Daniel Kudenko, Mr Nicholas H. Slaven and Prof Pieter Spronck for their insightful comments and encouragement.

I thank all of those who have shared an office with me and directly or indirectly supported my research, particularly Dr Edward Powley, Dr Daniel Whitehouse and Dr Sam Devlin.

The work displayed here was supported by EPSRC (`http://www.epsrc.ac.uk/`), the LSCITS program at the University of York (`http://lscits.cs.bris.ac.uk/`) and Stainless Games (`http://www.stainlessgames.com`).

I would also like to thank my family: my mother, my father and my sister for supporting me emotionally throughout writing this thesis and in my life in general.

Dedication

For Elizabeth Mary Newsome Simmons

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. The majority of the work is drawn from several papers published at appropriate conferences. The work presented in this thesis was carried out as part of a joint research project, where I contributed original ideas, actively participated in all discussions and meetings, and co-authored both the research code and written publications.

Chapter 1

Introduction

This thesis focuses upon creating varying play experiences for human players using Artificial Intelligence techniques. I apply particular focus to the use of *Monte Carlo Tree Search* (MCTS) algorithms along with various enhancements to attempt to create different styles of play. I also use Association Rule Mining to enable agents to predict hidden information which is only available to their opponents, with a view to enabling further behaviour which makes use of this information in future research.

The term "MCTS" [36, 42, 82] describes a family of search algorithms which explore a given decision space using guided stochastic sampling to determine the most promising moves. It is most frequently applied to games, and was responsible for recent success in Go [127]. The principle of MCTS is to use a large number of random playouts from the game state to be analysed, and to backpropagate win% information from terminal states, thus creating and updating statistic information on the decision space. The main difficulty in selecting the path for simulation is maintaining a balance between the *exploitation* of established powerful routes of play, and the potential to find new more powerful routes of play through further *exploration*. Part of the reason MCTS is so attractive is that it can operate completely aheuristically, requiring only a forward model of the game in question

and a starting game state in order to provide a decision. In this thesis, we use plain MCTS as our default agent, and employ a variety of enhancements to modify agent behaviour.

Association Rule Mining [2] (also sometimes referred to as *Association Rule Learning*) is a technique for determining relationships between variables in large sets of correlated information. Successful application of association rule mining results in the discovery of a set of strong rules which describe the data in the set of information, and can be used to make reasonable predictions on other data from the same or similar systems. These techniques are also often referred to as *Market-Basket Analysis*, due to their common application in the determination of likely patterns of product purchases in supermarkets. In this thesis, we use Association Rule Mining to model opponent pregame choices in a complex card game.

The work in this thesis provides insight into the application of artificial intelligence techniques in creating adversarial gameplay agents which not only consider play strength, but also variance of play style. We consider decision search and the determination of imperfect information which can be used to inform agents, and therefore increase their overall effectiveness in both play strength and variance in playstyle.

Progressing towards an agent which considers agent playstyle of high importance particularly to the games industry [34], which currently struggles for adaptive agents for highly complex games. Even recent successes in game AI, such as that in the popular game *Alien Isolation*[1], has had a variety of issues with undesirable behaviour, and required substantial restrictions to operate as desired [56].

Our sponsors *Stainless Games*[2] originally showed interest in this work as they desired a more appropriate adversarial agent as part of the *Duels of the Planeswalkers*[3] series of games. The issues with the existing artificial intelligence system were twofold. Firstly, it required heavy customisation with each release of the game in order to handle the new cards

[1] `http://www.alienisolation.com/`
[2] `http://www.stainlessgames.com/`
[3] `http://magic.wizards.com/en/content/magic-duels`

and game rules added. Secondly, it often required manual restriction to behaviour to account for it behaving in an undesirable manner towards human players. This undesirable behaviour usually took the form of illogical moves when the artificial agent saw no possible route to lose, or what seemed like intentional griefing of human adversaries. The initial objective of our work was to provide Stainless Games with a generic-use C++ library that implemented MCTS with appropriate configuration options, and could be deployed into their games.

In the first year of our work, it became clear that it would not be possible to work directly on the Duels of the Planeswalkers game series due to commercial concerns which prevented information sharing. As such, we expanded our targets to other games, with the hope to one day return to work upon Duels of the Planeswalkers and apply our research there. Some of our work here focusses on the parallelization of MCTS, which was originally an attractive target due to the strict memory requirements of the Duels of the Planeswalkers games. When it became clear that we would not be able to deploy our library as part of DotP, we decided not to pursue that work any further, and instead focus on further modification of MCTS. As part of our research, we have also provided a C++ library that implements MCTS, as was the original agreement.

In this work, we successfully create an agent based upon existing MCTS technology which demonstrates varied playstyle while maintaining play strength in a complex card game. We also offer several techniques for modifying artificial agents which may be successful in created varied playstyle in other games. We provide a general survey of parallelization techniques for MCTS, and a specific new technique for application to ISMCTS. Finally, we offer a selection of techniques for heuristic modification of data mining techniques in application to the determination of imperfect information.

1.1 Background

Artificial Intelligence is used in many different applications in modern life. While it is most often associated with play in games, it is also applied in autonomous vehicles, search engines, online agents (such as Siri [9] or Cortana [94]), image recognition and many other systems. A central problem in artificial intelligence research is usually that of optimised decision making, and by expansion of that goal, deduction, reasoning and problem solving. There are also many other problems under study, including those relating to creativity (such as story-writing [91] or music composition [95]), perception and interpretation of visual data, and issues of social intelligence (such as Affective Computing [110], and Natural language processing [40].)

A large number of modern artificial intelligence systems involve interaction with humans, and as such most artificial systems must be strongly constrained to ensure that they behave in an acceptable manner, despite these constraints restricting functionality and flexibility of the agents. This is especially true when the systems are security or safety critical. A notable example is that of a Microsoft chatbot named *Tay*, which quickly took a dark turn when allowed to interact with internet users in an unconstrained manner [107]. While unconstrained Tay was certainly more flexible and able to discuss more topics and display a wider range of opinions, interacting with Tay quickly became unappealing to the majority of humans, and thus artificially constrained behaviour became necessary. It would have been far preferable in this situation for Tay to be able to make the decision of what was appropriate for its own audience, and thus be truly in control of learning and interacting on all topics while being aware of what behaviour is unacceptable.

When applied to games, artificial intelligence is most frequently used to generate intelligent behaviour in non-player characters, most often in an attempt to simulate human-like behaviour. However, the focus is frequently upon strength of play, as most industry imple-

mentations of AI exist as opposing agents in games to provide challenge for human players, and to fail on this front would quickly invalidate the challenge presented by a game. It is also common for games to offer agents under the control of a human player, who use AI to optimise individual actions in a larger command given by the human user (e.g. Moving units from one position to another in a Real-Time Strategy game requires the units to use an algorithm to create a path between the two points.) The focus in this case will also be very much on optimality rather than any other objective such as interesting or human-like behaviour.

It is also important to note that while play strength is the most important single consideration in game AI, current technology is insufficient to provide adequate play strength in complex games. This can be seen in such strategy games as Starcraft 2, where the highest difficulty agents are required to cheat in order to provide a significant opponents for the best human players[4]. So while differing styles of play should become an objective, it is important to not abandon play strength completely.

Since the conception of MCTS in 2006, it has had substantial effects upon the development of artificial game playing agents in many fields such as strategic board games [88, 87, 64] and General Game Play [92]. In more recent years, it has been shown that the most advanced agents created are able to outperform the world's best human professional players in particular games [127].

As AI techniques continue to improve, it is natural that there will be a stronger demand for more human-like play from AI agents. This will partially be due to demand for AI to make more human-like decisions, but also to provide a better environment for learning a game, as well as a more user friendly experience. The standard at the moment in the games industry is to provide a scripted tutorial for the first games, as the MCTS AI effectively cannot be trusted to play both weak and logical moves at every opportunity.

[4] `http://starcraft.wikia.com/wiki/AI_script`

In 2009, Stainless Games released "Duels of the Planeswalkers" (DotP), a digital interpretation of Magic: The Gathering designed to abstract away a lot of the complex rules and play more like an arcade game. Since that time, Stainless have released several more games in the Duels series, each with general improvements to the game, new content and (to some degree) improved AI abilities. While the AI in the most recent game is a strong opponent, it still struggles in certain circumstances, and is also makes moves which seem illogical to human players. Producing a strong, entertaining AI that was easily adapted to new games in the DotP series is a substantial objective for Stainless Games, representing many man-hours of work saved.

When viewed from the commercial perspective, the objective of an AI is not necessarily to beat the human player, but more to provide an interesting experience. This might include beating the human player, but it should not be the primary objective. In fact in certain game situations artificial agents may take actions that have no particular advantage over other available actions, but have a direct negative effect on human enjoyment of the game, as no consideration is given to such factors.

1.2 Research Questions

It would be of interest to create an AI that, while focusing on making strong decisions, was also able to assess a potential decision for its novelty or creativity, and thus provide an entertaining experience for the opponent, much like another human player might. A choice as to whether to take this decision could then be made based not only on its optimality, but modified by its novelty.

A relatively small amount of research has been performed in the area of the "fun" of a given move, but I believe that this will become of more interest in the near future. As AI,

human-computer interactions and psychology researchers work more closely together, accurately simulating a human player is not necessarily the same as simulating a strong player, and while the latter currently seems like a more desirable goal, I believe that simulating a human will become more desirable as our ability to create artificial players advances. Once creating a challenging artificial player is easy for any particular game, the majority of games will have access to strong artificial agents that can routinely defeat the majority of human players. Therefore, it is clear that advancement of play skill will become less relevant from a commercial point of view, as conclusively beating high proportion of your target market is unlikely to make your product more commercially attractive, but rather will likely drive people away from your game. Instead, the focus must shift to making adaptive, entertainment-focused agents which aim to optimise human experiences.

1.2.1 Can modification of MCTS create artificial agents that affect playstyle?

A core objective of our work is the creation of an artificial agent that has the primary objective of optimising the play experience for the human opponent. This objective will include play strength as a sub-objective, primarily because an important factor in playing a game against an opponent is that opponent's play strength. In our work here, we use a measurement corroborated by Schmidhuber's compression theory, (also known as the *Formal Theory of Creativity, Fun and Intrinsic Motivation* [120],) which shows that situations which are of interest to us are those which are somewhat complex to understand, but not so complex that they become incomprehensible. I will therefore attempt to modify agents so that they produce game states falling into a midrange of complexity for the game in question. In order to do this, I will also develop complexity measures to determine the relative complexity of the game states I examine.

1.2.2 Can we create an artificial agent which displays modified behaviour without compromising play strength?

It would be of interest to modify a MCTS search algorithm so that instead of selecting the optimal move by play strength at the end of the search, it selects a different move based on specific configurable criteria. It would seem logical then to modify move selection towards states within a midrange of complexity, and to observe the significance in play.

For this work, I will consider both the modification of the MCTS process itself by using heuristic pruning, and also the Action Selection Mechanism to modify which move is finally selected once the search process is complete, investigating playing strength for modified versus unmodified MCTS.

1.2.3 Can the application of game knowledge improve rule artificial intelligence techniques for modelling opponent knowledge?

Using a combination of Machine Learning and Heuristic Knowledge, we will attempt to modify a well-established data mining technique to be applied to opponent deck prediction. We will then apply these techniques to a large database of game data, and determine if these modification have been successful in increasing the effectiveness of the techniques.

1.3 Publications

The following publications result from our work here.

- "Parallelization of Information Set Monte Carlo Tree Search" (2014 IEEE Congress on Evolutionary Computation (CEC) [123])

- "Heuristic Move Pruning in Monte Carlo Tree Search for the Strategic Card Game Lords of War" (2014 Conference on Computational Intelligence and Games (CIG) [122])
- "An Experimental Study of Action Selection Mechanisms to Create an Entertaining Opponent" (2015 IEEE Conference on Computational Intelligence and Games (CIG) [124])
- "Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner" (2016 IEEE Conference on Computational Intelligence and Games (CIG) [121])

1.4 Thesis Overview

This thesis is split into seven chapters. Chapter 2 contains a literature review of the fields relevant to the work contained in the remaining chapters, most principally MCTS. Chapter 3 introduces a series of games on which we have experimented, and also the software environment which we created to perform all experimentation. Chapter 4 describes our work on parallelizing MCTS, and the effectiveness of the various parallelization techniques we explored. Chapter 5 discusses our work in the modification of MCTS algorithms to simulate different behaviour in artificial agents. Chapter 6 contains our work on the application of data mining techniques to a specific opponent modelling problem. Finally, chapter 7 contains a summary of all work performed here, and some discussion of future work that could be performed from where this work leaves off.

Where possible, our contributions have been explicitly clarified.

Chapter 2

Literature Review

2.1 Introduction

The purpose of this chapter is to gather and unify information about various Artificial Intelligence techniques which we will apply during research. The most central of these techniques to this work in this thesis is Monte Carlo Tree Search (MCTS), and as such it has the greatest focus in this chapter. We will also review the application of MCTS to games and game playing, particularly to complex games which have so far defied other artificial intelligence techniques. We will discuss Rule Association Mining in application to opponent modelling, and also some literature relating to human interest and differing playstyle from artificial opponents.

This chapter will first cover some basic AI techniques and concepts (section 2.3), then provide a thorough review of current MCTS research (sections 2.4 & 2.5), before moving on to discuss research into human play experience and entertainment (section 2.6), and finally covering some background for the Data Mining Techniques which were used in my research (section 2.8.)

2.2 Games & Game terminology

The section will introduce some standard game terminology which will be used both in academia and throughout this document.

2.2.1 Game Terminology

The following terminology is generally standard to game theory and artificial intelligence research:

State:

The set of all information for a given position in a game. The game state of a game as it is first created is often called the *Initial State* of a game. Other game states encompass all positions reached in a finite number of *moves* (or actions) in a game. The *Terminal State*s of a game is the set of states where no further moves are possible.

Player:

An agent who makes moves which alter the game state. Depending on the game, players may make moves in turn or simultaneously, but in most games they are the primary agent that influences the transformation of one Game State into another.

Move:

An action taken by a player which (usually) alters the game state. Some games include *Pass Moves*, which generally have little to no effect on the game state and signify the player taking no action or passing. Some games also include *Chance Moves*, which represent

actions which a player cannot accurately predict the result of taking, such as the result of a card draw or die roll. It is usual to say that there is an artificial player that makes these moves, sometimes referred to as the *Environment Player*.

Playout:

The entirety of a game's course, from the initial state to any terminal state. To observe a Playout, the game begins in the initial state, and then actions are taken according to some playout policy until the game reaches a terminal state.

2.2.2 Formal Game Definition

A game is made of a set of possible states, S, which represent every possible configuration that the game could reach during a Playout. The initial state of a game is defined as s_0, with other game states represented as $\{s_1,\ s_2 \dots s_i\}$, for a finite set of states. From any given state, there exists a set of actions A, which allows transition to another state (in the case of a terminal state, this set will be empty). The set of terminal states is defined as $S_T \subseteq S$.

A state/action pair is denoted as (s, a), where a is an action that can legally be taken from state s. Each item in this set can be used as part of a state-transition to a resultant state $T(s, a, s')$. Due to the nature of these transitions, the whole system can be drawn as a directed graph known as a state diagram (see figure 2.1).

2.2.3 Combinatorial Games

Combinatorial Games [4, 16] are classified by the following properties:

- *Zero-sum*: In the case of a two-player game, the rewards to all players sum to zero. In the case of a one-player game this property is always considered to be true, as

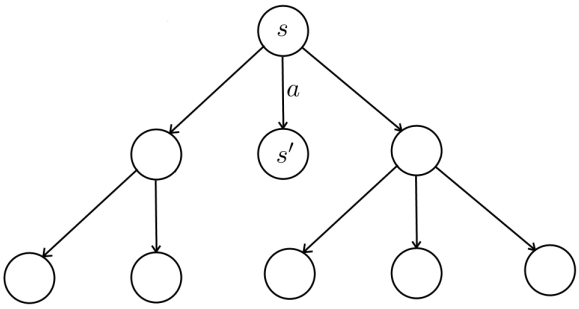


Figure 2.1: Example State Diagram for Formal Game Definition

the player can be viewed as competing against a game agent which will "lose" if the player wins. Zero-player automata can also be considered combinatorial games, but no games with greater than two players are combinatorial games.

- *Perfect Information*: The game state, all information about available moves from that state are fully viewable to all players, and the complete knowledge of the game's history.

- *Deterministic*: The outcome of all available moves from the current state is visible to all players, and is also completely predictable to all players.

- *Sequential*: Player actions are selected and applied sequentially, not simultaneously.

- *Discrete*: Player actions are applied discretely, not continuously.

Examples of Combinatorial Games include *Nim*, a strategy game in which two players take turns removing objects from heaps [23], and *Sylver Coinage* a game in which players name numbers, and the first player to name 1 loses [67].

2.2.4 Games of Imperfect Information

It is common for games to have game states in which the information is only partially viewable to players as part of the course of the game. For example, in most card games, players

have a hand of cards which opponents cannot view. This information may be viewable by some players, or hidden from all players. A game that has any such information hidden from one or more players is known as a game of *Imperfect Information*. The information may become revealed later in the game, at game end, or never (such as a player's hand in the game of Poker [134].) When creating a playout for games of imperfect information, *Information Sets* are often used [86]. An information set is a collection of all game states which could be true for a given game state from the perspective of the root player (the player conducting the playout). Use of information sets is discussed further in subsection 2.5.5.

2.2.5 Nash Equilibrium

A *Nash Equilibrium* is a solution for a game involving two or more players, which represents a combination of player strategies such that no player can benefit by unilaterally switching strategies [116]. Every finite game has at least one Nash equilibrium [97]. Nash equilibria are used to analyse the behaviour of decision makers, by comparing their selected strategies with the strategies of the calculated Nash equilibrium for that game.

A well-used example is that of the Nash Equilibrium in the game *Prisoner's Dilemma* [104]. Prisoner's Dilemma is a game in which two criminals are arrested for the same crime, and each of them is placed in a separate cell with no way of communicating with the other. They are then each asked to betray the other in exchange for a lowered sentence. If both betray, then each of them will serve 2 years. If only one betrayers, that prisoner will be set free, but his accomplice will serve 3 years. If neither betray, both of them serve only 1 year. The rewards for each player are shown in figure 2.2.

Given the knowledge that best decision for the each individual player is to betray the other, the Nash Equilibrium for this situation is that they should both choose to betray. The Nash Equilibrium is particularly interesting for this situation, as it shows that while it would

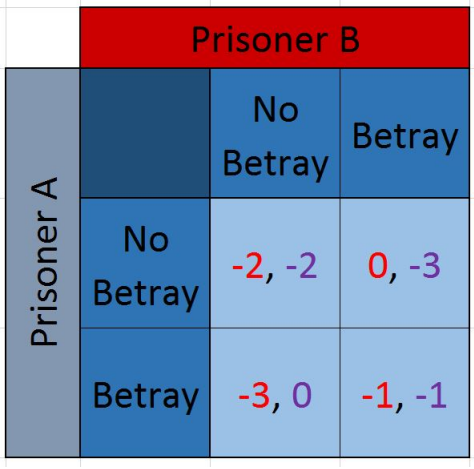


Figure 2.2: Example Payoff matrix for the game of Prisoner's Dilemma

certainly be preferable overall to both choose to not betray each other, each prisoner could then take advantage of the cooperation to improve their own individual circumstance by betraying the other.

2.2.6 Pure versus Mixed Strategies

A *Mixed Strategy* is the combination of multiple strategies with an associated a vector of probabilities, so that the strategies each have an associated probability of being chosen for any given action. In contrast, a *Pure Strategy* is simply the use of a single strategy for 100% of action choices [58].

The primary reason for using a mixed strategy is to create some level of unpredictability in play. This can be important for ensuring that an opponent is not able to determine your strategy by observing your action choices, however it should be noted that in some games, a Nash Equilibrium can be represented by a mixed strategy.

2.3 Basic AI techniques & Concepts

2.3.1 Classic AI Methods

2.3.1.1 Minimax Search

Minimax Search is a game tree search technique for use in combinatorial games. At each state, Minimax attempts to minimise opponents maximum reward, while maximising the player's potential gain. It considers that in a given state, the maximising player will always choose the best action for themselves, and the minimizing player will always choose the action that is worst for the maximising player. Traditionally minimax is only used to search partial decision spaces, as searching the complete space in most games can take an extremely large amount of time, and thus is not particularly useful for making good game decisions. Minimax search is often a strong choice for combinatorial games, as if provided with an unbounded budget, it will eventually locate the optimal strategy. Minimax search with alpha-beta pruning (see subsection 2.3.2.1) has endured for decades as the algorithm of choice [109] in positional board games such as Chess, Othello and Draughts, particularly when applied with alpha-beta pruning (see section 2.3.2.1. However in cases where the game tree is potentially very large and there is no reliable heuristic to aid the search, then minimax is unsuitable [29].

2.3.1.2 Expectimax Search

Expectimax Search, (also known as *Expectiminimax Search*) is a implementation of minimax search that adds a component for dealing with chance nodes. At chances nodes, the heuristic value of the node is equal to the probability-weighted sum of the heuristic values of it's children [68]. This allows evaluation of the "expected" value of the node. Expectimax

is only suitable for use in games which contain chance nodes, or for games that can be modelled in such a way [116]. Expectimax search performs well in games which contain rare or bounded chance moves, but are otherwise completely deterministic, such as Backgammon.

2.3.2 Move Pruning

Move Pruning describes the process by which a number of branches of a game tree are removed from consideration (hard pruning) or are de-prioritised from consideration, but may be searched later (soft pruning). Move pruning has been shown to be a powerful approach when applied with traditional minimax techniques [25], and has shown some strength when applied with MCTS [100].

2.3.2.1 Alpha-beta Pruning

Alpha-beta Pruning [116] is an enhancement to minimax search which allows removal of unpromising branches of the tree, while leaving at least one line of play that contains the best move from the current state. which in turn reduces the number of nodes evaluated and increases the search speed. Each state in the tree is evaluated using a heuristic, and α (the minimum score that the maximising player can obtain) and β (the maximum score that the minimising player can obtain) are evaluated. If at any point β becomes smaller than α, the branch is "pruned", as it cannot be a branch of optimal play, and need not be explored. Minimax search with alpha-beta pruning has long been the algorithm of choice for playing combinatorial games [109] such as Chess, Othello and Draughts.

2.3.2.2 Progressive Unpruning

Progressive Unpruning softly prunes nodes from the game tree using heuristic knowledge, but then progressively reintroduces them when the number of simulations reaches a certain threshold (Progressive unpruning is discussed further in subsection 2.5.3).

2.4 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) [36, 42, 82] is an adaptation of the standard tree search methodologies seen in more traditional minimax/expectimax AI, but also includes random sampling to increase the generality of the tree search and (in it's basic form) remove the need for heuristic knowledge (although such knowledge may still be exploited if available). By taking random samples of the decision space and using the results to guide the construction of a search tree, it is possible to locate optimal decisions for that decision space.

MCTS was first created in 2006, and has caused a great deal of further research and experimentation since that time. At the time of this writing, over 2,500 "Monte Carlo Tree Search" results are returned from Google Scholar search. It has been shown to be of novel application to games [35], and has seen much success in the field of Go [65], which proves challenging for more traditional AI techniques. More recently, an agent created by Google's DeepMind group used a combination of MCTS and Deep Neural Networks to become the first artificial player to beat a World Champion Go player in an even match [127].

2.4.1 Multi-armed bandit

A base consideration in MCTS is that of the *Multi-armed bandit* problem, which is the situation whereby a gambler must choose between a number of "bandit" gambling machines,

each which return rewards independently in an unknown random distribution. For example, one bandit machine might return a high reward 10% of the time, a medium reward 50% of the time, and a low reward for the remaining 40%. The ideal situation for the gambler is to maximise their return over time, so to play machines with the highest expected return. Given that the independent random distributions are unknown to the gambler, some time must be spent exploring the available machines. This is an example of the Exploration versus Exploitation dilemma [11]. Ideally the user would like to keep exploiting a seemingly rewarding bandit for as long as possible, hopefully scoring the highest result possible for as long as possible. However, initially there is little information about the other bandit machines, and exploring those machines may result in a far superior reward being discovered. The issue of wanting to play the best machine, but not being able to confidently determine which is the best machine is known as the exploitation-exploration dilemma.

Regret is defined as the expected loss due to not playing the best bandit [29] (i.e. not selecting the optimal decision). Regret is often used as a measure of the proximity to optimality of a decision (as the optimal decision would have a regret of 0). The problem is expressed as trying to minimise the total regret of a user repeatedly playing these machines. Regret is described as the loss due to not choosing to play the optimal bandit. As the underlying reward distribution of each bandit machine is hidden from the user, any reward must be estimated based on the user's experience of pulling that lever. Thus a policy can be used to attempt to obtain the highest reward possible from plays of the machines, and therefore to minimise player regret.

A policy which is designed to be applied to a multi-armed bandit problem is known as a Bandit Algorithm. Such an algorithm will make a decision on which arm to pull given past data from previous exploration of the available arms. These policies work by estimating the upper confidence bound of each machine, which is described as the highest expected reward from a given machine [29]. The upper confidence bound (UCB), proposed by Auer et

al. [11], is called *UCB1*, and is critical to many further developments in MCTS. UCB1 provides uniform regret growth over multiple plays, without any heuristic knowledge regarding the reward distributions. UCB1 is used as the bandit algorithm of choice throughout this work (see subsection 2.4.3 for more on UCT).

Monte Carlo Methods use sampling and statistical methods in order to estimate the strength of a given move. Abramson [1] showed that sampling actions in this manner might be of use to approximate the value of a given move.

2.4.2 Monte Carlo Tree Search

The term *Monte Carlo Tree Search* [36, 42, 82] describes a group of tree search algorithms first implemented in 2006. The basic principles involve building a game tree step-by-step, gradually adding tree nodes, and running playouts from leaf states at each iteration to determine the direction of further growth. (see figure 2.3) From these Playouts (also called simulations), a reward signal is received from the terminal game state, and the information is propagated upwards back through each parent node, modifying that parent node's value as it does so. The iterative growth of the tree is non-symmetrical, and controlled by a tree policy (a bandit algorithm) which attempts to balance exploitation against exploration by selecting potentially high reward nodes.

An extensive survey of Monte Carlo Tree Search and associated information was performed in 2012 by Browne et al. [29].

The basic MCTS algorithm is made up of 4 steps, and works as follows (see figure 2.3):

- Selection: The algorithm moves down through the tree until it reaches a node which has unexpanded children or a terminal node.
- Expansion: If the selected node has unexpanded child nodes, then one (or more) of those nodes are added to the tree.

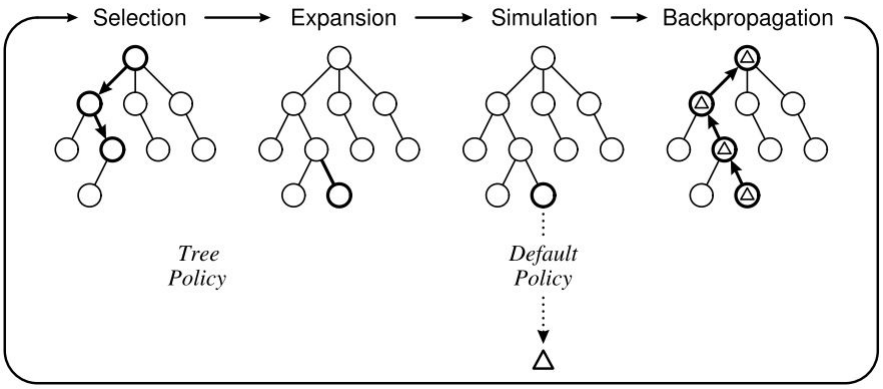


Figure 2.3: Overview of the typical MCTS process [29]

- Simulation: A simulation is run from each of the new child nodes, normally to a terminal state.
- Back-propagation: The simulation result is backed up through the parent nodes of the selected node, updating statistics until it reaches the root node.

During this process, the algorithm makes use of two policies; a tree policy and a default policy. These steps are shown in algorithm 1, with s_0 representing the initial search state, v_0 the root node, t_i the current iteration, t_{max} the iteration budget, v_1 the selected child, r_1 the rewards from the simulation conducted by the default policy, $v_1.s$ the state that is represented by the node v_1, and a is an action that leads to a specific node. The tree policy handles the selection of nodes during the selection of the algorithm, and also expansion if not every child of the selected node is being expanded. The default policy controls the algorithm's action selection during simulation. MCTS can be customised by specifying different algorithms for the tree policy and default policy. In most vanilla implementations of MCTS, the tree policy used is UCB1 (see subsection 2.4.3), and the default policy used is random node selection. This configuration of MCTS is most often called *UCT*.

Once the budget for tree simulation has been fulfilled, the best child node of the root is selected as the result of the best move. The two most common methods for selecting the best child at this point are either to select the child with the highest reward, or to select the

child with the most visits, but other options exist [29], and more are detailed in our work here (see chapter 5.)

2.4.2.1 MCTS Strengths

There are three main strengths of MCTS:

- Aheuristic: MCTS techniques do not require domain knowledge in order to function.
- Any time: All values are always up to date following every iteration of the algorithm, meaning that at any time in the process, the best move determined so far can be obtained.
- Asymmetric: Builds an asymmetric tree, so avoids building areas of the tree that are unpromising.

Aheuristic As vanilla MCTS uses no domain knowledge, it can in theory be introduced in an unmodified state to a new game and make somewhat sensible moves. It is important to note however that while heuristic knowledge is not required, it can greatly improve the strength of a MCTS player. (see subsection 2.4.2.2). A UCT player is a very strong opponent without heuristic knowledge in simple deterministic games such as Connect 4, Tic Tac Toe (Noughts and Crosses), and Checkers (Draughts).

Any-Time MCTS can produce a decision at any time during its operation, and the decision will be representatively strong of the budget consumed by that time. Each MCTS run produces a new "best" action, the MCTS process can be stopped at any point and the optimal decision (for that budget) can be retrieved. There is also an associated advantage that all nodes will have up-to-date internal values at every stage of the process, enabling analysis of

the MCTS process and also access to data on areas of the decision tree that are not selected for moving forward.

Asymmetric Tree growth in MCTS is asymmetric, as nodes are selected based on how promising they appear based on collected statistics from the MCTS process. This results in a smaller amount of budget being spent on areas of the decision space that are unpromising.

2.4.2.2 MCTS Weaknesses

There are four main weaknesses of MCTS:

- Agent Confusion: MCTS agents can become confused in complex "trap" states.
- Heuristic Agent Strength: MCTS may not behave as well as a heuristic agent specifically designed for the game in question.
- Lazy Play: MCTS can make poor moves if it determines that the current move is not "required" to be strong.
- Playout Bias: Bias in simulation from the forward model can greatly reduce MCTS performance.

Agent Confusion Browne et al. [28] demonstrated that MCTS agents can become confused and lead into trap states in certain situations. This is largely due to the simulations during the MCTS rollout being performed randomly, as it means that no consideration is given to the opponent's ability to move play away from certain states, and can direct tree growth towards lines of play that would never occur against an actual opponent. This weakness can be addressed by using a default policy that addresses this for the game in question, however this both requires heuristic knowledge, and likely slows the MCTS process, as ad-

ditional processing must occur during rollout (although heuristic knowledge can also reduce the number of rollouts required for a good solution.)

A *Trap State* is a game state where most moves will lead onwards towards a win, however opponent skill will drive the gameplay away from those wins and towards a nearly certain loss. As such, the state may appear very favourable to initial MCTS analysis, and might be chosen in error. Ramanujan et al. [108] suggest that MCTS/UCT is weaker than other techniques for such games (particularly Chess).

Heuristic Agent Strength If a decision space has a very strong heuristic agent, it is likely to be a stronger player than aheuristic MCTS [29]. This is due to the fact that heuristic agents can usually quickly prune a decision tree without performing any type of search. However heuristic agents are also very difficult to build for most complex games, and there are many domains in which aheuristic MCTS outperforms even heuristic agents designed specifically for those domains.

Lazy Play It has been shown that MCTS can be "lazy" in non-tight situations [7], which is to say that they can make suboptimal choices when they are not immediately required to make an optimal choice to receive a reward. This is a natural consequence of all search algorithms, not just MCTS, which are not incentivised to reach a quick decision, as there will be no consequence of including additional cyclic moves before the final desired state is reached. As such, an agent may decide to embark upon a pointless cycle of moves which have no lasting result on the game state, simply because it assigns no negative value to such a pointless cycle.

Playout Bias As MCTS relies on a relatively weak reward signal generated by simulation on the forward model, any bias in that forward simulation can greatly reduce agent performance [64].

2.4.3 Upper Confidence Bound applied to Trees (UCT)

Upper Confidence Bound applied to Trees (UCT) refers to the use of MCTS (as described in subsection 2.4.2) with a random default policy, using the specific tree policy known as *UCB1*.

2.4.3.1 The UCT Algorithm

UCB1 is a tree policy that treats the choice of a child node as a multi-armed bandit problem [82, 83], and selects the child node that has the most expected reward approximated by Monte Carlo simulations.

2.4.3.2 The UCB1 Equation

The UCB1 equation is shown in 2.1. During tree policy operation, this equation is used to evaluate each child node to determine which node the tree policy should select for expansion and simulation. The terms used in the equation are: $\bar{X}_i$ is the average reward from child node i (the node currently being evaluated), C is the *Exploration Parameter* or *Exploration Bias* (see subsection 2.4.3.4), n is the total number of visits to the parent node, and n_i is the total number of visits to the child node i.

$$UCB1 = \bar{X}_i + C\sqrt{\frac{2\ln n}{n_i}} \tag{2.1}$$

The UCB1 equation provides a balance between exploration and exploitation by scaling the number of visits to a given node against the rewards from that node's children. The term "Plain UCT" (or vanilla MCTS) is commonly used to describe using MCTS with the UCB1 algorithm and a default policy of random selection.

The two terms that make up the UCB1 equation are often referred to as the *Exploitation*

Algorithm 1 UCT Process Summary

function UCT(s_0)
  v_0 = new TreeNode(s_0)
  while $t_i < t_{max}$ **do**
    $v_1 \leftarrow$ TREE_POLICY(v_0)
    $r_1 \leftarrow$ DEFAULT_POLICY($v_1.s$)
    BACKUP(v_1, r_1)
    return ROBUST_CHILD(v_0).a

function TREE_POLICY(v)
  while v has children **do**
    if v fully expanded **then**
      $v \leftarrow$ BEST_CHILD(v)
    else
      $a \leftarrow v.s.GetRandomUntriedAction$
      $v' = newTreeNode(v.s.makeMove(a))$
      return v'
    return v

function DEFAULT_POLICY(s)
  while s is not leaf **do**
    $a \leftarrow s.GetRandomAction$
    $s \leftarrow s.makeMove(a)$
    return s.GetReward

function BACKUP(v, r)
  while v is not $NULL$ **do**
    $v.n \leftarrow v.n + 1$
    $v.q \leftarrow v.q + r$
    $v \leftarrow v.parentNode$

function ROBUST_CHILD(v)
  return $\underset{v' \in v.childnodes}{\arg\max} \bar{X}_i + C\sqrt{\frac{2 \ln n}{n_i}}$

Term and the *Exploration Term*. It's obvious that the exploitation term (shown in 2.4) grows larger as the average reward from a given child node increases (i.e. that node is shown to be more rewarding on average, so it should be exploited), and thus exploitation of valuable nodes is encouraged.

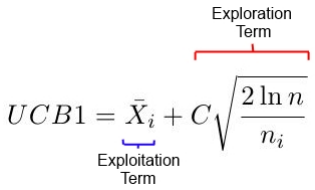


Figure 2.4: UCB1 algorithm used in UCT (MCTS), highlighting the Exploitation & Exploration Terms

The exploration term decreases as the child node i is visited, but increases as other sibling nodes are visited (i.e. other nodes with the same parent), encouraging exploration of nodes that are sibling to the nodes most visited. The UCB1 algorithm has been applied with much success in multiple other games such as Tron [101], Arimaa [85] and Go [87].

2.4.3.3 Optimality

With a large enough budget, UCT is optimal for any decision space. Kocsis and Szepesvári [82], [83] showed that for games of perfect information, when provided with enough iterations, UCT converges to the entire minimax tree. So as the iteration budget tends to infinity, the probability of selecting a suboptimal move tends to zero. This convergence, combined with the "Any-Time" nature of UCT, already makes it a strong candidate for many environments. It is worth noting that complete convergence is very slow, but any significant convergence has a substantial effect on play strength.

2.4.3.4 Exploration Parameter

The exploration parameter C can be altered to modify the bias towards exploration. As the value of C approaches 0, then bias towards exploration decreases (as the value of the exploration term with also approach 0). Kocsis and Szepesvári showed that a value of $C = 1/\sqrt{2}$ would satisfy Hoeffding's inequality [72] while all rewards were with the range 0-1 [83] and hence leads to an optimal decision choice in the limit.

2.4.4 Flat Monte Carlo

Flat Monte Carlo (also known as *Pure Monte Carlo*) describes any Monte Carlo approach in which the moves of a given state are sampled uniformly rather than guided by a default policy, and no tree building occurs. For example, if the root state had 10 different moves available, a flat Monte Carlo approach would send 1/10th of the total simulations down each of those branches, and not construct a tree [7]. Althöfer also describes a "basin" effect seen in win rates when increasing number of simulations, which suggests that an increase in simulations may actually yield a poorer result at very low numbers of simulations.[8].

2.4.5 Counterfactual Regret (CFR)

Counterfactual Regret (CFR) [80, 147] is an algorithm for calculating Nash equilibria in two player, zero-sum games. It operates by running sample games using selected strategies for each player, then calculating the regret of the players, and running another game with a modification to each strategy that is expected to cause a decrease in regret against the selected strategy of their opponent. If a sufficient number of iterations are performed, then the player's strategies approach equilibria.

2.5 MCTS Enhancements & Variations

2.5.1 Perfect Information Monte Carlo (PIMC)

Perfect Information Monte Carlo (PIMC) (also known simply as *Determinization*) is a technique for handling games with imperfect information and/or stochasticity (non-determinism). It operates by by sampling random instances of the equivalent deterministic game of perfect information [143].

For example, a determinization of Magic: The Gathering [128, 146] would be an instance of the game that reveals all player hands and all shuffled decks. Determinization provides a manner in which to handle stochasticity/imperfect information in games using AI techniques developed for deterministic games.

Ginsberg produced a PIMC agent for Bridge which was at the time the strongest AI in the world for that game, and dominated the World Computer Bridge Championships in 1998 and 2000 [66]. PIMC approaches have also been used with outstanding success in Scrabble [126].

Flat Monte Carlo would seem to be a strong contender for games where there is a plausible (if expensive) perfect information variant, and it is reasonable to assume that opponents are playing with perfect information [66]

2.5.1.1 Weaknesses

While this determinization is a powerful tool, it has two main weaknesses (discussed below).

2.5.1.1.1 Strategy Fusion Determinization exhibits a weakness known as *Strategy Fusion*. The term strategy fusion describes the set of errors caused when strategies that are effective in different determinized versions of the same game state (or "worlds") are com-

bined in an attempt to work in the actual game state, but the actual game state is inconsistent with the determinizations [90].

This type of error is illustrated in figure 2.5, where the upwards pointing triangles represent "our" moves, and the downwards pointing triangle represent opponent moves. The squares represent terminal states that grant rewards as noted below, depending on whether we are actually in world 1 or world 2. Starting at the top node, the correct strategy if we exist in world 1 is to play towards action a, and the correct strategy if we exist in world 2 is to play towards action b. At the root node however, the expected value of the choices between a, b & c all appear identical, so a move towards a & b may be erroneously selected, leading to a negative reward.

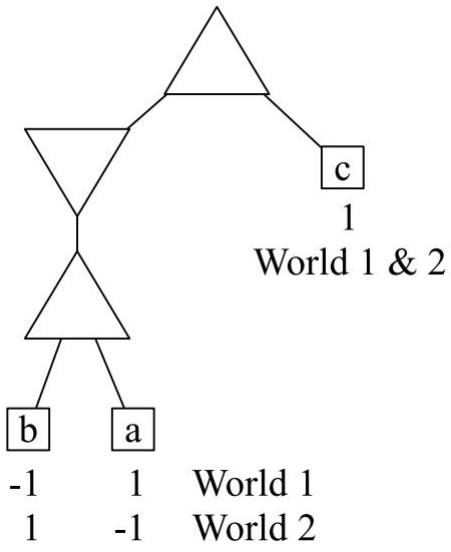


Figure 2.5: State diagram displaying an example of Strategy Fusion [90]

2.5.1.1.2 Non-Locality *Non-Locality* describes the set of errors caused by information to which an opponent has access but the player does not. An opponent with this information advantage will steer away from some areas of the tree, but this is unknown to the player from the current game state [90].

This type of error is illustrated in figure 2.6. The top node is a chance node of which

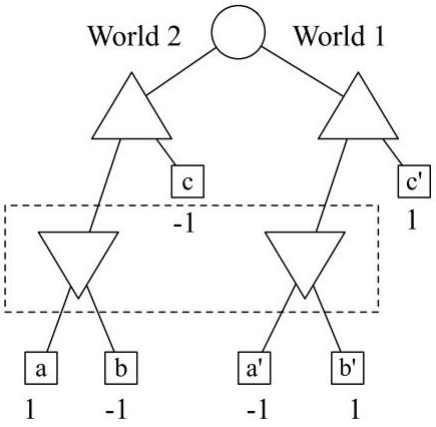


Figure 2.6: State diagram displaying an example of Non-Locality [90]

the maximising player (represented by the upwards pointing triangle) knows the result. As the minimising player (represented by the downwards pointing triangle) does not know the result of the chance node, they cannot distinguish between the two worlds shown in the dotted rectangle, and has no way to decide between them. In this case, the minimising player would likely decide between these two worlds randomly.

This type of error is similar to the "Agent Confusion" error experienced by UCT in general (see section 2.4.2.2), except that the misunderstood factor is not opponent's skill to play away from certain situations, but rather opponent knowledge of the actual game state.

2.5.1.2 Inference

Inference is the process of deriving logical conclusions about the actual nature of hidden information from observing opponent actions. Whitehouse et al. [143] showed that *Inference* could be used to reduce the effect of the weaknesses of determinization, specifically non-locality, by eliminating certain determinizations from consideration in response to certain opponent actions.

One method of incorporating inferences is to use *Belief Distributions* [141], probability distributions which describe the likelihood of the actual game state existing in the associ-

ated worlds. These probabilities are constructed from the history of observed agents moves. Belief distributions have been frequently used in games such as Poker [115, 102], Scrabble [111] and Skat [119, 32].

2.5.2 Transposition Tables

Previously visited positions in the game tree are sometimes identified by storing them in a *Transposition Table*, in order to handle situations where the same state can be reached by several different lines of play. This technique is used frequently in chess [27] and occasionally in Go [26].

When transposition tables are used during tree building, the tree can instead be considered a directed acyclic graph (DAG), and thus can be reconstructed with multiple inbound connections to the same node.

Childs et al. [39] identify four different variants for observing transpositions when using UCT:

- Exclude detection of transpositions (i.e. use unaltered UCT).
- Share cumulative visitation and valuation statistics between identical states independent of where they occur in the tree (UCT1).
- As UCT1, but use a refined value estimate for move selection of the parent (UCT2).
- As UCT2, but also use cumulative visitation and valuation statistics to evaluate the parent node (UCT3).

Childs et al. [39] report that of the four variants, the UCT3 seemed the most promising in artificial tree and Go experiments, followed by UCT2. However they also note that UCT3 is only applicable when time required for simulating games dominates the time required for updating the statistics collected in the nodes.

2.5.3 Progressive Strategies

Due to the exploratory nature of MCTS, initial iterations are often poorly guided, as insufficient information has been gathered to determine the strength of a line of play. On the other hand, knowledge dependant agents tend to suffer from a weak global sense due to the common approach of dissecting the global game state into smaller local problems. Techniques that combine heuristic knowledge with Monte Carlo simulations have been shown to somewhat address the weaknesses of each separate techniques, and have been shown to work well in complex games such as Go [24].

Progressive Strategies initially use heuristic knowledge as a tree policy, then perform a soft transition to using a non-heuristic tree policy [38]. Often the strength of a move calculated based on the previous strength of a given move [99]. Two strategies suggested by Bouzy [24] are to attempt running the MCTS simulated games using knowledge based agent, or to allow a knowledge based agent to pre-process the moves available, and hand the top candidates to MCTS for simulation. This is further expanded by Chaslot et al. [38], who defines two progressive strategies for applying domain knowledge to an MCTS agent, Progressive Bias and Progressive Unpruning.

Progressive Bias [38] uses an agent with domain knowledge as the tree policy for initial searches, with the agent providing less influence as more games are played through. As the agent loses influence, the non-heuristic tree policy takes dominance. *Progressive Unpruning* [38] describes a process by which child nodes are added as normal to any node p in the MCTS tree until the number of games n_p in a node equals a predefined threshold T. At this point, a large number of child nodes are artificially pruned, and with each further game that plays through p, these moves are slowly "unpruned" (made re-available for selection & simulation), by an unpruning policy. A technique described by Chaslot et al. [38] in their Go program *MANGO*, unprunes branches from a given node when a specific number of

simulations has passed through that node. This ensures the initial simulations are guided, but allows for additional exploration once certain playouts have been explored. Progressive unpruning has been shown to be very effective, particularly when combined with Progressive Bias. Progressive unpruning is very similar to a simultaneously proposed scheme called *Progressive Widening*, proposed by Coulom [41].

Progressive History [99] is a combination of progressive bias and a *History Heuristic* [145] used in enhancements such as RAVE (subsection 2.5.4.1). The history score of a given move is calculated by determining the strength of that move in any position throughout simulation. The history score is then combined with the UCT selection strategy, to estimate a value for the move.

2.5.4 Value Approximation Techniques

2.5.4.1 All Moves As First (AMAF)

The *All Moves As First* (AMAF) heuristic [30] is a general optimisation technique which looks at the value of each move independently of the game state into which the move is played, basically asserting that good moves are almost always good, and bad moves are almost always bad. It is important to note that this varies by game, and is normally a more valid assertion for positional games which have immovable pieces, such as Go and Hex, which is exactly where it has had most success [10, 64]. This technique was applied to MCTS independently by Drake & Uurtamo [54] and Gelly & Silver [62].

Last-Good-Reply (LGR) is an enhancement that modifies the MCTS simulation policy. During simulation, moves are chosen based on their suitability as a reply to the previous move, based on the results of previous Playouts [13, 133, 53]. Specifically, moves are assigned a value based on their effectiveness as a reply to the last move (or set of moves)

made by an opponent, effectively considering the move's strength as a reply rather than its direct effect on the game state.

Move-Average Sampling Technique (MAST) [106, 21] and *N-gram Average Sampling Technique (NAST)* [57, 106, 137, 133] are effectively extensions of the concept of LGR, such that instead of recognising the value of a single move in reply to the previous one, it recognises the value of a set of N moves, or of a move as a reply to the previous N-1 moves.

Normal operation of MCTS estimates the value of an action by averaging the value of all lines of play in which the action is immediately selected. *Rapid Action Value Estimation* (RAVE) is a modification of AMAF which instead averages the return of all lines of play in which the action appears at any point prior to when it was actually played [62, 63]. In this way in bears a strong similarity to MAST.

2.5.4.2 Hindsight Optimisation (HOP)

Hindsight Optimisation (HOP) is a technique which allows use of a deterministic planner in a stochastic environment. HOP makes multiple calls to the deterministic planner using different determinizations of the game, and combines the values of those calls to create an estimate for the value of the stochastic action [20].

2.5.4.3 Macro-Actions

In cases where search algorithms are required to operate in continuous time, a common technique is to use searches to create macro-actions which consist of a sequence of smaller actions intended to bridge the gap before the next search can be completed. This technique was employed by Powley et al. [105] to create an MCTS controller which could generate sequences of actions and thus allow the controller to operate in a continuous environment.

2.5.5 Information Set MCTS

Information Set MCTS (ISMCTS) [44, 106] is an enhancement to MCTS for making decisions in games of imperfect information. ISMCTS eliminates strategy fusion (see subsection 2.5.1.1.1). An *information set* is a collection of game states that could be the true state of the game from the perspective of the observing player at a particular point in the game. By collecting many similar states into sets, the game tree is vastly simplified. For example, in a card game where an opponent has a hidden hand of cards, the player's information set would be every game state which corresponds to all possible combinations of the opponent's hidden cards (note that depending on the game in question, these information sets may not appear with uniform distribution.)

ISMCTS operates as vanilla UCT MCTS, uses determinized games during simulation, but does not cheat. A random determinization is used for each simulation, effectively creating a search across a large number of possible combinations of hidden information. ISMCTS is currently implemented in the successful commercial mobile game, Spades [142], and has also seen much success in other similar games such as *Dou Di Zhu* and *Lord of the Rings: The Confrontation* [44].

2.5.6 Genetic Programming in MCTS

A number of attempts have been made to evolve a good MCTS player using genetic programming. EvoMCTS [17] is an enhancement to MCTS that uses genetic programming techniques to enhance the level of play. It was originally proposed for Reversi, and has been shown to outperform vanilla MCTS agents.

Alhejali & Lucas used genetic programming to evolve heuristics for playing Ms Pac-Man [5], with some success, but encountered problems performing game runs in a reasonable time and were forced to reduce the number of evaluations below the number recom-

mended by genetic programming experts [144].

2.5.7 Parallelization of MCTS

As MCTS is a search technique, parallelizing MCTS can allow for an increase in overall strength, as a wider and/or deeper search can be performed in a shorter time. Cazenave et al. [33] suggest three different methods; Single-Run Parallelization, Multiple-Run Parallelization and At-the-leaves Parallelization (see figure 2.7).

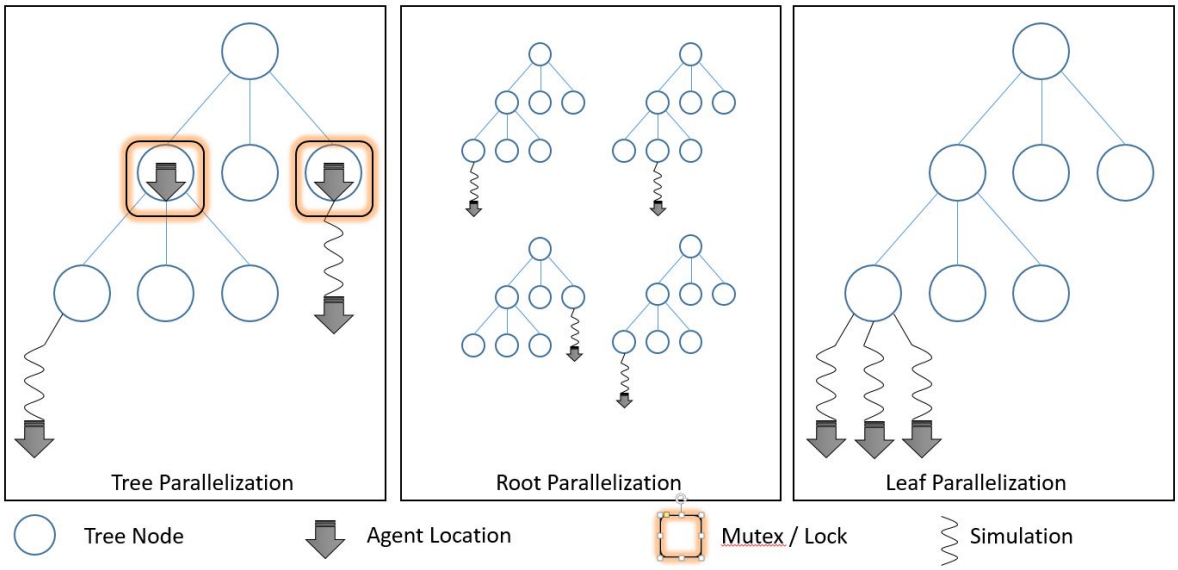


Figure 2.7: Overview diagrams for the most popular three parallelization methods

These have been provided more concise names by Chaslot et al. [37], who named Single-Run Parallelization as Root Parallelization, reflecting the complete parallelization of the MCTS from the root, and renaming At-the-leaves Parallelization as simply Leaf Parallelization. Chaslot et al. [37] also provides another method named Tree Parallelization.

2.5.7.1 Root Parallelization

Root Parallelization [33] (also known as slow tree parallelization or Single-Run Parallelization) describes the process by which each system runs a separate MCTS from the same game state, then the results are amalgamated by a master process. As each tree would have a different random seed, different results will be generated in each tree.

There is also some consideration to be given to the method of data combination. Bourki et al. [22] suggest an amalgamation policy which filters some outlying information. According to this policy, information is only included if it came from a node which received at least 5% of the simulations which passed through its parent, thus removing nodes which are not thoroughly explored. This technique proved effective in reducing the noise in the final search.

2.5.7.2 Multiple-Run Parallelization

Multiple-Run Parallelization [33] is similar to Root Parallelization, except the amalgamated tree is then sent back out to the slaves for further independent processing. This cycle can repeat multiple times until a budget is reached.

2.5.7.3 Leaf Parallelization

In *Leaf Parallelization* [33], the main process is run on a master machine, but all simulations are passed out to the other systems for processing.

2.5.7.4 Tree Parallelization

Tree Parallelization [37] (also known as fast tree parallelization) uses a shared tree to which all threads read and write information. Mutexes are used to lock parts of the tree that are

currently being operated upon. There are two styles of Tree Parallelization, one which locks the entire tree with a mutex when any write operation is in progress, and another which attempts to dynamically lock specific parts of the tree which are being written.

2.5.7.4.1 Virtual Loss During Tree Parallelization, in order to prevent different processes selecting and exploring the same node, a *Virtual Loss* may be assigned to nodes which are currently being processed by other threads, in order to make them appear less valuable for selection. After the node processing has finished, these losses are removed. Chaslot et al. attributes this concept to Coulom through a personal communication [37].

2.5.7.5 Strength of different parallelization techniques

Of the the established three parallelization processes described here, Cazenave et al. report that Root Parallelization seems the most preferable [33], as the results are comparable to Multiple-Run, but implementation is far simpler. Chaslot et al. [37] report from their experimentation on Go that Leaf Parallelization seems a poor method, as using 16 agents results in only a strength increase equivalent to that of serial MCTS with only 2-4 times as much CPU time.

They acknowledged Root Parallelization as the stronger technique, but also stated that Tree Parallelization with Virtual Loss performs as well on smaller Go boards. This is supported by Bourski et al. [22], however Mehat et al. [93] later contested this, stating that Tree Parallelization showed improved results, and that the improvement is related to the ability to keep all threads consistently busy.

2.6 Entertaining Play Experiences

The games industry in 2013 was worth an estimated $93 billion USD and was predicted to grow to $111 billion USD in 2015 [61]. This industry represents world interest in receiving entertainment through interactive game play, and indicates the importance of entertainment industries. A similar increase of interest in entertainment is occurring in research, as shown by the founding of new research groups (DC Labs[1]) and PhD programs (IGGI[2]).

2.6.1 Making more human-like moves

There has been a limited amount of research into creating human-like players for a number of games, and a number of competitions awarding a cash prize exist for success. These competitions include a tournament evaluating the "naturalness" of moves in Go [77], a second Go tournament which evaluates the "humanity" of moves [78], and a Super Mario Bros. tournament evaluating the "human-like" natural of an players moves [125].

The original test of the "humanness" is the famous Turing test, which has been one of the most disputed topics in artificial intelligence since its statement in 1950 [117, 140]. The original proposal is that a artificial agent would pass the test if it could be mistaken for a human after 5 minutes of conversing in natural language (the original specification stated that a 70% success rate is sufficient to pass.) One of the notable weaknesses of the Turing test however, is that some human behaviours and unintelligent, and some intelligent behaviour is inhuman. This explains some of inhuman from strong AI players, and also leaves us in the realisation that simulation of human play is likely significantly different from simulation of strong play, and also significantly different from simulation of entertaining play (i.e. play designed to entertain a human opponent.) There have been promising attempts at passing

[1] http://digitalcreativity.ac.uk/
[2] http://www.iggi.org.uk/

the Turing test within specific gaming environments, however it is important to note that these attempts are often using variations of the Turing test as they are normally directed at gameplay rather than natural language [69].

Making human-like moves is a very challenging and complex concept, because defining the human-like qualities of a move is very difficult [139]. There are currently no established parameters to measure the human-like nature of a game move [59]. One key area in which AI fails is in an attempt to be "believably weak". A common route to making an AI weaker is to reduce its time budget to make decisions, however this often has the side-effect that it frequently misses selecting moves which would be obvious to even the weakest human players. Effectively it fails to be believably weak, which makes it appear non-human. Rather than simply reducing time budget to create a weaker AI, the budget should instead be used to select a believably weak move. This further requires some categorisation of what qualifies as a believably weak move, and how a search can be tuned to generate these moves.

2.6.2 Entertaining Play

In this context, "entertaining" refers to a combination of fun, creativity and interest in a move. Simulating this experience in general has long been the subject of research [48]. Schmidhuber has defined a formal definition of creativity and fun [120]. Schmidhuber's theory is that the interest or fun associated with a given piece of information is related to the compressibility of that information. If the information is very easy to compress, then it appears simplistic and uninteresting, if it's too complex, then it appears confusing and is difficult to understand or remember. Information that is in the correct range of compressibility is both easy to remember (and possibly difficult to forget), and also challenging enough to prove interesting.

In 2005, Sweetser et al. collected various heuristics into a system called *The GameFlow*

Model in an attempt to better model the enjoyment a player experiences when playing a game [135]. The model contains eight different elements, each of which provide a set of criteria for classifying the effectiveness of a specific game. While useful in categorising and clarifying the concepts that contribute towards play experiences, it is unclear whether any direct benefit was obtained, as the work experienced difficulty in achieving practical results, and was limited to providing guidelines for further work.

There is also research on using heuristics to evaluate the utility of a game design [51], however it is worth noting that these techniques all apply to classifying the player experience due to the game design, and not specifically the opponent play.

It is also worth considering whether an agent which maintains a constant 50% win rate against a human player is operating as an entertaining agent. If we remove all other concerns, the it could be seen as the agent providing a competent challenge which is well matched to the human player. However, depending on the style of play or the human player's understanding of the agent's actions, it could actually be seen to be antagonising the human player. An agent which you can never truly improve against, and always maintain an even win rate is possibly highly annoying over a long period of time, as the human player may see no long term improvement in their record.

2.6.3 Human-like play

Complex play from artificial agents has lead to research into how they can be made to appear more human in their play [59, 15], largely due to the assumption that human-like play is desirable from an opponent. As discussed in section 2.6.1, we recognise a distinction between human-like play and optimal play, as some non-optimal play is human-like. It is likely that in specific scenarios we desire unintelligent play from our opponents most notably tutorials or introductory levels, it seems unlikely that regular unintelligent play is

desirable, but rather that we would like to experience play that meets our own expectations of intelligence and human-like behaviour.

Ikeda & Viennot recently studied Go with intent to create a more human player [76]. They had an excellent approach to dissecting the game play, starting with building an opponent model to assess not only opponent strength, but also the opponent's expectation from the game (i.e. what that opponent would find entertaining). They also discuss playing *Gentle Moves* when facing a less skilled opponent, which is a suboptimal move chosen to avoid a devastating defeat of an opponent. Their *Gentle play* variant showed promising results when compared against a simple reduction of thinking time. We can understand a clear link between gentle moves and human-like behaviour by considering a benevolent teacher playing against a student. The teacher may make gentle moves specifically to coach the student through the initial stages of gameplay, and thus this experience, if recognised by the human opponent, may appear human-like, however it is dangerous to assume such recognition, as it may be mistaken instead for unintelligent play.

More recently, Devlin et al. demonstrated combining gameplay data with MCTS to emulate human play [52]. The research focused on collected gameplay data from the top rated Spades game in the Google Play store (AI Factory Spades[3]). Previous research had shown that the adversarial agents in AI Factory Spades behaved substantially differently to the human players [142, 43], and work be Devlin et al. has shown a successful technique for biasing towards human-like behaviour.

2.6.4 The importance of challenge to an entertaining experience

Level of challenge is an important part of deriving enjoyment from a game, as players generally enjoy a level of resistance in gameplay that they can defeat before claiming victory [45].

[3] `http://www.aifactory.co.uk/AIF_Games_Spades.htm`

While different players find different levels of challenge enjoyable due to different skill levels and different tastes, it is fair to say that a complete lack of challenge or an unbeatable challenge are undesirable scenarios for the majority of players [84, 75]. Achieving a balance when directly manipulating win rate can be challenging however, as the player can feel like they are receiving a diminished play experience due to artificial constraints on win rates forcing AI losses [49, 43].

2.7 Opponent Modelling

The term *Opponent Modelling* refers to the process of creating an agent which can simulate opponent knowledge and decision-making. In recent years, this approach has become more frequently adopted, as digital games are more capable of capturing massive amounts of statistical play information from players. This easily-accessible wealth of information can then be used to generate or train opponent models.

Opponent models are generally used to predict the action an opponent would take in a specific game state. As such, the accuracy of an opponent model is of critical importance, as regular incorrect predictions could have an a significant effect on an agent's play strength.

The importance of opponent modelling is particular to each individual game, for example opponent modelling is has high value in card games which use bluffing and manipulation such a Poker, but has lesser effect upon deterministic perfect information games such as Chess [19].

A lot of opponent models use statistical techniques to predict moves based upon a player's previous moves. A number of statistical models have been proposed which use opponent play information to dynamically adapt and to take advantage of opposing agent behaviour [18, 118].

More recent contributions include a Bayesian model for opening prediction in the real-time strategy Starcraft [136]. This agent predicted a specific opening based upon the opposing agent's past behaviour, and played to counter that opening. Dereszynski et al [50] used a similar model to learn high-level strategy in the same game, and demonstrated an improvement in player strength and strategy selection.

2.8 Data Mining Techniques

Data Mining is a Machine Learning (and thus Artificial Intelligence) technique by which knowledge is extracted from collected data [6]. Due to this, it is occasionally historically referred to as *Knowledge Discovery from Data* (KDD). The main focus of research is to create rules of patterns which exist in the data, and then use those rules to provide analysis of the data, or to allow for prediction of future data from the same system.

2.8.1 Data Preparation

Before analysis on a specific data set can be performed, there is usually an amount of cleaning and preprocessing to be performed upon the data. This may be as simple as ensure the data is received in the correct form, or a more complex process which involves replacing missing values and/or excluding certain entries from analysis [79]. There are also occasionally techniques applied to reduce the data in order to speed analysis. These techniques can either function as a form of pre-analysis, or as an advanced form of Data Cleaning [81].

Analysis of a data set in Data Mining begins by determining the *attributes* of the data which are of interest in determining rules. These are typically attributes that are of interest for analysis and/or show a statistical trend in the data. Additional attributes may be included in order to determine their significance in creating rules for the data.

2.8.2 Association Rule Mining

Association Rule Mining is the determination of correlations between a set of items, and then the subsequent creation of rules which describe the data [2]. It is also known as *Market-Basket Analysis*, due to the common usage of determining which products a shopper may purchase based on what is already in their shopping basket. A typical rule-mining algorithm functions by generating rules that describe which items are likely to be included in a partially observed set, given the items in the observable part of the set. Itemsets are drawn from the data such that each itemset describes a correlation between items. Association rule mining is employed in many application areas, including intrusion detection [89], web usage mining [132] and bioinformatics [46].

2.8.3 The *a priori* Algorithm

A commonly used algorithm in association rule mining is *Apriori* [3]. Apriori first generates all *1-itemsets* that appear in the data at least a number of times equal to a predetermined support value, then passes this generation onward to create a second generation of *2-itemsets*. This process continues until an empty generation is found (that is a generation with no candidates that appear at least *support* times in the data.) Each generation member then creates a single association rule of the form which describe the correlation recognised by that member. Our rules take the form of $S \rightarrow c$, where S is a multiset of antecedents, and c is the consequent. The Apriori algorithm is shown in figure 2.8

There are many variations on the Apriori technique to generate rules [70], most notable of these are a technique which attempts to identify the n-most interesting itemsets by characteristic for rule generation rather than using a minimum support value [98, 60]. More recent contributions use functional languages rather than support constraints [71, 74], generating substructures within the available data in order to determine the likely most popular rules,

```
1)  L_1 = {large 1-itemsets};
2)  for ( k = 2; L_{k-1} ≠ ∅; k++ ) do begin
3)     C_k = apriori-gen(L_{k-1});  // New candidates
4)     forall transactions t ∈ D do begin
5)        C_t = subset(C_k, t);  // Candidates contained in t
6)        forall candidates c ∈ C_t do
7)           c.count++;
8)     end
9)     L_k = {c ∈ C_k | c.count ≥ minsup}
10) end
11) Answer = ⋃_k L_k;
```

Figure 2.8: Apriori Algorithm [3]

and thus speed the generation of all rules.

2.8.4 Data Mining for Games

Data Mining has found a number of applications in games, principally in application to attempting to determine patterns in player behaviour, and in modification of AI agents in response to such behaviour. In this area, dynamic game AI has been shown to be applicable and somewhat affective in modifying game behaviour to match player expectations [130, 131].

2.8.4.1 Dynamic Difficulty Adjustment

The concept of *Dynamic Difficultly Adjustment* (DDA) in response to player behaviour has become popular over the last decade [47]. Hunicke & Chapman created a Dynamic Difficulty Adjustment system in order to provide online tuning of opponent difficulty in Valve's *Half Life* game engine [75].

More recently, Missura & Gartner created a complete system for implementing difficult adjustment depending on player behaviour [96]. They categorised players into different

types, determined a local difficulty modifier for each type of player, and then applied difficult adjustments to subsequent games after determination of the type of player.

2.9 Summary

In this chapter, we have presented a survey of literature relevant to our work in this thesis. Firstly we established all terminology relevant to discussing games as a scientific discipline, then AI concepts with a specific focus upon MCTS. We then moved on to literature relating to entertaining play experiences, and finally provided an overview of Data Mining Techniques which are relevant to our work here.

Chapter 3

Game Domains and Experimentation Software

3.1 Game Domains

The following Game Domains were selected for experimentation.

3.1.1 Lords of War

Lords Of War[1] is a two-player strategic card game by publisher Black Box Games. A board is used for card placement, and the relative positions of the cards on the board are the main strategic interest. A player wins when they eliminate twenty of an opponent's cards, or they eliminate four of their opponent's *Command Cards*. Command cards are significantly more powerful than other cards, but placing them onto the board carries a risk that they may be eliminated.

The game board is 7×6 squares each of which can hold a single card. Cards have between 0 and 8 attacks, each with a strength value, and a directionality towards an orthogonal or diagonally adjacent square (see figure 3.1). Attacks from multiple cards can be combined to eliminate an opponent's card with a high defence value. Some cards also have ranged at-

[1] `http://boardgamegeek.com/boardgame/135215/lords-of-war-orcs-versus-dwarves`

Figure 3.1: Bestial Raptor, an example card from the Lords of War game.

tacks which can eliminate (or contribute towards the elimination) of opponent's cards which are not adjacent. In regular play, cards can only be placed so as to attack enemy cards, however *Support Cards* also have additional placement rules allowing them to be placed next to friendly cards instead of attacking enemy cards.

On each player's turn, they are required to place exactly one card, then process combat to identify and remove eliminated cards, then they have a choice of either drawing a new card from their deck, or retreating a friendly unthreatened card from the board. The complete rules of Lords of War appear in Appendix C.

A normal game rarely extends beyond 50 turns, as most moves (particularly strong moves) result in a capture. Once an average human player has made 25 moves, they have probably captured more than 20 cards, and thus the game would have completed. Of course the games can end much sooner if command cards are placed carelessly or last much longer if players play cautiously. Games with MCTS agents last on average between 30 and 60 turns, depending on the nature of the agent. Games using random moves vary wildly in

length, but normally last between 50 and 120 turns. Our experience with Lords of War has revealed that it commonly has a mid-game branching factor of 25-50, making move selection challenging.

Lords of War was chosen for research as it represents a complex game in which players can adopt a number of different strategies and play styles, which provides a wide range of behaviour for us to analyse.

3.1.2 Android: Netrunner

Android: Netrunner is a two-player strategy card game published by Fantasy Flight Games[2], which includes elements of bluffing and deception. Netrunner is similar to other popular card games such as Magic:The Gathering, and is described as an LCG (Living Card Game [55]).

During a standard match of Netrunner, opponents do not have access to the content of their opponents deck. Access to such information would provide a substantial advantage to a player, as they would both be able to predict their opponent's likely strategy, and also determine which strategies they are poorly defended against.

Due to the nature of the game, the content of an opponent's deck is critical strategy information, and a player who is able to accurately model their opponent's deck is at a substantial advantage. There are currently more than 600 cards released for Netrunner, so accurately modelling a deck is a significant challenge. The combination of the wide number of choices, plus the complex and specific rules for which cards may be included in decks makes Netrunner deck construction a highly intricate process.

Netrunner has a well documented rules structure for deck building. Every Netrunner deck has exactly one *Identity* card which defines some rules for that deck, most notably

[2] `http://www.fantasyflightgames.com`

a *Side*, an amount of *influence* and a *Faction*. There are exactly 2 sides (named *Runner* and *Corp*), and each card in Netrunner is associated with one side and cannot be included in decks associated with the other side. Identities which are from the corp side must also include a specific number of *agenda points*, which are provided corp cards (the specifics of agenda points are not relevant to this work, other than to recognise that there is a required number of agenda points for some decks to include, which presents an additional restriction upon decks.) All non-identity cards also have a *Faction* and a *Influence Cost*, the latter of which describes the amount of influence which must be paid to include the card in a deck which contains an identity of a different faction. The complete rules of NetRunner appear in Appendix D.

Netrunner was chosen as a target for research due to the large amounts of imperfect information, and that bluffing and deception are integral to play, meaning that the content of an opponent deck is of very high importance.

3.2 Experimentation Hardware & Software

The experimental MCTS engine and Lords of War game were implemented in C++ and all experiments were run on a Intel(R) Xeon(R) CPU E5645, with two processes (2.40GHz & 2.39GHz), each with 6 cores & hyperthreading and 32GB of RAM.

Much of the practical progress up to this point has been directed towards the creation of a MCTS engine. Due to the efficiency requirement of the experimentation engine, C++ was selected as the language of choice. The following work has already been completed on the C++ engine.

3.2.1 MCTSTree

The central part of the MCTS engine is the representation of the MCTSTree, which is contained within three classes; MCTS_AI, MCTSTree and MCTSTreeNode. The class diagrams of these three classes are shown in figure 3.2.

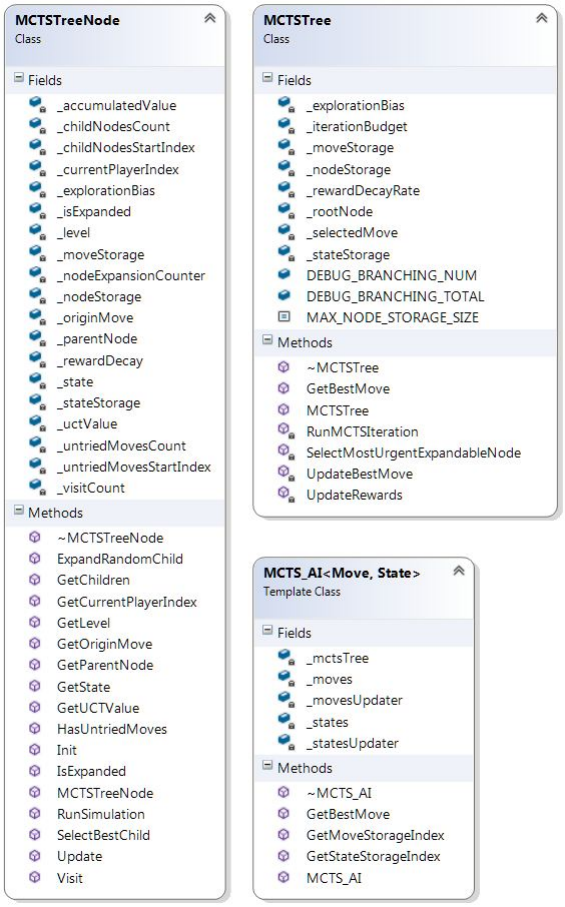


Figure 3.2: Class diagram showing the MCTS Tree class and associated sub-classes

3.2.2 Move & Game State interface

The MCTS engine uses abstract interface classes to allow games to make use of it without having access to the code. In order to use the engine, games must implement IGameState and IGameMove. The class diagrams of these are shown in figure 3.3.

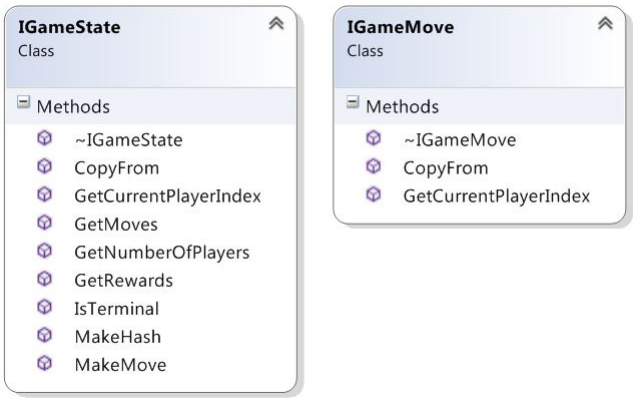


Figure 3.3: Class diagram showing the IGameMove & IGameState abstract interface classes

3.2.3 Memory Management

Any MCTS model can potentially build very large game trees, so it was critical to implement a strong memory management system, both for efficiency and for ensuring that memory associated errors or crashes were prevented. There were also a number of complications associated with creating a memory management system for an object of an unknown type, but that must subscribe to a specified interface. In the end, a total of three template classes were created to deal with this issue. A fourth simpler template class was added for managing objects which did not require subscription to a specified interface (most notably the MCTSTreeNode class). The class diagrams of these are shown in figure 3.4.

3.2.4 Engine Profiling

Initial profiling of the engine was performed using a simple maze search game. The player's piece was placed in the top left corner of the grid (0,0) and the target in the bottom right. Variable sized grids were tested, and the total time for the player's piece to reach the target was tracked. Each grid size was tested for 200 games. The graph in figure 3.5 shows the game completion time before the memory management system was implemented (woMM), after the memory management systems completion (wMM) and after second stage optimi-

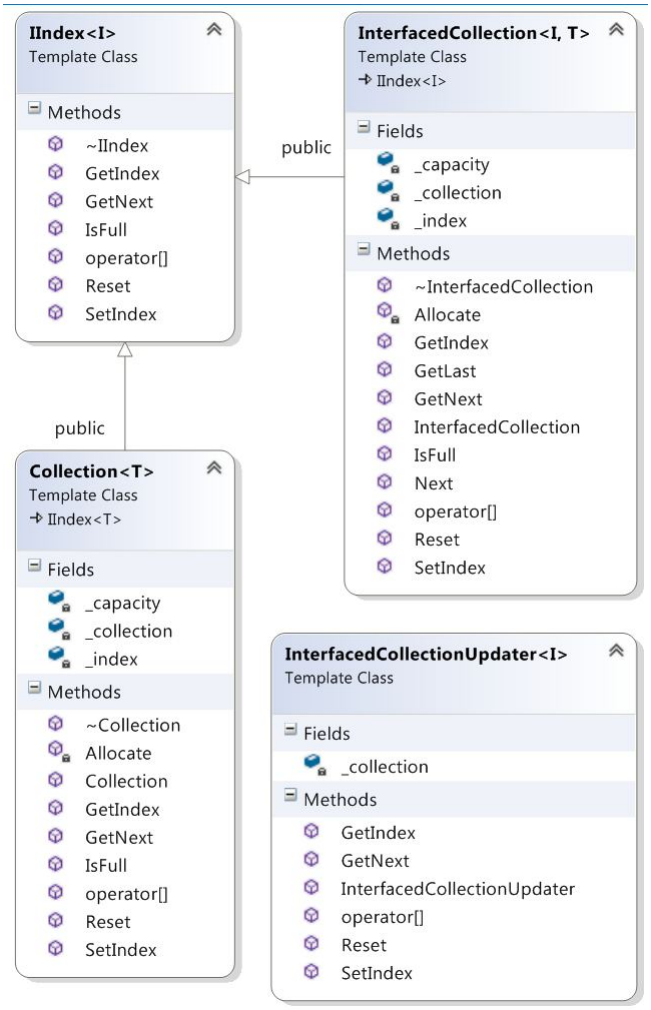


Figure 3.4: Class diagram showing the Memory Management classes

sation (wMM2).

We can see the substantial effects of the memory management system upon game completion time, particularly after second stage optimisation of the C++ code. The original unoptimised system completed a 7x7 game in approximately 62 seconds, whereas our optimised system completed the same game in approximately 5 seconds, representing a twelve-fold increase in efficiency.

3.3 Summary

In this section we have provided a summary of all games relevant to our work here, accompanied by references to their various rulebooks which allow the reader to further educate themselves on those games. We also outline our work on a C++ MCTS engine, and the architecture of the engine.

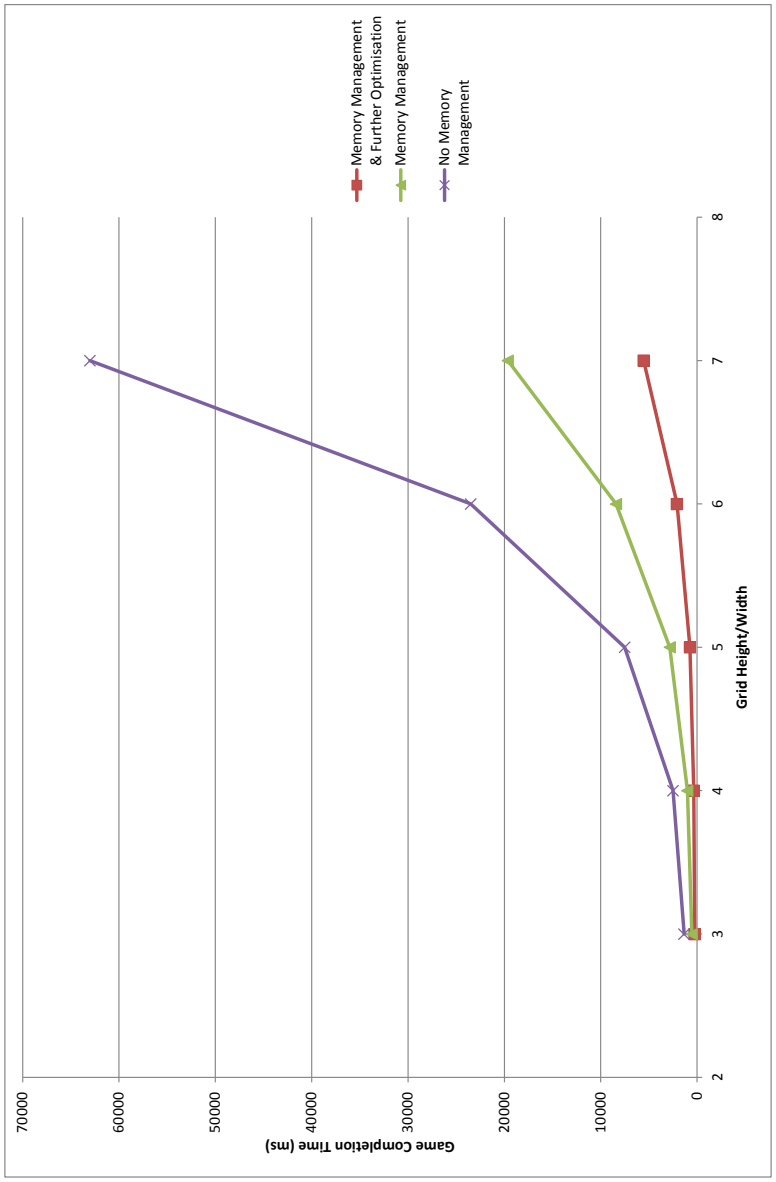


Figure 3.5: Simple engine profiling test results, using the SimpleGrid game and varying memory management techniques. Horizontal axis shows the size of the grid used, vertical axis shows the game completion time.

Chapter 4

Parallelization of Information Set MCTS

Whilst traditional computer software was written for serial computation, the vast majority of modern computers, games consoles and even mobile devices have multi-core processors. This means that parallel computing is an essential concept to getting the most that this hardware can offer. As time is a critical factor, it follows that algorithms using multiple parallel threads of execution are required to use these processors to their full potential. MCTS is readily adapted to parallel execution, with several methods having been proposed [33]. The three proposed methods described below were compared on MCTS (UCT) by Chaslot et al. [37]. Our work here confirms the original result and expands that comparison to ISMCTS.

As the time assigned to an MCTS process increases, more MCTS iterations are performed, and thus more information on the decision space is gathered, and this is likely to have an improvement upon the search result. There are problems with providing a large amount of processor time to an MCTS process, most notably memory limitations which will become apparent with any reasonably complex game. There will also come a point in the search where additional time results in diminishing returns (i.e. the final decision will not be changed by these additional iterations), as no stronger options are identified by the ongoing search.

MCTS has traditionally been applied to games of *perfect information*: that is, games

where the full state is observable to all players at all times and moves are deterministic, non-simultaneous and visible to all players. More recent work has applied MCTS to games of *imperfect information*. Generally this means games with *information asymmetry*, i.e. games where parts of the state are hidden and different parts are hidden from different players. The class of imperfect information games also includes those with chance events, simultaneous moves or partially observable moves. This chapter focusses on *Information Set MCTS (ISMCTS)* [143, 44]. ISMCTS works similarly to regular MCTS, but each simulated playout of the game uses a different *determinization* (a state, sampled at random, which is consistent with the observed game state and hence could conceivably be the actual state of the game). More detail on ISMCTS appears in chapter 2.

Previous work on ISMCTS has focussed solely on the single-threaded version of the algorithm. This chapter applies parallelization techniques for perfect information MCTS to ISMCTS. Some parallelization techniques involve multiple threads searching the same tree, in which case it is necessary to use synchronisation mechanisms such as locks/mutexes to ensure multiple threads do not update the same part of the tree simultaneously. If threads spend most of their time waiting for mutexes to be unlocked, the efficiency of the algorithm is diminished. Games of imperfect information tend to have a larger branching factor than games of perfect information, particularly at opponent nodes. Furthermore, the determinizations in ISMCTS restrict each iteration to a different sub-tree of the overall search tree, reducing the likelihood that two threads will attempt to take the same branch simultaneously. From this we suggest that threads in parallel ISMCTS will spend different amounts of time waiting on mutexes than in the perfect information case, and the relative efficiency of tree parallelization will be different. Our measure of efficiency is two-fold. Firstly, we measure the amount of time taken by the complete MCTS process when assigned a specific number of iterations to make a single decision. Individual agent efficiency is also measured by determining the effectiveness of adding additional agents to the process.

Our focus on parallelization was specified by our sponsor, Stainless Games. However our focus changed as discussed in chapter 1, and this work became less of a priority outside of our first year of work.

The work here focuses specifically on the efficiency of the parallelization techniques (i.e. the number of iterations within a given time budget), which means that optimality of decision is not fully explored here. This work was published in the paper "Parallelization of Information Set Monte Carlo tree search" appearing at IEEE CEC 2014 [123].

4.1 Experimental Methodology

4.1.1 Root Parallelization

Root Parallelization [33] (also known as slow tree parallelization or Single-Run Parallelization) describes the process by which each system runs a separate MCTS from the same game state, then the results are amalgamated by a master process. As each system would have a different random seed, different results should be generated. Effectively we are providing more processor time to the MCTS process by increasing the number of trees which are used for simulation. While some of this time will be consumed examining the same states, the additional randomness hopefully causes a different pattern of exploration, and thus returns useful information once the data is combined at the end of the search.

Root parallelization was implemented as shown in algorithm 2. A separate tree is built by each agent, and then the node statistics of the first level nodes are combined to determine the overall most visited node, and thus the decision to select. First three empty lists are created to hold the trees, agents and threads. Then n agents are created and started, and the algorithm waits until the threads are complete before collecting all the statistics from the

searches and returning the best move from all of them.

Algorithm 2 Algorithm for operation of the Root Parallelization method

```
function DoRootParallelization(nAgents)
    treeList = list < MCTS_Tree > ()
    agentList = list < MCTS_Agent > ()
    threadList = list < Thread > ()

    for nAgents do
        agentList ← newAgent
        treeList ← newTree
        newThread(newAgent.Run, newTree)
        threadList ← newThread

    for threadList do
        thread.Join()

    statistics = list < MCTS_Statistics > ()

    for treeList do
        statistics ← tree.GetStats()
return statistics.GetBestMove()
```

4.1.2 Tree Parallelization

In Tree Parallelization a single shared tree is maintained, and each agent works to add nodes to that tree and update statistics in the tree nodes. $Mutex$, $future$ and $lock\_guard$ are used to ensure that thread safety is maintained (i.e. no two agents attempted to write to the same memory at the same time, or read from memory that was being altered). This technique effectively increases the amount of time used to search by allowing multiple agents to work simultaneously upon the same tree, thus effectively multiplying the available time budget by the number of agents operating, minus some time for negotiating lock statuses.

This process is shown in algorithm 3. First two empty lists are created to hold the agents and threads. We do not need a list for trees, as we will only be maintaining one. Then *n*

agents are created and started, and the algorithm waits until the threads are complete before determining the best move from the single Tree.

Tree Parallelization often uses a technique known as *Virtual Loss* in order to discourage selection of the same node by two different threads. While a node is locked, an additional loss is reported any time it is considered, in order to make that node appear less valuable for selection, and thus decrease the amount of time spent waiting for a node to unlock. We used Virtual Loss in a separate set of experiments. Before we begin a simulation on a node, in addition to locking its local $mutex$, we also add a loss to that node's statistics to reduce the chance of the node being selected by another thread.

Algorithm 3 Algorithm for operation of the Tree Parallelization method

```
function DoTreeParallelization(nAgents)
    agentList = list < MCTS_Agent > ()
    threadList = list < Thread > ()

    for nAgents do
        agentList ← newAgent
        threadList ← newThread(newAgent.Run(tree))

    for threadList do
        thread.Join()
return tree.GetBestMove()
```

4.1.3 Leaf Parallelization

In Leaf Parallelization a single tree is maintained, a single "parent" agent is used to operate on that tree. Whenever a simulation run is required, the parent agent hands that simulation to a "child" agent which then runs independently. Child agents are checked to see if they are clear of an existing simulation before new child agents are created up to the limit by the number of agents. Here we are effectively splitting the MCTS process into two, and allowing parallelization of the Simulation phase only. This technique is likely to be effective

only if the simulation phase of the game is long relative to the length of the decision tree, as otherwise there is little point in attempting to share this phase between different agents, and the limiting factor will be the central MCTS process.

Our Leaf parallelization is implemented as shown in algorithm 4. First we create an empty list for agents, and initialise *n* agents to include in the list. We initialise and run the primary agent, and wait until it completes before determining the best move from the single Tree.

Algorithm 4 Algorithm for operation of the Leaf Parallelization method

```
function DoLeafParallelization(nAgents)
    agentList = list < MCTS_Agent > ()

    for nAgents do
        agentList ← newAgent

    primaryAgent.RunLeaf(agentList)
    statistics = list < MCTS_Statistics > ()

    for treeList do
        statistics ← tree.GetStats()
return statistics.GetBestMove()

function RunLeaf(nAgents)
    agentList = list < MCTS_Agent > ()
    for nAgent do
        agentList ← newAgent
    SimStartNode = RunWithoutSim(tree)
    while currentAgent.IsBusy() do
        currentAgent = GetNextAgent()
    currentAgent.RunSim(SimStartNode)
```

4.1.4 Iteration Budget Experimentation

As we are interested in the speed of decision-making and not the optimality of the decision that results, initial experiments dealt with single game moves instead of complete games.

The state that was used for most experimentation is that of the game after the first two "Issuing the challenge" moves described in the Lords of War rulebook (essentially an initial setup for the game). Two cards are placed during the initial set up, both of which were the Orc General card (see figure 4.1).

Figure 4.1: The Orc General card, an example card from the Lords of War game.

This position was selected for testing as it represents a consistent state which is regularly arrived at, and is similar to other states regularly reached during game play. The initial set up position is displayed in figure 4.2, and is referred to as S_1 for the remainder of this chapter.

During experimentation, the player decks were stacked so they would draw identical cards, and the order was maintained between tests, to ensure that all the examined decisions were identical.

The following series of experiments were then performed, each repeated 1000 times on Plain UCT and ISMCTS:

- Root Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)
- Tree Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)

Figure 4.2: The initial state used in our experimentation (S_1) with two Orc General cards (see figure 4.1).

- Tree Parallelization with Virtual Loss (between 1 and 8 threads, plus an additional set at 16 threads)
- Leaf Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)

During these experiments, the Plain UCT was running on the perfect information game (i.e. all hidden information was made visible), and the ISMCTS agent was playing the imperfect information game (and as such was creating new determinizations as required for its search process.) All experiments were run with 5000 MCTS iterations, as this value was significant enough to ensure time differential between the parallelization techniques, but not so high as to unnecessarily draw out the experimentation. In cases when parallelization was used, the MCTS iterations were split across different agents, with each agent receiving a static 5000/n iterations to perform (where n is the total number of agents.)

It should be noted that Root Parallelization with a single agent is identical in operation to unparallelized MCTS, as only one tree is created and there is no mutex locking during the process. Tree Parallelization with one agent was included to determine the effects of the

mutex locking & unlocking on the decision speed, as this should be the only factor that is different between the two processes.

4.1.5 Win Percentage Experimentation

Further experimentation was conducted to determine the effect of parallelization technique on agent strength. For the purposes of these experiments, complete games were tested, with each decision being restricted by processor time. Each type of parallelization was played 1000 times with decision making time restricted to 500ms. It should be noted that the time restriction was approximate, as due to hardware limits, it is difficult to enforce the time restriction to exactly 500ms. In each case, when the timer had expired, a result was requested from the agent, and the agent returned the result at the next possible opportunity. All results were returned within 50ms of the time restriction.

The following series of experiments were then performed, each repeated 1000 times on Plain UCT and ISMCTS:

- Root Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)
- Tree Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)
- Tree Parallelization with Virtual Loss (between 1 and 8 threads, plus an additional set at 16 threads)
- Leaf Parallelization (between 1 and 8 threads, plus an additional set at 16 threads)

As before, Plain UCT was running on the perfect information games, and the ISMCTS was playing against the imperfect information games and determinizing on each iteration.

4.1.6 Experimentation Environment

The experimental MCTS engine and Lords of War game were implemented in C++ and all experiments were run on a Intel(R) Xeon(R) CPU E5645, with two processes (2.40GHz & 2.39GHz), each with 6 cores & hyperthreading and 32GB of RAM.

A total of four different methods of parallelization were implemented (Root, Tree, Tree with Virtual Loss and Leaf). When appropriate to the style of parallelization, C++ 11 support for $mutex$[1], $future$[2] and $lock\_guard$[3] was used to lock nodes that were being processed. The only nodes that are locked are those selected for the Expansion step of the MCTS process, and those which are being updated after a new simulation has been performed. At all other points, no data should be written to the MCTS tree, and thus no nodes need be locked.

4.2 Results

4.2.1 Iteration Budget Results

The mean average results of the state S_1 experimentation are displayed in figures 4.3 and 4.4. The values for variance of each of these averages are very small ($< e^{-13}$), due to the high number of repeats.

There is a negligible difference between using Tree and Tree with VL in both MCTS and ISMCTS. As there is little overhead to adding or removing the virtual loss, then the main difference in speed would be seen when a virtual loss fails to cause a different node to be selected by the selection policy, as this means that a thread would contest a mutex. The

[1] `http://en.cppreference.com/w/cpp/thread/mutex`
[2] `http://en.cppreference.com/w/cpp/thread/future`
[3] `http://en.cppreference.com/w/cpp/thread/lock_guard`

fact that the results for both are nearly identical suggests that the virtual loss makes little difference to selection, and this is confirmed by additional testing which shows that virtual loss is causing a selection difference less than 1% of the time. So either the selection choice is very clear and a single loss is not affecting the choice, or that the choice is very unclear as the statistics are similar in most nodes at a given level, and nodes are effectively being chosen at random. Chaslot et al. [37] reported that Tree with Virtual Loss performs as well as Root Parallelization on smaller boards in the game of Go, but this does not seem to be the case with Lords of War.

Leaf parallelization is clearly a far slower technique than any other used here. Using more than 3 agents does not result in a speed increase. From the results in figures 4.3 - 4.6, we can see that the addition of agents numbered above 3 has almost no effect on the results: the simulations assigned to earlier agents are already complete by the time a simulation would be assigned to an agent numbered 3 or higher.

Tree Parallelization shows itself to be a competitive technique in terms of speed (figures 4.3 and 4.4), but still a lot slower than Root Parallelization in both MCTS and ISMCTS. If we calculate the difference in speed between Tree and Root in MCTS, then the difference in speed between Tree and Root in ISMCTS, it can be seen that the difference is comparatively lessened in ISMCTS, but that the speed decrease caused by ISMCTS is still more significant.

As discussed earlier, we can see the effects of using $mutex$es to lock nodes by comparing the difference in performance between root and tree parallelization when using 1 agent, however this only accounts for the actual cost of the locking procedure, not the expense caused by causing any threads to wait. The average of this difference is very small, the best estimate being less than 16ms (due to the resolution of the timer used). This indicates that the time spent locking $mutex$es is very low, and almost all of the expense comes from threads waiting to obtain lock on a $mutex$. We can see a similar difference between the

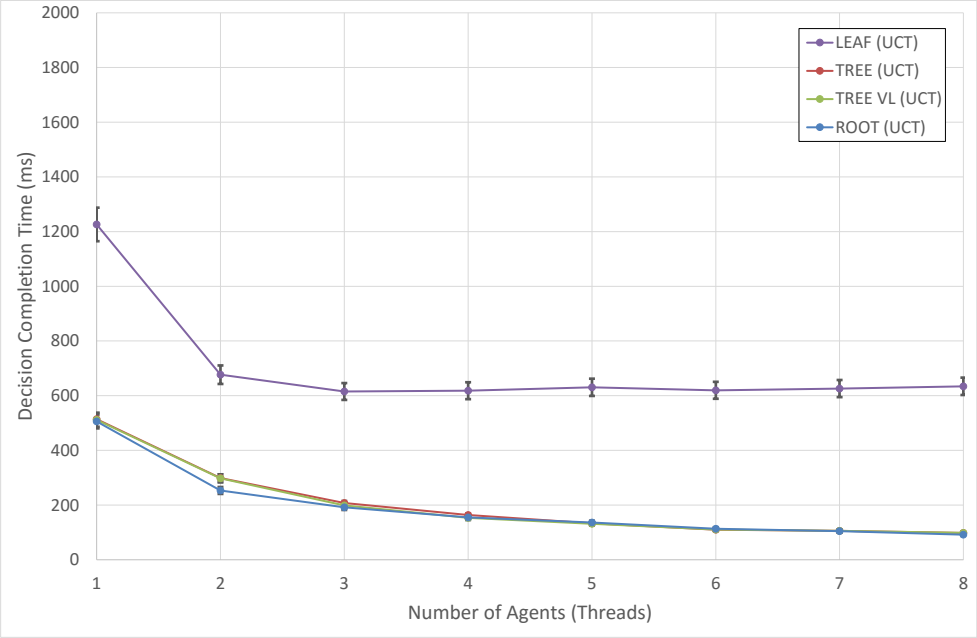


Figure 4.3: Results of applying four different parallelization techniques to UCT operating upon state S_1. Results are expressed in milliseconds used to complete a decision.

ISMCTS runs of root and tree parallelization.

In order to see the relative effects of different parallelization techniques on UCT and ISMCTS, we can compare the relative efficiency of individual agents within each technique (see figures 4.5 and 4.6.) Efficiency is calculated as $\frac{t_1}{n \cdot t_n}$, where t_n is the decision time for n agents. In particular, the efficiency for $n = 1$ is $\frac{t_1}{1 \cdot t_1} = 1.00$. In an ideal scheme with 100% efficiency, using n agents would result in an n-fold increase in speed: adding the second agent would cause overall speed to double resulting in a decision time of $t_2 = \frac{t_1}{2}$, and so on. The results show that root parallelization spreads the load between agents most effectively, with one exception of note - the 2nd agent in leaf parallelization on MCTS.

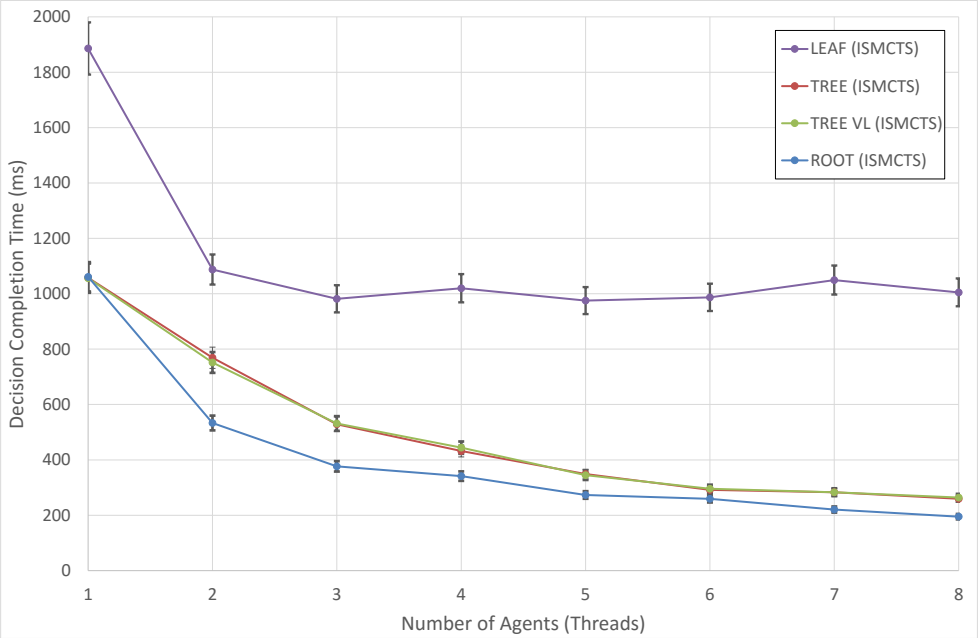


Figure 4.4: Results of applying four different parallelization techniques to ISMCTS operating upon state S_1. Results are expressed in milliseconds used to complete a decision.

4.2.2 Win Percentage Results

The results of the state S_1 win percentage experimentation are displayed in figures 4.7 and 4.8. The general curve of the results shows that each iteration added is less effective than the previous, and thus has a smaller effect on play strength for our fixed 5000 iterations.

As in the previous experiments, we can see that there is little difference in behaviour between Tree with and without Virtual Loss. As mentioned previously, we can attribute this to differences in the game of study, and also that a single temporary loss is unlikely to affect overall decision making, as the difference is intended to have a minimal effect on algorithm operation.

We can see that tree parallelization performs substantially better in these experiments, which can be attributed to the fact that the focus of all agents is on a single search space,

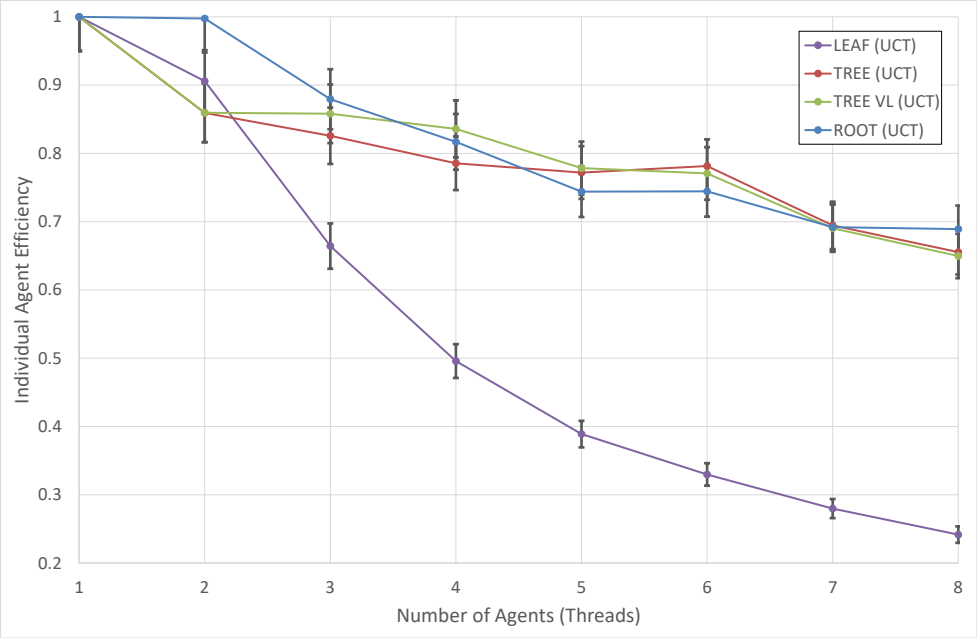


Figure 4.5: Results of applying four different parallelization techniques to UCT operating upon state S_1. Results are expressed in individual agent efficiency, where 1.0 represents optimal efficiency.

unlike in Root parallelization where processor time may be wasted re-exploring decision space already explored by other agents.

We can see that overall ISMCTS behaves more poorly than UCT in terms of efficiency, however this is to be expected, as the UCT agents have access to perfect information of the game state, and the ISMCTS agents include enhancements to cater for the lack of such information, so this direct comparison does not speak of their relative effective play strengths.

Leaf parallelization performs approximately equally to standard UCT, flattening out almost immediately after adding additional agents. This indicates that the simulation process is not significantly expensive for this game, and thus parallelizing this process alone does not have a significant effect on agent strength.

We also ran some brief experiments to track win percentage on the different paralleliza-

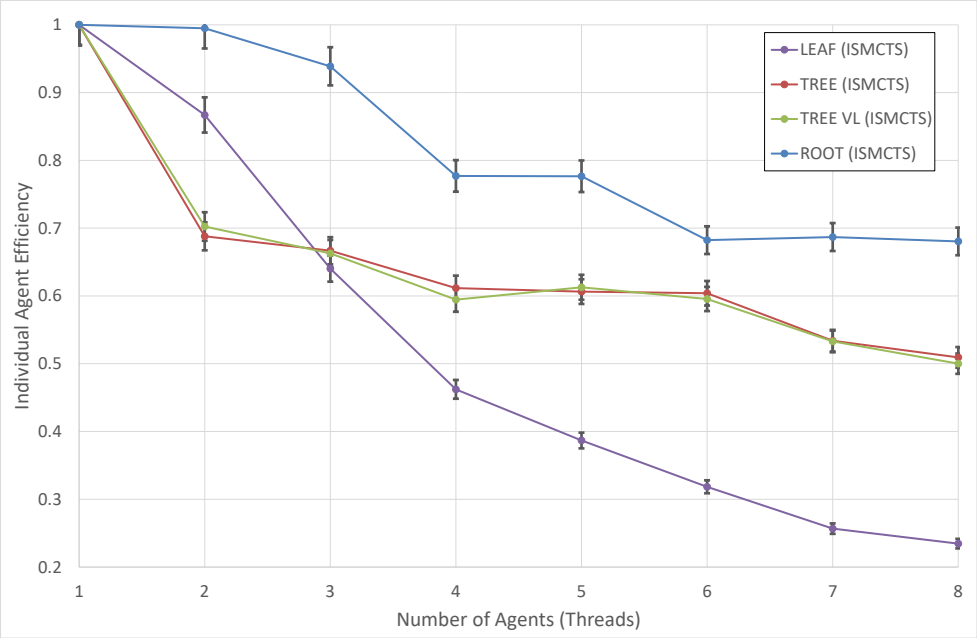


Figure 4.6: Results of applying four different parallelization techniques to ISMCTS operating upon state S_1. Results are expressed in individual agent efficiency.

tion types (see figures 4.7 and 4.8).

4.3 Summary & Discussion

It can be seen that in no combination of tested factors did ISMCTS outperform UCT in terms of time to execute a fixed number of iterations. The results of this comparison are not particularly unexpected however, as ISMCTS is a significantly more complex process which caters to the use of imperfect information, and these overheads to its operation will significantly reduce it's operational speed when compared to that of Plain UCT. These enhancements take the form of determinizations of the game state which consume operational budget, but add significantly to the play strength in games of incomplete information [44].

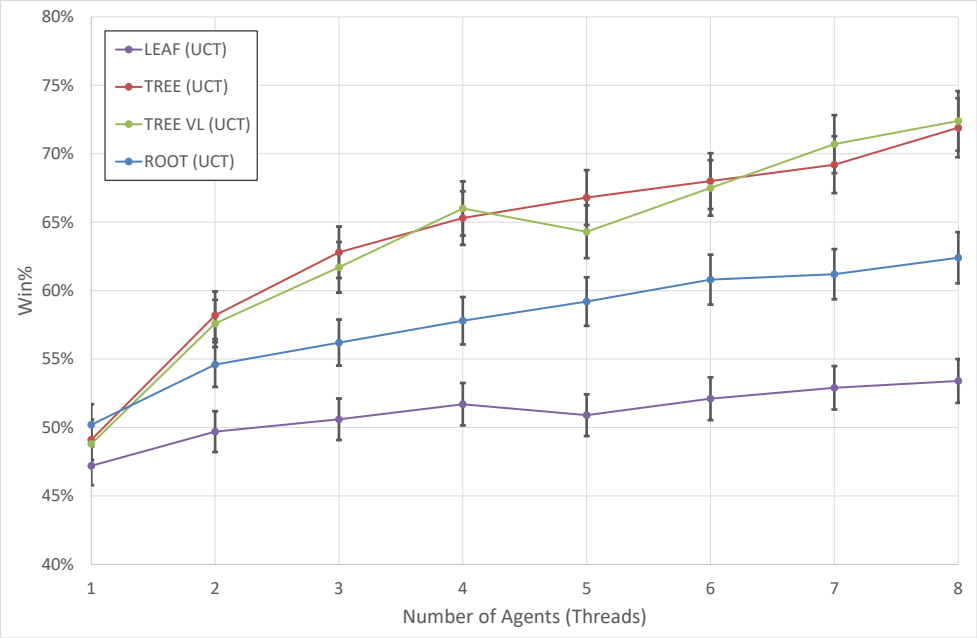


Figure 4.7: Win Percentage when applying four different parallelization techniques to a UCT agent. The opponent in each case was an unparallelized UCT agent with the same iteration budget.

There does not appear to be a significant difference in the slow-down caused by tree parallelization between ISMCTS and UCT, which is contrary to what was originally expected. This indicates that the slow-down is unlikely to be caused by threads awaiting locked resources, but more likely by code associated with the locking and unlocking process. It should also be noted that parallelization of a single process across n agents for a single unit of time will always be less efficient than the same single process on a single agent running for n units of time. However the idea of parallelization is use advanced processing power to achieve more in a reduced amount of time.

One possible explanation is that due to the shape of the trees, the locking of initial nodes is having a larger effect on an ISMCTS tree than a UCT tree. As mentioned previously, the expansion step of the MCTS process locks the node to be expanded. The UCT tree will only

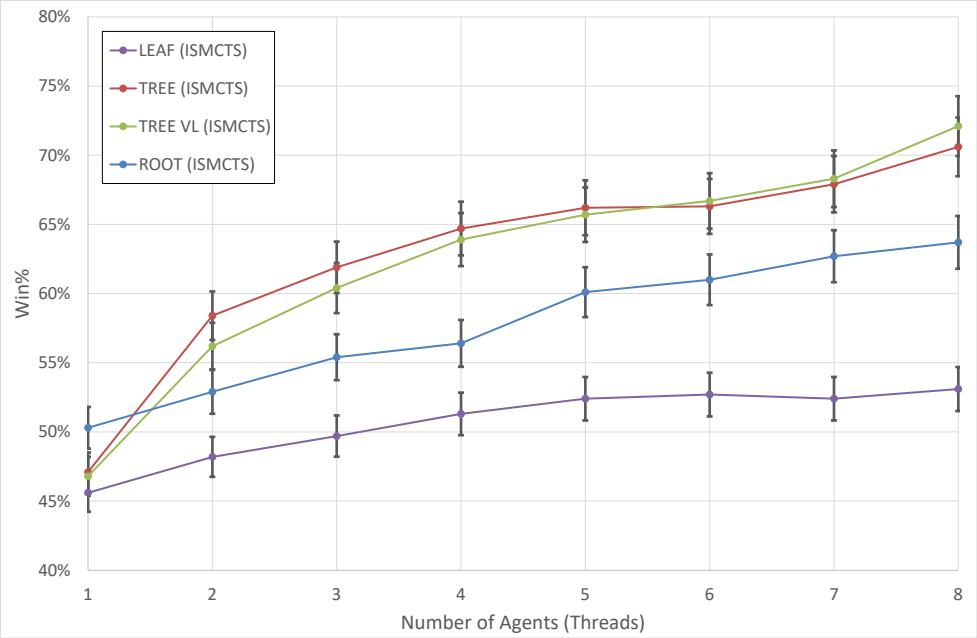


Figure 4.8: Win Percentage when applying four different parallelization techniques to a ISMCTS agent. The opponent in each case was an unparallelized ISMCTS agent with the same iteration budget.

have a relatively small number of nodes at the first level (approximately 20-30), meaning that once these nodes are expanded, other agents are cleared to continue through the root node without locking. The ISMCTS tree will have a large number of nodes at the first level (approximately 150), which will cause an initial delay as multiple agents compete to lock the root node. This effect may continue on other promising nodes during the early stages of the tree building process.

Of all approaches attempted, root parallelization across high numbers of agents was the most time efficient approach (decision time 82ms using 16 UCT agents, 187ms using 16 ISMCTS agents), although the speed increase became mostly negligible when adding more than 4-5 agents.

It's also worth noting that tree parallelization is less effective when applied to ISMCTS

(decision time 248ms using 16 agents) than when applied to UCT (decision time 84ms using 16 agents), which is somewhat surprising. Using tree parallelization in an environment where agents are unlikely to be blocked should increase efficiency, but the relative decrease in efficiency suggests that more blocking is occurring.

Adding virtual loss has almost no effect on the effectiveness of tree parallelization in Lords of War. This may be due to the positioning of valuable states in the game tree - if there are few positions of high value in a tree, then adding a virtual loss is unlikely to dissuade from their further immediate exploration.

It is important to note that our work on parallelization only used a single game for study, and thus the results may have been influenced by factors that are unique to the decision space of this particularly game. Further work should investigate other games, if only to verify that no game specific factors interfere with the results.

4.3.1 Contributions

Our contributions from this chapter are:

- A survey of the effects of parallelization upon ISMCTS.
- An insight into the effects of MCTS (Plain UCT) and ISMCTS parallelization upon the complex card game *Lords of War*.
- A comparison of the effects of parallelization upon MCTS (Plain UCT) and ISMCTS.

Our contributions show that effective use of parallel processing will directly result in a more effective use of MCTS and ISMCTS artificial agents, and provides insight into the effects of adding processes on speed up and playing strength.

While the survey here is performed upon one target game, parallelization of MCTS is useful in virtually any MCTS usage where enhanced performance would be an advan-

tage. Given that MCTS is search technique, and a higher budget generally increases the effectiveness of the search, a majority of MCTS implementations would stand to gain from parallelization techniques.

We can expect parallelization techniques to be used frequently in industry implementations of MCTS. Whether it be for oppositional agents in games, where parallelization could occur on available memory (or GPU) in the user's system, or other industry search and logistical applications, where entire server farms could be used upon a single search. It would be important to be wary of the negative effects of parallelization, particularly in this last example, where the application of more processing power might not have the desired results. A good example of this is AlphaGo [127], created by Google DeepMind, which makes extensive use of parallelization to take advantage of advanced hardware.

Chapter 5

Modifying MCTS to Create Different Play Styles

While MCTS is simple in operation, it results in complex and asymmetrical decision trees, the form of which cannot easily be predicted before creation. Relatively minor modifications of the MCTS process can lead to radical changes in the direction of tree growth, and also the final decision. Two common methods of altering the MCTS process are *Move Pruning* and modification of the *Action Selection Mechanism*. The work in this chapter is performed on the game Lords of War, however the techniques are easily applicable to a wide range of complex games, particularly (but not limited to) complex card games (e.g. Magic: The Gathering, Netunner, Hearthstone) and positional board games.

The motivation for this work is the creation of artificial players with interesting, complex behaviours, which include human heuristic knowledge and thus might include the collateral bonus that they make more human-like moves.

This work was published in the papers "Heuristic Move Pruning in Monte Carlo Tree Search for the Strategic Card Game Lords of War" appearing at IEEE CIG 2014 [122], and "An Experimental Study of Action Selection Mechanisms to Create an Entertaining Opponent" appearing at IEEE CIG 2015 [124].

5.1 Move Pruning

Move Pruning is a technique which removes sections of a decision space in order to narrow the space to be searched, and thus reduce the amount of time a search approach (such as MCTS) needs to reach a stable decision. If poor sections of the tree are removed from consideration, the following tree search should converge more quickly upon a strong solution. Therefore the ideal Move Pruning technique is one that removes only unpromising sections of the tree, removing then from consideration before the search begins. Some methods of pruning also look to remove *trap states* [108], which are states that appear strong to a search agent, but are actually guaranteed losses, or simply unpromising. In this work, we describe pruning techniques as either *Hard* or *Soft*. *Hard Pruning* techniques are those which permanently remove sections of the tree from consideration, effectively blocking them from ever being searched. *Soft Pruning* either temporarily removes sections of the tree or deprioritises them, making them less likely to be searched. In this chapter we use heuristic hard pruning to reduce the branching factor of the search tree and demonstrate that this produces stronger play in Lords of War. We then combine heuristics to produce multi-heuristic agents which are played-off against the single heuristic agents to determine their relative playing strengths. Our best combined heuristic agent proves to be a competitive opponent that exhibits a rather different play style.

We also investigate pruning using *State-Extrapolation*. For the initial heuristic pruning tests, we prune moves based on a heuristic evaluation of the game state after the move is made. When pruning a move using state-extrapolation, we move the game forward until just before the next opponent decision, then determine the suitability of the move by the heuristic evaluation of that state. This applies both in our chosen domain, Lords of War, and in many other domains where a player move consists of a series of linked decisions before the opponent has the opportunity to react (e.g. Magic the Gathering, Netrunner.) We show

that this technique improves the strength of the search by allowing the heuristics to evaluate a more representative game state. We then compare State-Extrapolation across a selection of the strongest heuristics we have used, and examine its comparative effect on play strength for those agents.

5.2 Action Selection Mechanisms

The term *Action Selection Mechanism* describes the process by which a move is chosen from the root level of the MCTS tree once the search is completed. There has been limited work on action selection mechanisms, likely due to the apparent relative effectiveness of the mechanisms presented by Chaslot et al [38]. In this chapter we experiment with modification of the action selection mechanism, while leaving the centre ISMCTS process intact, and avoiding modification to the number of iterations used, in order to create AI agents based on providing a tailored level of challenge and complexity (and hence, we would argue, entertainment.)

5.3 Experimental Methodology

5.3.1 State Evaluation

We applied our own experience of Lords of War in the creation of several functions which may be useful for examining a state. These functions (f_j) are applied to a specific game state S_i such that $f_i : S \rightarrow \mathbb{R}$, and are intended to form building blocks for construction of heuristics which will be used to measure the fitness of a state (correlated to the probability that the assessing player will win from that state). Each function is performed including only cards of the active player unless the otherwise specified by the modifier *opp*, in which

case the opponent's cards are considered instead (e.g. f_j^{opp}). The set B_i is the set of all the active player's cards on the board in state S_i (or the opponent's cards if opp is used). The set H_i, E_i and D_i are similarly the sets of cards in the players hand, eliminated pile and deck respectively. The following functions were used to simplify the expressions for the state evaluation heuristics:

$$f_1(S_i) = |B_i|$$

$$f_2(S_i) = |E_i|$$

$$f_3(S_i) = \sum_{b \in B_i} (b.DefenceValue)$$

$$f_4(S_i) = |\{b \in B_i | b.Threat() > 0\}|$$

$$f_5(S_i) = |\{g \in E_i | g.IsCommand()\}|$$

To briefly explain these functions, $f_1(S_i)$ counts all the active player's cards, $f_2(S_i)$ counts all the active player's dead cards, $f_3(S_i)$ sums all defence values for all the active player's cards, $f_4(S_i)$ counts all squares threatened by the active player's cards, and $f_5(S_i)$ counts all the active player's dead commander cards. Functions $f_6(S_i) - f_{11}(S_i)$ use the heatmaps in figure 5.1 to assign values to the active player's cards based on their position, and then sums those scores. The functions were then used to create the *State Evaluation Functions* listed below. It is worthy of note that during early testing, our heuristics which considered card attack strength were universally weak. This likely speaks of the importance of defensive play in Lords of War. As such, no heuristics incorporating attack strength were taken forward to further experimentation.

5.3.1.1 Card Count (h_1)

This heuristic was selected for testing partially because it was the simplest of the heuristics, but also because it performed very well in initial testing. h_1 assigns a weight of +1 for each card on the board and a weight of -1 for each card in the graveyard, negating these weights for opponent cards.

$$h_1(S_i) = (f_1(S_i) - f_2(S_i)) - (f_1^{opp}(S_i) - f_2^{opp}(S_i))$$

5.3.1.2 Average Defence (h_2)

This heuristic was selected for testing because it would appear that strong players often play defensively in Lords of War, and this heuristic would hopefully mimic that style of play. This heuristic measures the difference between player and opponent of the mean defence value of cards on the board. In the case when a player has no cards on the board, we assume a value of 0 for that player's contribution to the value of h_2.

$$h_2(S_i) = (f_3(S_i)/|B_i|) - (f_3^{opp}(S_i)/|B_i^{opp}|)$$

5.3.1.3 Threatened Area (h_3)

This heuristic counts the number of empty or opponent occupied squares on the board that are directly threatened (under attack by adjacent card's non-ranged attacks) by active player cards. The same calculation is made for the opponent and subtracted from the total. This heuristic was selected so as to consider the positional elements of the game.

$$h_3(S_i) = f_4(S_i) - f_4^{opp}(S_i)$$

5.3.1.4 Simple Card Count with Dead Commander adjustment (h_4)

This heuristic is similar to h_1, except command cards in the dead pile count for two cards instead of one. The adjustment to h_1 is due to our own play experience and understanding

of the importance of command cards to the game (as it is possible that an AI may be too willing to lose its first 2-3 command cards in combat).

$$h_4(S_i) \ = \ (f_1(S_i) - f_2(S_i) - f_5(S_i)) - (f_1^{opp}(S_i) \ - \ f_2^{opp}(S_i) - f_5^{opp}(S_i))$$

5.3.1.5 Active Player Average Defence (h_5)

This heuristic was a modification upon h_2 to remove the subtraction of an opponent score from the total. This was tested as a heuristic mainly because h_2 seemed like such a strong candidate, yet performed poorly in tests. In the situation where the value of $|B_i|$ is 0, then the value of $h_5(S_i)$ is also evaluated as 0.

$$h_5(S_i) = (f_3(S_i)/|B_i|)$$

5.3.1.6 Basic Heat Maps ($h_6 - h_{11}$)

This set of heuristics is similar to h_1, except each card is assigned a weight value which depends on its placement location, then these values are summed to create the state score. When the modifier "*opp*" is used, the heat maps are reflected about the horizontal axis to account for the opponent playing from the opposite side of the table (this is only of significance to m_a and m_b). The maps for these heuristics are shown in figure 5.1. We would expect these heuristics to be poor when used in isolation, but perhaps stronger when com-

bined with another heuristic which measures strategic strength, such as h_1 or h_5.

$$f_6(S_i) = \sum_{b \in B_i} m_a(b)$$

$$f_7(S_i) = \sum_{b \in B_i} m_b(b)$$

$$f_8(S_i) = \sum_{b \in B_i} m_c(b)$$

$$f_9(S_i) = \sum_{b \in B_i} m_d(b)$$

$$f_{10}(S_i) = \sum_{b \in B_i} m_e(b)$$

$$f_{11}(S_i) = \sum_{b \in B_i} m_f(b)$$

5.3.2 Single Heuristic Experimentation

During pruning, we apply the appropriate heuristic (h_i) to the state that results from the move under examination. We then prune all except the top scoring moves. The number of moves that each heuristic selects to exclude from pruning is referred to as the *Hard Pruning Limit (HPL)*.

During our experiments, values ranging from 1 to 30 were used for HPL. Each of the heuristics were run against Plain UCT using 10000 iterations for the perfect information variant of the Lords of War game. The following experiments were each repeated 500 times in each case, where UCT is plain UCT, and UCT(h_i[n]) is UCT using h_i for hard pruning with a HPL of n, $(i = 1, 2, ...11)$ and the values of HPL tried were 1,2,5,10,15,20,25 and 30.

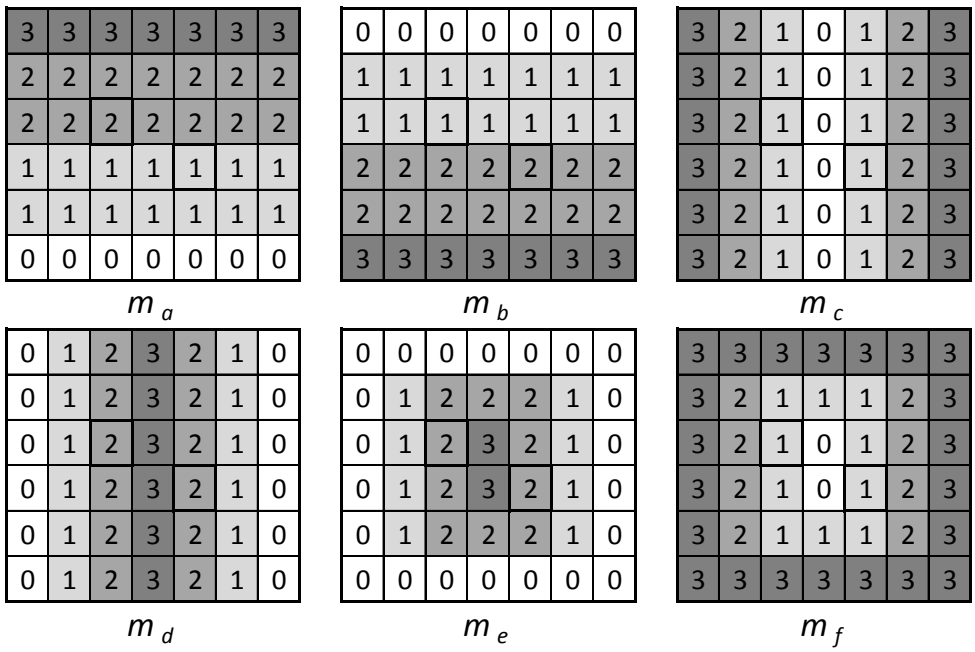


Figure 5.1: Sample board heatmaps for use by heuristic pruning algorithms in the game Lords of War

5.3.3 Multi-Heuristic Experimentation

In later tests, multiple heuristics were used in combination. An AI using multiple heuristics would select moves up to the HPL from each heuristic in turn, then combine the obtained results, removing any duplicate entries. This results in a list of top moves containing between HPL and $(n \times HPL)$, where n is the number of heuristics being used by the agent. The average number of duplications that appeared in multi-heuristic experiments is discussed in the results section.

Our strongest single heuristic (h_4), was combined with each of the other heuristics in an attempt to create a strong multi-heuristic agent. These new agents were then played against the original h_4 to determine their strength against our strongest single heuristic agent. Results of this experiment are given in section 5.4.2.

5.3.4 State-Extrapolation

In all previous experiments, we have pruned moves based on the score obtained from the state following that move. In these State-Extrapolation experiments, we roll forward the game state to some forward point in the game and prune based on the state at that point. This has the effect of running through the combat step (when ranged attacks are assigned, and dead cards are removed from the board), and thus should provide a better estimation of the strength of the move.

There are multiple ways in which we can roll forward. The simplest way is to randomly select moves until some point in the future, most logically the opponent's next move. We can also look to perform a search over the sub-tree from the remainder of the turn. In this study, we experiment with randomly rolling the state forward until the opponent's next move.

We expect State-Extrapolation to be a strong technique, as it should give a more accurate representation of the actual game state. For example, rolling forward past the end of combat step would allow us to observe clearly which cards will be removed, and thus the resulting layout of the game board.

5.3.5 Action Selection Mechanism

Strength of play in Lords of War is closely tied to positional elements of card placement. As such, the deployment move is the most strategically important move. The selection of whether to remove a card from the battlefield or draw a new card is a comparatively simple binary choice, as retreating a weak card is more often the correct choice, particularly if there is an exposed high value card. The simplest choice is that of selecting a target for a ranged attack. The target is normally obvious as the situation in which more than one destroyable target is available is uncommon, and when there is an available target which can be destroyed with a ranged attack, performing that ranged attack is almost always the stronger decision.

If there is no such target that will be destroyed by a ranged attack, then not making a ranged attack is the correct choice (as this allows the unit to melee attack instead, which may have an effect.)

The selection mechanisms which we investigated are listed below. As mentioned in Chapter 2, RobustChild is the standard applied in MCTS, selecting the move from the root that has the highest visits (v) amongst its neighbours. MaxChild is also well established, and selects the move that has the highest wins/visits (w/v). RobustRand$_n$ and MaxRand$_n$ perform uniform selection across the top n moves by visit or value respectively. The value n was provided as an input parameter (note that RobustRand$_1$ is identical to RobustChild, and MaxRand$_1$ to MaxChild.) RobustRoulette^k and MaxRoulette^k perform fitness proportionate selection (or roulette wheel selection) by visits or value respectively. An input parameter was supplied to RobustRoulette^k and MaxRoulette^k, each fitness value was raised to the power k before making the selection (to increase the likelihood of selecting a fitter move). RobustRoulette^k_n operate similarly to RobustRoulette^k, except the roulette selection was limited to the top n moves. In each case, the agent using the new selection mechanism was assigned a budget of 10000 iterations. All opposing agents were using ISMCTS with RobustChild selection mechanism and a budget of 10000 iterations, except where otherwise noted.

$$RobustChild : argmax(v_i)$$

$$MaxChild : argmax(w_i/v_i)$$

$$RobustRand_n : P_i = \tfrac{1}{n} \text{ for } i \in \{a_1, a_2, ..., a_n\}$$

$$MaxRand_n : P_i = \tfrac{1}{n} \text{ for } i \in \{a_1, a_2, ..., a_n\}$$

$$RobustRoulette^k : P_i = (v_i)^k / \sum (v_j)^k$$

$$MaxRoulette^k : P_i = (\tfrac{w_j}{v_j})^k / \sum ((\tfrac{w_j}{v_j})^k)$$

$$RobustRoulette_n^k : P_i = (v_i^k)/(\sum_{j=1}^{n}(v_j^k))$$

5.3.6 Complexity Measurements

To evaluate the selection mechanisms, we used five measures of complexity during experimentation. These measures were proposed due to accumulated heuristic knowledge of the Lords of War game, both our own and those of expert players and the game designers. The set A is all available actions to the player in state S, and the set B is the set of all the active player's cards on the board in state S (and B^{opp} the opponent's board cards).

$$Cx_0 = |A|$$

$$Cx_1 = |\{b \in B_i | b.Threat() > 0\}|$$

$$Cx_2 = \sum_{b \in B_i}(b.Threat())$$

$$Cx_3 = |B^{opp}|$$

$$Cx_4 = |B| + |B^{opp}|$$

Cx_0 is a simple count of the moves available to the agent at each decision state, Cx_1 is a count of all enemy or empty squares threatened by friendly cards, Cx_2 is the sum of all threat values directed at the squares in Cx_1, Cx_3 is the total number of opponent cards on the board, and finally Cx_4 is the total number of all cards on the board. All values for complexity measure were normalized to lie in $[0, 1]$ according to the observed limits for those measures.

These measures were selected primarily because they represent a human expert's interpretation of what constitutes a complex and interesting board state. The most basic measure (Cx_0), states that a board with more available moves is more complex to the player selecting

which move to take, which in most situations will be true. The measure Cx_1 is a measure of how much threat the active player is placing upon the board, which increases complexity because there are more interactions to be considered between cards on the board. The measure Cx_2 states that the more threat an opponent directs towards your pieces, the more complex the board state, as you must more carefully consider where to make your next move without endangering both the card you place and your cards already upon the board. The measure Cx_3 states that complexity increases as the opponent places more cards on the board, however this is a mixed measure, as this will both increase threat on surrounding squares due to melee and ranged attacks (included in Cx_2), and also directly block a single square, potentially reducing the available moves of the opposing player if that square was previously a valid move. Finally, the measure Cx_4 just counts all cards on the board, as a board with more cards has more interactions to consider, and also more squares affected by threat. These complexity measures are not intended to completely encapsulate everything that is complex about a game state, as that itself would be a challenging and project with somewhat subjective targets, but rather to provide an insight into what players may consider complex, and a guide for our heuristic pruning and action selection.

Our own experience of playing against agents of differing configuration has indicated that the complexity measures do indeed reflect the likelyhood of an agent behaving unusually and creating unexpected game states. Generally speaking, agents with low complexity measures made moves which directly achieved clear objectives, such as eliminating or threatening opponent pieces, or securing existing positions. Agents with higher complexity measures tended towards making somewhat unexpected moves, which tended towards being weaker, but were occasionally very strong and difficult to counter. This seemed particularly true with regard to Cx_1, which we felt through personal observation was the strongest indication of complexity in play.

In addition, a short survey has conducted with human players using a deployed Windows

Executable version of the Lords of War game. The users were asked to play a series of games of Lords of War against an unknown AI opponent, and then answer questions relating to how strong and complex the opponent's play was. Unfortunately we received a very small number of responses to this survey, and as such no meaningful conclusions could be drawn, however those we did receive seemed to support our own personal experiences noted above.

5.3.7 Online tuning

We also investigated online tuning of parameter values in order to match the strength of the agent to an opponent. The tuning method was to measure the MaxChild (w_i/v_i) value of the selected move at each decision and then increase the configuration parameter if $w_i/v_i > 0.5$, or decrease it otherwise (in the case of $RobustRoulette_n^k$, where there are two configuration parameters, the value of n was modified by online tuning.) This should result in an agent that becomes weaker when it selects moves that lead to a concentration of wins, and stronger when it selected moves that lead to a concentration of losses. The value of n was limited in the range [1-10], mostly to ensure that the value remained within a reasonable range for computation. Results for online tuning are shown in figures 5.14 & 5.15.

5.4 Results

All experimentation was performed on the hardware and software specified in section 3.2.

5.4.1 Single Heuristic Results

The results for the initial heuristic tests are shown in figure 5.2 & 5.3.

A Hard Pruning Limit of below 5 seems poor for all heuristics tested, with all such agents consistently losing to Plain UCT, and this was confirmed by testing HPL=4 and HPL=2. If a

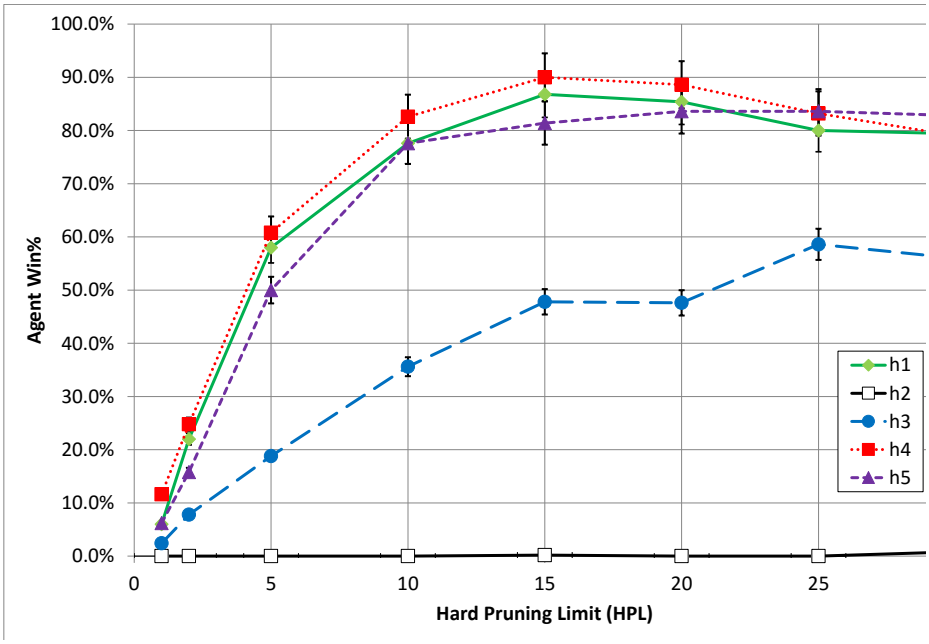


Figure 5.2: Mean win% of single heuristic agents h_1 - h_5 playing against a standard UCT agent with 10000 iterations at varying Hard Pruning Limits (HPL)

heuristic was a poor indicator of move strength, we would expect to see a slow and gradual increase in strength as the HPL rises, which should come to a halt at approximately 50% win rate. This is due to a high HPL being equivalent to using no heuristic at all, as no moves will be pruned by either method. We can see this behaviour in the agent using h_3, and all the heat map heuristics (see figure 5.3). This is consistent with our belief that the heat maps in isolation would be poor pruning heuristics.

The strongest single heuristic results appear between a HPL of somewhere between 15 and 25, with the strongest individual result being h_4 at a HPL of 15. h_1 and h_4 are very similar heuristics. However as h_4 performs better, we can see that including an adjustment for command cards within the heuristic has increased its effectiveness. This could be considered for future heuristics, and may increase their play strength.

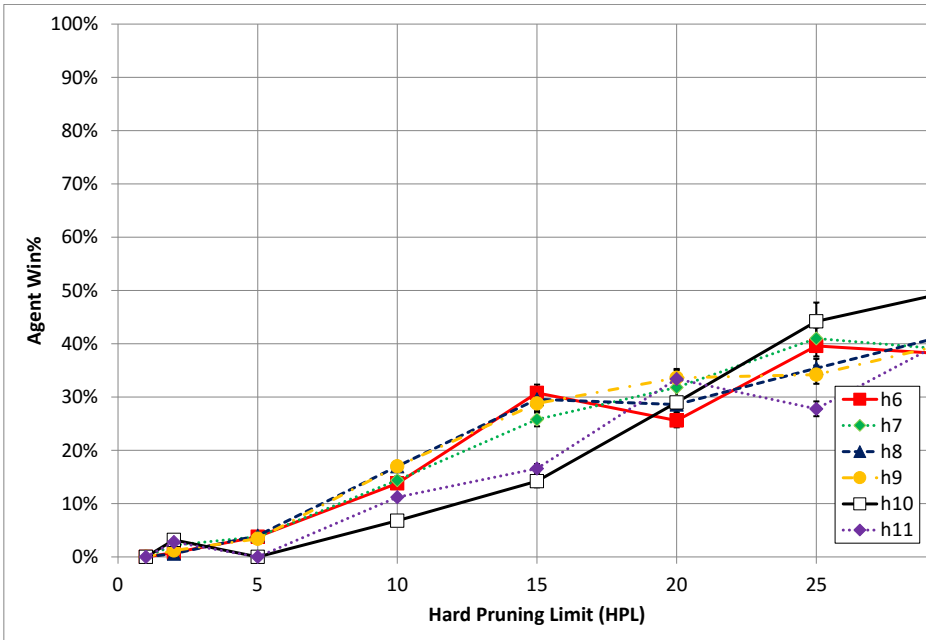


Figure 5.3: Mean win% of single heuristic agents h_6 - h_11 playing against a standard UCT agent with 10000 iterations at varying Hard Pruning Limits (HPL)

The most effective heuristics seem to be h_1 and h_4, with h_5 being a strong third. It is surprising that h_2 performed so poorly given that h_5 performed so well. This is possibly due to h_2 being highly susceptible to strong play from an opponent, however we can establish no reason why it would be more susceptible than h_1 or h_4. It is more likely that the state with no cards on the board is stronger than the heuristic indicates (for example, having no cards on the board while your opponent only has one is not as poor a position as h_2 would indicate, since it means that you have a target for attack where your opponent has none.)

When applied alone, the heatmap heuristics (h_6 - h_{11}) behave very poorly as we suspected (see figure 5.3), likely due to their complete lack of consideration for any strategic element other than position. The underlying MCTS agent will still ensure some level of skilled play, but it is clearly outmatched by the non-pruned opponent. This is of little con-

cern however, as these agents were not designed to function alone, but rather as part of a multiheuristic.

Due to the poor results from heuristic h_2, we also conducted an experiment with a negated value of h_2, but it was completely unsuccessful, winning 0 games in all tests. The clear trends shown in the results for the improvement of the results with an increase in HPL prompted further experimentation with higher pruning limits, so further experiments were run increasing the pruning limit until the Win% exhibited a decrease. Results of this experiment are given in section 5.4.1.

5.4.2 Multi-heuristic Results

The results of the experimentation with multi-heuristic agents are displayed in figure 5.4.

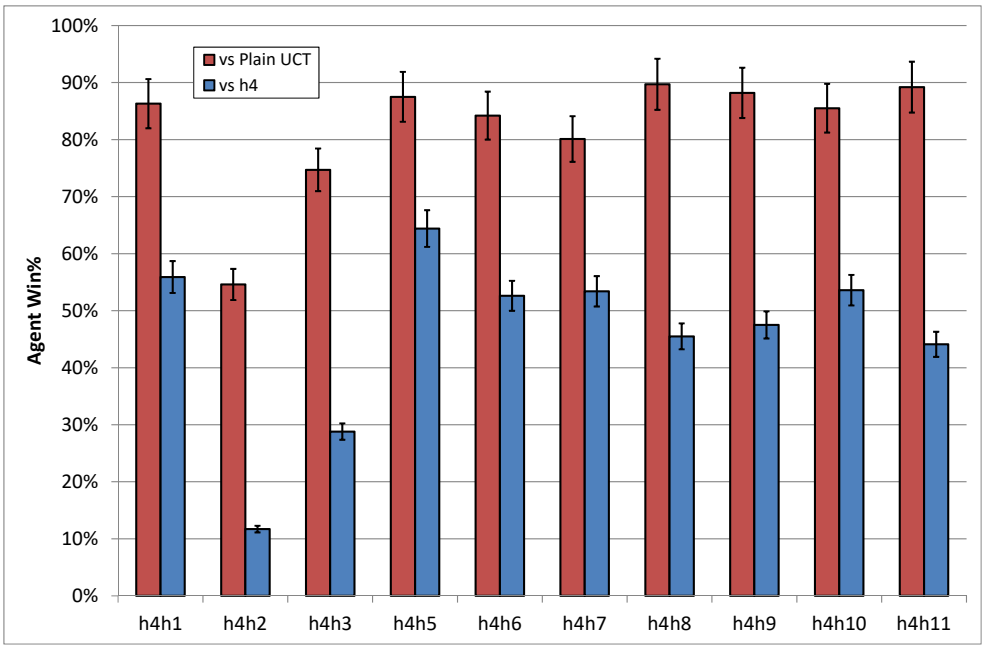


Figure 5.4: Win% of multi-heuristic agents h_4h_1 - h_4h_{11} playing against a standard UCT agent at Hard Pruning Limit 15

The combination of h_4 with any of the heat map heuristics causes a strong improvement in performance, and while only one clearly exceed the original performance of h_4 against plain UCT, they perform at about the same level. This is likely due to the moves being selected by h_4 being responsible for most of the strong decisions. When each of these agents are played against h_4, the agent h_4h_5 performs the best, suggesting that h_5 is contributing towards the success of h_4. Of the multi-heuristic agents using heat maps, the agents using h_4h_6, h_4h_7 and h_4h_{10} show the best performance. This confirms our experience that playing near the front or back of the board is strong, but suggests that playing in the centre of the board is stronger than playing at the edges. This may indicate that controlling the centre of the board is more important than the benefit of playing your cards against the edge in order to protect their weaker sides. This difference in performance may also be due to human players trying to place blank card sides (sides with no attack value) against the board edges, whereas no such consideration is included in the agents.

5.4.3 State Extrapolation

We can see from comparison of the original agents versus those using state-extrapolation that there is little difference in win% in most cases (see figure 5.5). However the difference in two specific cases is significant, those of agents using h_{10} and h_{11}, and for all but one heuristic state extrapolation gives slightly stronger results.

Heuristics h_{10} and h_{11} use heat maps which are exact opposites of each other (see m_e and m_f in figure 5.1). Our experience of Lords of War is that the strength of a move can be closely associated to proximity to a board edge. The difference in effect upon these two heat maps and the other heat maps ($h_6 - h_9$) can likely be attributed to this difference.

Figure 5.6 shows us that using state extrapolation has strengthened the h_4R agent (where R denotes the use of state extrapolation), displaying a win rate of approximately 80% against

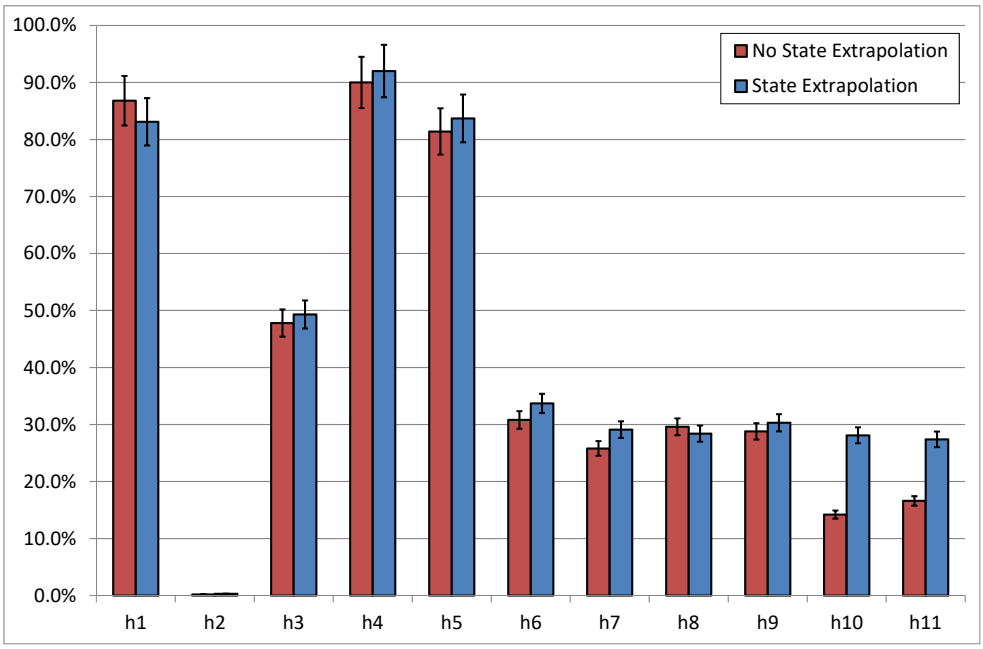


Figure 5.5: Win% of single heuristic agents h_1 - h_{11} playing against a standard UCT agent applying State-extrapolation to heuristic agents

our previous strongest single heuristic agent (h_4). It is also worth noting that h_1R wins approximately 50% of games against h_4, meaning the application of state extrapolation to h_1 has improved performance to that of our previous strongest single heuristic agent.

In the following results, BestMove% represents the percentage of decisions in which the selection mechanism chose the move with the highest value. All experiments were performed across 200 games.

The amount of duplicate moves selected by the paired heuristics was also investigated. For the majority of the combined heuristics, the value was in the range 6-10, meaning that approximately half of the moves left unpruned were selected by both heuristics. Notable exceptions were $h4h2$ and $h4h3$, which had average duplications of below 5, perhaps explaining somewhat their poor decision making since the branching factor is large in this case.

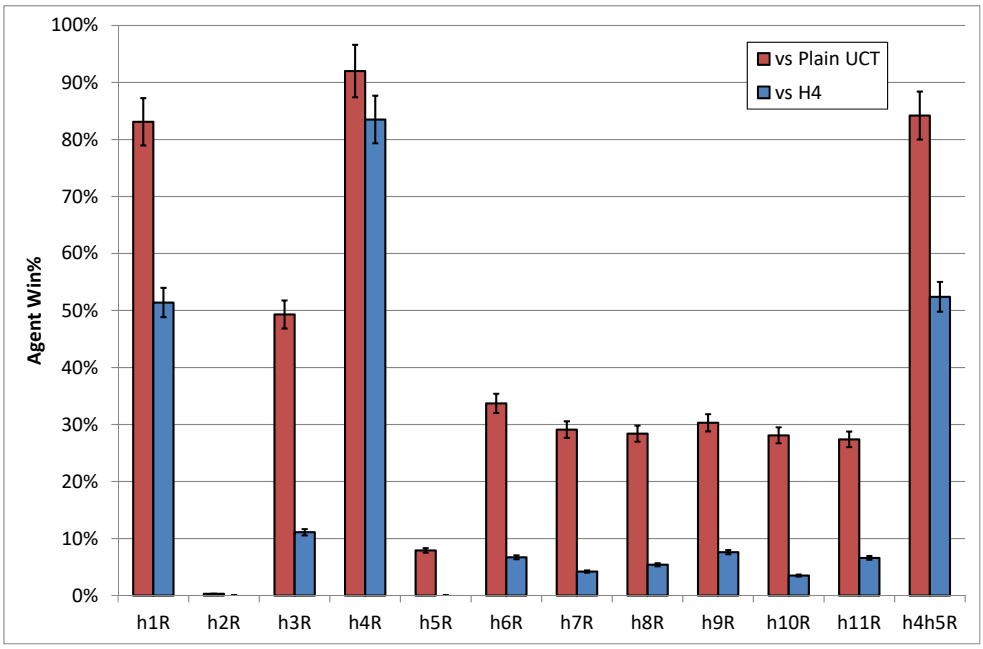


Figure 5.6: Mean win% of single heuristic agents h_1R - $h_{11}R$ and multi-heuristic agent h_4h_5R playing against a standard UCT agent and previous strongest candidate single heuristic agent (h_4).

5.4.4 Max Techniques

The results of experimentation using the Max value estimation ($\frac{w}{v}$) during selection are shown in figure 5.7. When using Max value estimation as a gauge of move strength, all agents behave worse than those using Robust value estimation (v). The only configuration that reaches close to 50% win rate is MaxRand_6, although the confidence interval in this result suggests that this could just be an anomaly.

MaxChild generally performed better when used in MaxRand rather than MaxRoulette, indicating that the order of the top N moves determined by MaxChild is likely not strongly related to play strength. This is supported by the sharp decline in play strength amongst the MaxRoulette agents when the configuration parameter is increased, and also by the fact that the indicated "best" move is selected with a higher frequency as this parameter decreases.

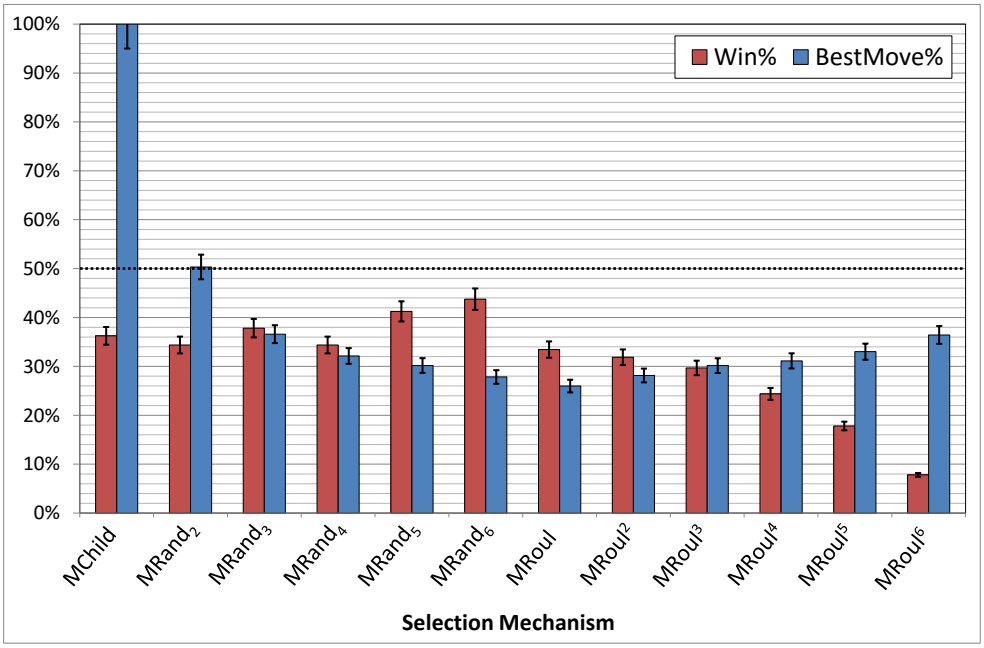


Figure 5.7: Win percentage and Best Move percentage of UCT agents using MaxChild, MaxRand$_n$ and MaxRoul^n against a standard UCT agent using RobustChild.

Due to poor performance, MaxChild was not considered in the later rounds of experimentation.

5.4.5 Robust Techniques

The results of experimentation using the Robust value estimation during selection are shown in figure 5.8. Robust (v) proved to be a much more indicative estimate of move strength, with a number of candidates winning 50% or more games.

Agents using RobustRand$_n$ show approximately 50% win rate while $n < 5$, and we can see that the agent in question is not always selecting the best move, which would cause non-standard behaviour from this agent, and therefore potentially generate some interest. As the value of n increases, we can see the rate at which the best move is selected decreases, as

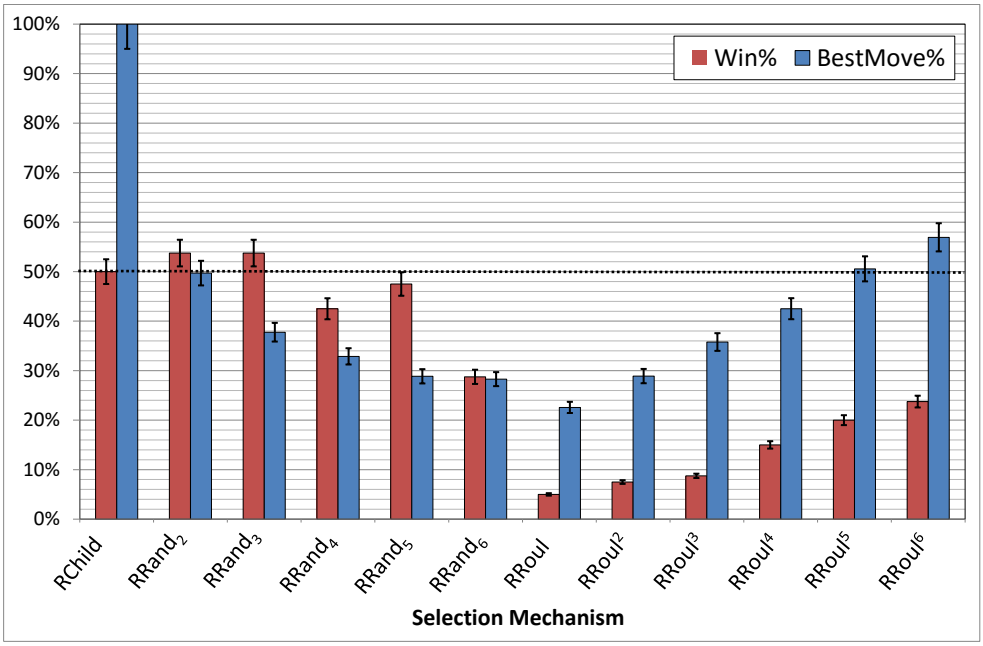


Figure 5.8: Win percentage and Best Move percentage of UCT agents using RobustChild, RobustRand$_n$ and RobustRoul^n against a standard UCT agent using RobustChild.

we would expect due to more moves being available for selection. The decrease is not as sharp as we might expect at higher values, and this is due to there are not always n moves available for selection (this explains the deviance of BestMove% at $n = 5$ from the expected 20%).

RobustRoulette^k shows an increased chance of best move selection as the value of n increases, but this technique is shown to select the best move only 50% of the time at the highest value of n, which may explain its poor behaviour. The win% follows the trend of BestMove%, further collaborating that Robust is a good measure of move strength. Further experimentation was performed to track improvement of win% with n, but it was found to tend towards approximately 30% and then no further improvement was found with further increase.

5.4.6 Further action selection mechanisms

Based on the perceived strength of Robust value estimation and the Roulette selection mechanism, two further selection mechanisms were tested, RobustRoulette$_n$ and RobustRoulette^2_n. These results are shown in figure 5.9.

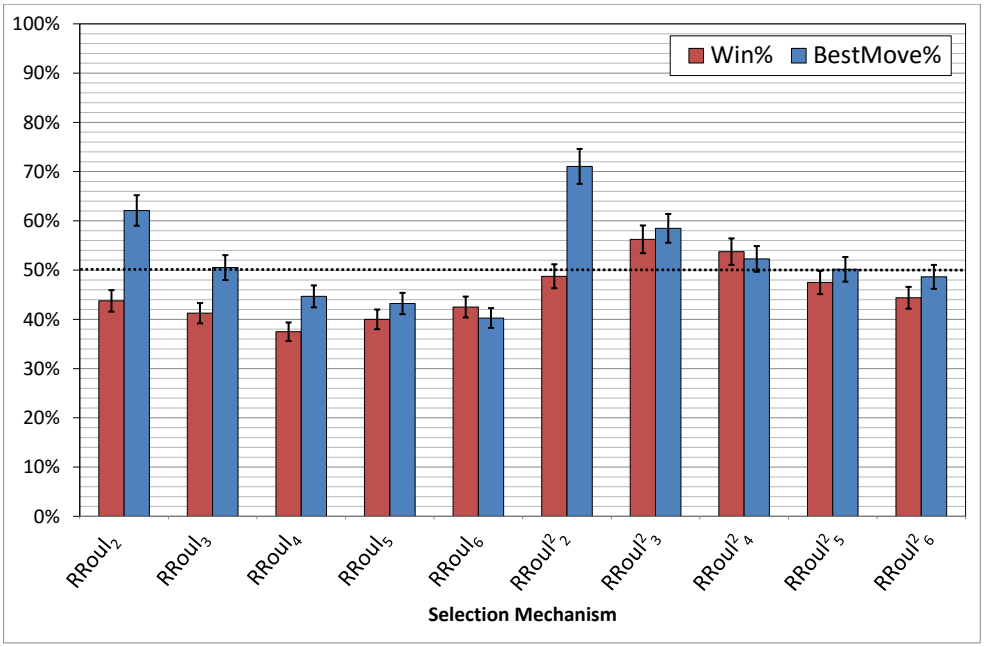


Figure 5.9: Win percentage and Best Move percentage of UCT agents using RobustRoul$_n$ and RobustRoul^2_n against a standard UCT agent using RobustChild.

The results with RobustRoulette$_n$ show a relatively stable win%, and a BestMove% of between 40% and 65%, which could represent a moderately strong player that selects non-traditional moves, and is potentially worthy of further exploration. The results using RobustRoulette^2_n show a slight improvement upon 50% win against our unmodified agent, which shows that the agent is potentially a formidable opponent which does not always choose the same move as the standard ISMCTS player. It is also worth noting that despite RobustRoulette$^2_2$ has a higher BestMove% than RobustRoulette$^2_3$, RobustRoulette$^2_3$ actually

performs slightly better. This is likely due to the ease of agents exploiting each other when they are playing the "best move", as these moves will be perfectly predicted by each agent's forward models. As such, a slightly lower BestMove% might actually result in a stronger result versus an agent playing 100% "best moves".

5.4.7 Varying Iterations

Two promising agents (RobustRoulette$_n$ and RobustRoulette^2_n) were carried through to the next stage of experimentation, in which the opposing agents were given a variety of different iteration budgets. Throughout this experimentation, the budget of the modified agent remained at 10000 iterations. The results for RobustRoulette$_n$ and RobustRoulette^2_n are shown in figures 5.10 & 5.11 respectively.)

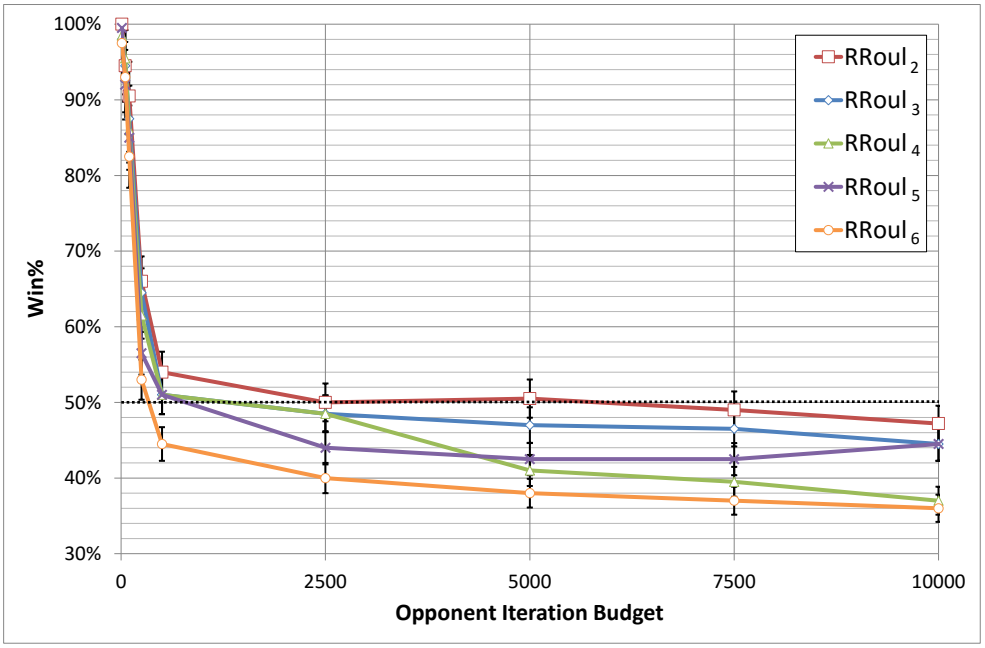


Figure 5.10: Win percentage of UCT agents using RobustRoul$_n$ against a standard UCT agent using RobustChild when varying iteration budget available to the RobustRoul agent.

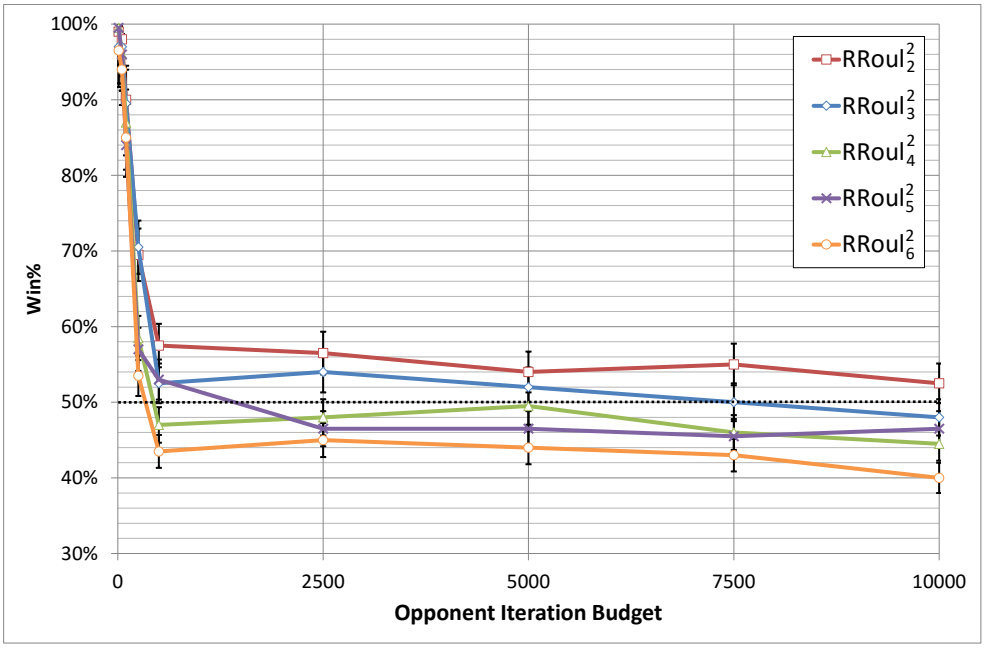


Figure 5.11: Win percentage of UCT agents using RobustRoul$_n^2$ against a standard UCT agent using RobustChild when varying iteration budget available to the RobustRoul agent.

These results clearly indicate a critical minimum budget for agents operating in this environment, as there is an observable drop off in win% before 500 iterations. The severity of this decrease in performance also indicates the unsuitability of simply using agents with reduced iteration budget as simpler opponents, as the play level behaves in a non-linear and unpredictable way as the number of iterations varies. It can also be said however that there is some smooth scaling exhibited by the gentle decrease in performance at the higher opponent budgets.

We can see from the displayed data that the configuration options most able to maintain a win rate of 50% are those using a low value of n, which suggests that staying within a close range of the best move is still required for strong play.

By comparing the performance of these agents across all iterations, we create the graph shown in figure 5.12. We can see from this graph that RobustRoulette$_n^2$ with a value of 2 or 3

appears to be a strong candidate for stabilizing play strength against opponent play strength.

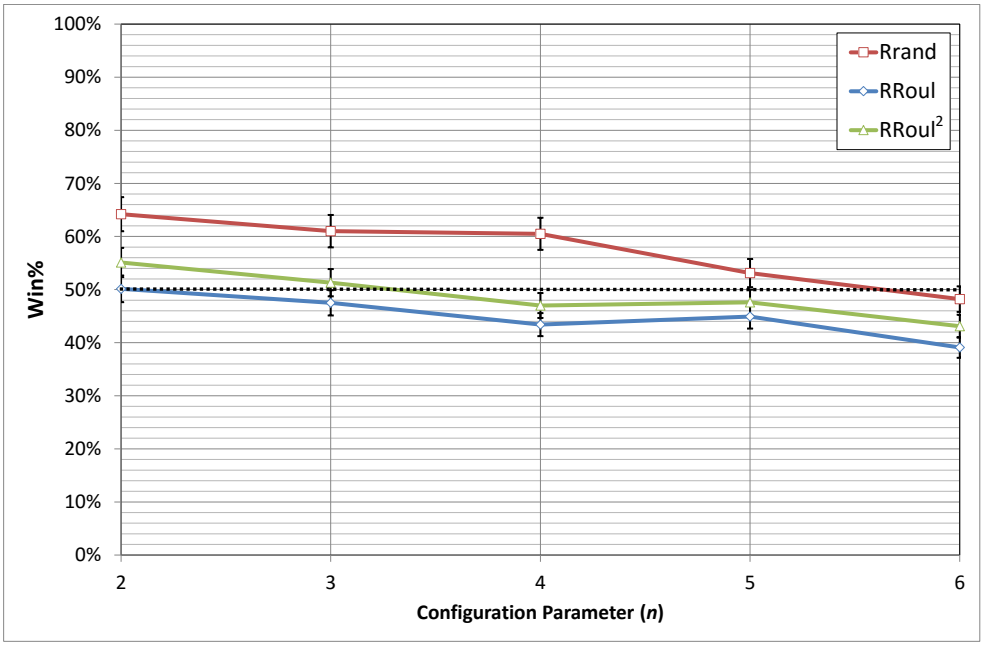


Figure 5.12: Win percentage of UCT agents using RRand, RRoul and RRoul^2 against a standard UCT agent using RobustChild across all iteration budgets while varying n.

5.4.8 Complexity Measure

The output from the complexity measures is displayed in figure 5.13. The outputs are normalised using observed maximum values for play, and a sample of the behaviour of RobustChild and a Random agent are included for comparison. We hope to see complexity measures which are in a midrange of the normalized values, which would indicate a moderate level of complexity, where as very high levels could indicate confusingly complex behaviour from the agent. We can see the massive difference in complexity measure between the RobustChild and Random agents, which speaks of their effectiveness in determining ordered play, as the Random agent should in theory be the least ordered player and the Ro-

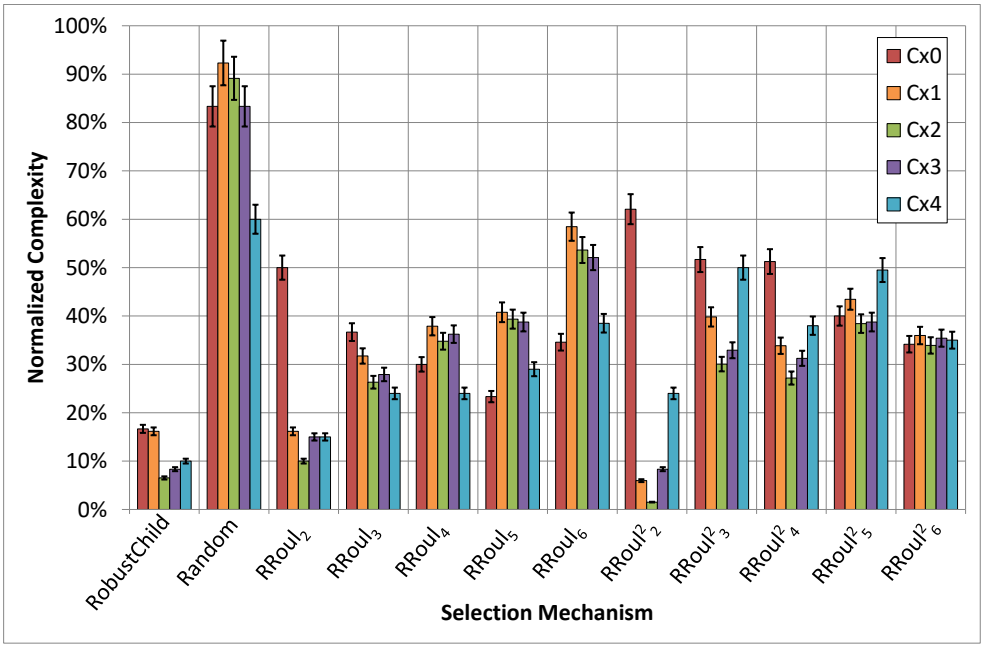


Figure 5.13: Complexity Measurements

bustChild should be very ordered and effective. Our experimental results support this, show that the random player displays relatively high values for all complexity measures when compared to our optimal player. It is likely that an agent which intentional performs moves to create complex board states would achieve even higher values of complexity on these measures.

The first thing worth noting is that all of the complexity values generated by the action of these two agents are significantly lower than the random agent and most are significantly higher than the RobustChild agent, so it is clear that there is an inverse correlation between win rate and our complexity measures. This is likely due to the fact that logical and effective play in Lords of War is to eliminate opposing cards, and this will cause a reduction in board state complexity. We can see this correlation particularly as the value of n increases in RobustRoulette$_n$.

The complexity measures for RobustRoulette$_n^2$ were all quite low with the notable exception of Cx_0 and (to some degree) Cx_4. Complexity measures Cx_1 - Cx_4 are closely tied to the number of player cards placed on the board, whereas Cx_0 is more associated with cards in hands. This suggests that this agent is in some way protecting played cards, most likely by boxing them into corners or against other friendly cards. Our own experience of playing against the agent certainly demonstrates this behaviour, however a further study of agent behaviour interpretation would strengthen this assertion.

5.4.9 Online tuning

Results for online tuning are shown in figures 5.14 & 5.15.

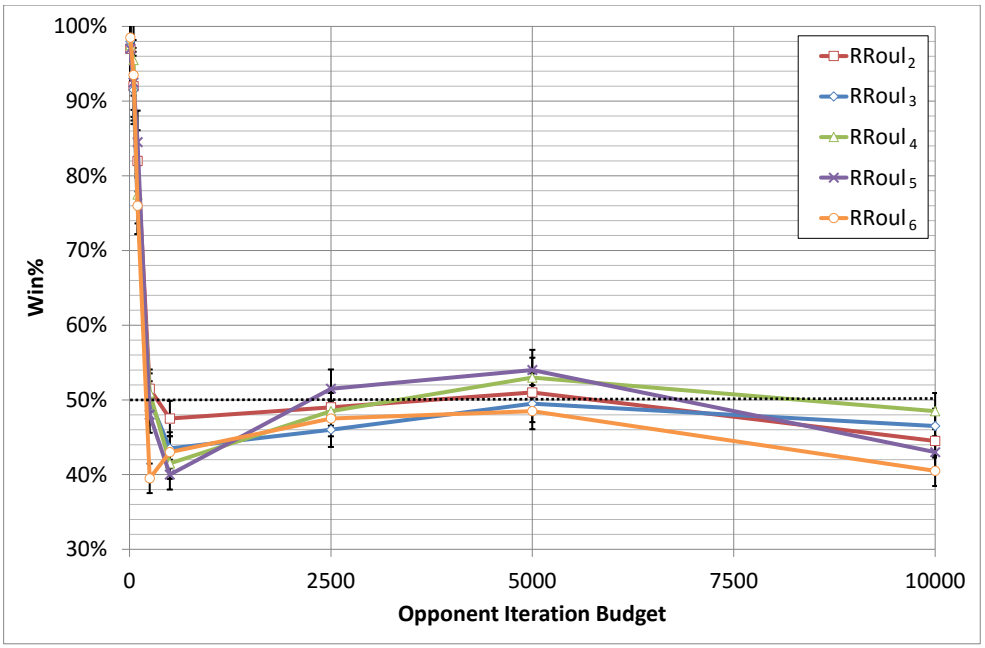


Figure 5.14: Win percentage of UCT agents using RobustRoul$_n$ against a standard UCT agent using RobustChild and the auto-tune modification.

The application of online tuning to the agents has created a slight curve in the results

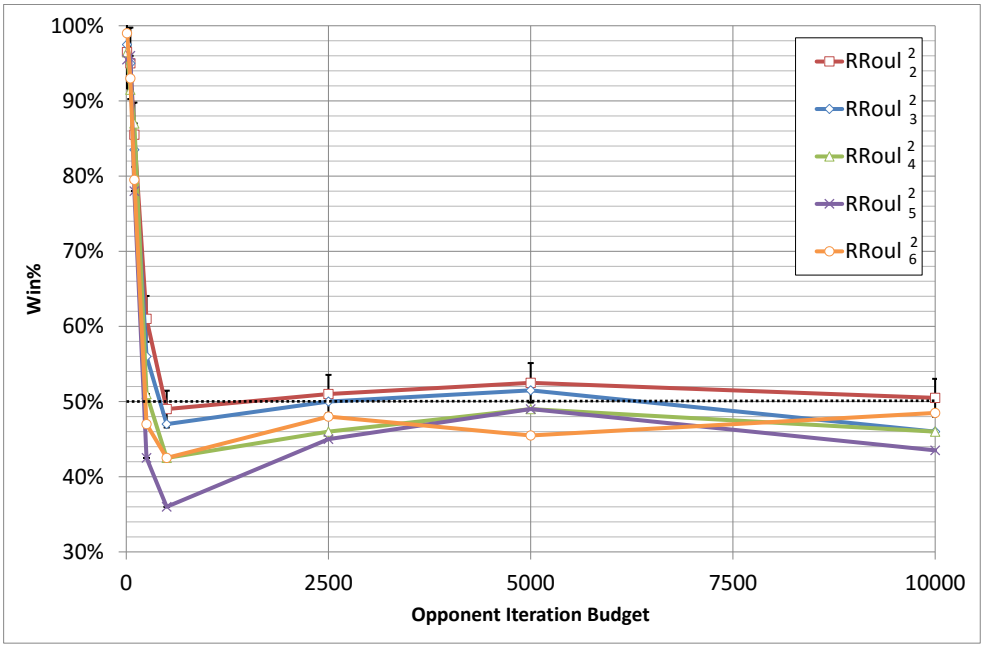


Figure 5.15: Win percentage of UCT agents using RobustRoul^2_n against a standard UCT agent using RobustChild and the auto-tune modification.

with a crest at roughly 5000 opponent iterations. This curve is also evident in the results of RobustRoulette^2_n, but it is less obvious. The tuning also appears to have had an averaging effect on all results, notably improving the lowest result by 4%, but also negatively affecting the more promising results.

5.5 Summary & Discussion

5.5.1 Heuristic Move Pruning

In this chapter, we experimented with heuristics in two general categories; heuristics that drew statistics from the cards in the game, and heuristics that used heat maps to prioritise card placement in specific positions. The idea is to make best use of limited CPU time by

focussing the search process on promising moves, and therefore maximise play strength. Overall the first category proved the most effective, particularly the simplest heuristics that merely totalled number of cards. The heat map heuristics were generally ineffective, however they did show the largest relative improvement from state-extrapolation, likely due to the additional evaluation step providing some much needed additional guidance after the heatmaps poor influence.

While state-extrapolation did have an effect upon playing strength in some cases, it was only effective in improving agent strength in certain cases, most notably when using heatmap heuristics h_{10} & h_{11}. This is possibly due to the fact that placing cards around the edges and/or the centre is strategically important (as our own experience would suggest), however the strategic impact is not always immediately apparent from the game state halfway through a player's move decisions. As discussed earlier, the heat maps alone were not expected to create strong agents, and the application of state extrapolation to h_{10} & h_{11} may have revealed that placing around the edges or centre is a strong move, and thus that these two maps are actually superior to the other heat maps.

5.5.2 Action Selection Mechanisms

The use of RobustRoulette selection mechanisms has a significant effect on the play strength of the ISMCTS agents. This is likely due to the approach of averaging across multiple possible states, which in the case of traditional selection mechanisms would cause a preference for more likely possible states, and potential ignorance of less likely states. Using a roulette technique here allows simulation of a mixed strategy approach to the statistical information presented by ISMCTS. Future work could be focussed on further exploration of these techniques to ISMCTS, and also the features of the game of study which could indicate appropriate configurations for such selection.

In terms of creating an agent that was a more entertaining opponent, we created multiple configurations that are worthy of future study, both playing a good game against our unmodified ISMCTS agent and generating some interesting complexity features. The most notable of these appears to be a RobustRoulette$_n^2$, which showed comparable play strength with the RobustChild agent, and generated good complexity (with the exception of the instance of RobustRoulette$_2^2$ mentioned above.) Further research should also likely include a study of human entertainment when playing against the agents, so as to determine actual human entertainment, and also examine further our metrics for complexity and their effectiveness.

It is also worth noting that agents which had fewer than 500 iterations available to them started to perform both extremely poorly, indicating the flaw in iteration budget scaling that we proposed originally. These agents also demonstrated behaviour closer to the stronger agents rather than the random agent, meaning their behaviour was also relatively simple (and therefore not of interest). A far more gentle decline was provided by the agents with the maximum available budget, but with various configuration options we displayed. In this way, we believe that a well configured agent would likely create a more appropriate opponent for a game requiring multiple difficulty settings.

The effect of online tuning appears to have been beneficial in scaling the modified agents against opposing ISMCTS agents using differing iteration budget. This suggests that this technique may be preferable to simply reducing iteration budget when balancing player strength with an opponent.

5.5.3 Contributions

Our primary contribution from our work in this chapter is the demonstration that modification of action selection mechanism and the application of heuristic pruning can create a variety of agents which display differences in play style according to our metrics for com-

plexity, yet maintain roughly the same win rate against a specific opponent. This highlights that agent behaviour in such games has many more dimensions than simple win percentage. To further clarify, there already exist cases of heuristic pruning and action selection mechanisms, and our work here expands those considerably, but does not suppose to invent the concepts themselves.

While the heuristics created are specific to the game Lords of War, then application of the development of these heuristics, the manner in which they are applied (using a hard pruning limit), and also the application of roll-forward (see section 5.3.4) and multi-heuristics are all transferable to other domains. The heuristics will be different, but the same techniques can be applied.

As with our previous work on parallelization, the techniques explored here could be useful in a wide variety of MCTS applications where enhanced performance (both in terms of playing strength and varied behaviour) would be an advantage. Pruning is a technique for reducing decision space, and thus accelerating the search process. As such, it would likely be applicable to similar domains as parallelization, assuming that appropriate pruning heuristics could be found for the domain in question.

Action Selection Mechanisms could be employed in any environment in which a variety of effective decision making is required. For example, in the games industry, a MCTS process with a modified action selection mechanism could create varied play in oppositional agents. This work is also important to the games industry (and of our industry partner, Stainless Games), as it demonstrates a simple technique for modifying an artificial agent to create different behaviour. This could be used in a game to simply create a variety of opponents that played in different manners, and demonstrated different strengths and weaknesses while maintaining a similar overall playing strength.

As industrial game AI advances, we can expect action selection mechanisms and heuristic pruning to be used frequently in industry implementations of MCTS. The advantages

these techniques provide in terms of easy modification of play style are too valuable to be disregarded.

I also see great potential in using the information generated by the MCTS process itself during the decision process (e.g. determining whether a specific decision is easy or hard based on the number of available choices that qualify for selection.)

Chapter 6

Rule Association Mining for Opponent Modelling in Games

A variety of games often include incomplete or hidden information as a form of challenge to the players, indeed games such as poker would be somewhat more trivial if such an element was excluded. Card games in which players bring decks of their own construction to play are now relatively common-place, and are represented both in physical card gaming (e.g. Magic: The Gathering[1]), and in digital gaming (e.g. Blizzard Entertainment's Hearthstone[2]). In such games, knowledge of the content of an opponent's deck represents a potentially powerful strategic advantage which can be exploited to significant advantage. This is true of competition outside of the game domain also, as being able to adequately predict the strategy of a potential competitor will likely give significant advantage.

In this chapter we consider a deck of cards to be a multiset consisting of a known number of cards, each of which has a type identifier. We then use a variety of rule-mining techniques applied with heuristic knowledge to attempt to predict the content of the deck after observing a specific number of cards chosen at random. It is important to note also that our game of choice is sufficiently complex, such that constructing a deck in the manner a human might is substantially more difficult than prediction using any method we have attempted here. Human players generally construct decks by identifying a central idea for the deck, then

[1] http://magic.wizards.com/
[2] http://us.battle.net/hearthstone/en/

fitting cards into the deck that either support that concept or appeal to the constructing player in some other way. While our techniques here produce similar results, there is no such central idea for each deck. Each deck is simply constructed from the association rules without a guiding deck concept.

It is important to note that the ability to model an opponent's actions exposes more information to an adversarial agent, and thus allows such an agent to become a stronger opponent. This applies to both traditional strength of play, and also to an agent which is trying to entertain a human player, as more information on the actual game state allows more informed choices, and thus more ability to make moves which affect human entertainment.

This research could also be applied outside the realm of games, to any similar, highly complex, partially observable system with specific rules which govern the system construction. Optimising association rule mining to these complex requirements is clearly of interest as a general advancement of research in this area. The techniques here could easily be converted for use in other fields which have similar complex requirements on sets or multisets, simply by applying heuristic knowledge to data mining and rule generation processes as performed here.

This work was published in the paper "Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner" appearing at IEEE CIG 2016 [121].

6.1 Experimental Methodology

6.1.1 Netrunner Deck Data

Experimentation data consisted of 6000 community made decklists posted on a popular Netrunner community website[3] that allows users to collect and compare decklists. Some

[3] `http://netrunnerdb.com`

filtering based on popularity was performed.

Algorithm 5 GetPredictedDeck(...) for a_1

```
1:  function GETPREDICTEDDECK(D_obs, R, C, n)
2:
3:      ##Initialise all cards with rule support
4:      InitCardRuleCounts(D_obs, C, R)
5:
6:      ##Sort cards desc by rule support
7:      sort(C, rulecount, 0)
8:
9:      ##Set predicted deck to include observed deck
10:     D_pred ← D_obs
11:
12:     ##For each card
13:     for all c ∈ C do
14:
15:         ##Take the required number of cards
16:         k = min{n − |D_pred|, c.MaxCount}
17:
18:         ##Add them to the predicted deck, if possible
19:         D_pred.AppendMultiple(c,k)
```

6.1.2 Apriori Rule Generation

Rules were mined from data using the Apriori method detailed in Agrawal & Srikant [3], with modifications as detailed in sections below. This process generates a large number of *rules*, which describe the relationship between items in the analysed set. These rules are made up of one or more *antecedent* items, and one *consequent* item. The antecedent items is a multiset of items which must be found in any observed set in order for the rule to become active. The consequent item is the item which results from rule activation, and thus the item which will be added to the predicted set. Our rules take the form of $S \rightarrow c$, where S is a multiset of antecedents, and c is the consequent.

Each rule also has a *support* [12] value, which states how many occurrences of the complete set of antecedents and the consequent appear in the training data, and is useful

to describe the magnitude of the effect of the rule. Support is calculated by the formula $supp(X \rightarrow Y) = |(X \cup Y)|/|N|$ [138], where $(X \rightarrow Y)$ represents a rule, and N represents the total size of the data set. Each rule also has a confidence value, which measures the reliability of the rule. Confidence is calculated by the formula $confidence(X \rightarrow Y) = supp(X \cup Y)/supp($

The primary piece of evidence used to model an opponent's deck will be the identity card, as it is always visible, and also provides the constraints for deck construction in the form of faction, side and influence. As other cards are revealed through play, these can be added to the deck with complete confidence. It is usual to have observed a small number of opponent cards during the first turn of play (we estimate 1-4 is usual), and as such we vary the number of observed cards we randomly select to determine the effectiveness of our technique upon different sized sets of cards.

After rules were generated from the data, the set of 6000 decklists were tested using 30 fold cross-validation, with each individual prediction being made based upon a set of randomly selected cards from the decks. As these cards could potentially be duplicates, for each test a minimum of two unique cards are observed.

6.1.3 Apriori Prediction

As discussed above, we employed several difference algorithms for rule generation. Our results are displayed in terms of match accuracy, which is calculated by determining the percentage of the original decklist which exists with in the predicted deck. As the deck size is fixed, this results in an match accuracy of 0-100%, where a 100% describes the situation in which the predicted deck exactly matches the original deck, and 0% describes a predicted deck with no instance of any card in common with the original deck.

6.1.3.1 Standard Apriori Prediction (a_1)

The standard Apriori method of prediction is shown in algorithm 5, where D_{obs} represents the observed known cards, n represents the size of the observed deck, D_{pred} represents the predicted deck, R represents the set of all generated rules, and C represents the set of all Netrunner cards. In the first step of the algorithm we set the rule counts of each card to 0, then we run through all rules and determine if they are active for the set of cards we have observed (D_{obs}). We then set D_{pred} to contain D_{obs}, as our prediction will always include the cards we have observed, and this makes further operations easier. We sort all cards by their $rulecount$ attribute, and then move through them in descending order of $c.rulecount$ until we find sufficient cards to fill the remainder of D_{pred}.

6.1.3.2 Modifying for duplicate cards (a_2)

A notable error performed by a_1 is number of duplicates which appear in the predicted decklists. As Netrunner decks can include up to three copies of each card[4], we attempt a technique that allows us to predict the number of copies of each item in the predicted multiset. Without this modification, the a_1 simply adds the maximum number of each item until it cannot add more, resulting in three copies of each card in the predicted deck.

In order to modify this behaviour, we make a separate calculation using the rule metadata to determine the number of duplicates included in the original data. We then use this information to include copies in the prediction. This algorithm is very similar to algorithm 5 except that after a card is selected, the rule metadata is averaged to determine the number of duplicates to be included.

Therefore each run of $GetPrediction\_a_2(D_{obs})$ adds 1-3 cards to D_{obs}, and bans the included card from further selection. This technique may appear arbitrary, but in the case of

[4]A few cards have specific rules which break this allow more copies or restrict the number of duplicates, but the vast majority may only appear in sets of 1-3

Algorithm 6 GetPredictedDeck(...) for a_2

```
1:  function GETPREDICTEDDECK(D_obs, R, C, n)
2:
3:      ##Initialise all cards with rule support
4:      InitCardRuleCounts(D_obs, C, R)
5:
6:      ##Sort cards desc by rule support
7:      sort(C, rulecount, 0)
8:
9:      ##Set predicted deck to include observed deck
10:     D_pred ← D_obs
11:
12:     ##For each card
13:     for all c ∈ C do
14:
15:         ##Take the required number of cards
16:         k = min{n − |D_pred|, c.Cardinality}
17:
18:         ##Add them to the predicted deck, if possible
19:         D_pred.AppendMultiple(c,k)
```

duplication in a specific decklist, the nature of the individual card is far more relevant than any patterns between the card and other cards in that deck. For example, some cards are so strong and usable in any deck that they almost always appear in sets of 3, whereas others frequently appear alone due to the narrow field of use or difficulty to fit into a deck.

6.1.3.3 Prioritising by Influence (a_3)

A review of the all data used here shows that 84% of decks in our dataset used all of their influence, 92% used all except 1 point, and 95% used all but 2. Considering that our data likely contains a large number of casual decks, which likely accounts for those not using all of the influence, this is indicative of how important the concern of influence during deck construction.

In order to prioritise influence spends, we change the method of deck prediction so that we first attempt to make predictions which would spend all available influence (both

influence and non-influence cards still undergo the duplicate procedure mentioned in section 6.1.3.2 above.) This new method is not shown in algorithm, as the only change is a sorting C so that all of the rules with a resultant card that will cost influence appear first, and this is restated later in algorithm 8. Notation is as before, however in the set C is sorted not only by rulecount, but also by a boolean that represents whether including any given card in D_{pred} would cost influence. This means that the first predictions made by a_3 will cost influence, and then when all the influence is spent, only cards that do not cost influence will be added.

6.1.3.4 Using influence during Rule Generation (a_4)

Here, we separate item sequences that were generated from influence spend and non-influence spend. This allows us to separate the item sets into two groups, one which represents cards which players have spent influence on, and one which represents card sequences that were used "in-faction". We can then generate specific rules for influence and non-influence spend. In the case that we had insufficient data, the prediction reverted to using all generated rules. This method is shown in algorithm 7. Notation is as before, however in addition R_{inf} represents rules originally generated from influence sets, and R_{noinf} represents rules which are generated from non-influence sets only. This algorithm is very similar to algorithm 6 except that $GetPrediction\_a_4$ uses only rules generated from influence selections when selecting an card that costs influence, and only rules generated from non-influence selections when selecting a card that does not cost influence.

6.1.3.5 Rule Generation including duplicate cards (a_5)

We also attempted to remove the calculation for duplicate cards by allowing the rules to be constructed from duplicate items, and thus we should be able to predict those duplicates with more relevancy to the observed deck, rather than the general attributes of the cards.

Algorithm 7 GetPredictedDeck(...) for a_4

function GETPREDICTEDDECK(D_{obs}, R_{inf}, R_{noinf}, C, n)

    ##Initialise all cards with rule support (inf)
    InitCardRuleCounts(D_{obs}, C, R_{inf})

    ##Sort cards desc by rule support
    sort(C, rulecount, 0)

    ##Set predicted deck to include observed deck
    $D_{pred} \leftarrow D_{obs}$

    ##Spend influence first
    for all $c \in C$ **do**

        ##Take the required number of cards
        $k = min\{\lfloor (maxinf - inf(D_{pred}))/c.inf \rfloor, c.Cardinality\}$

        ##Add them to the predicted deck, if possible
        D_{pred}.AppendMultiple(c,k)

    ##Initialise all cards with rule support (no inf)
    InitCardRuleCounts(D_{obs}, C, R_{noinf})

    ##Then fill the deck with non-influence cards
    for all $c \in C$ **do**

        ##Take the required number of cards
        $k = min\{n - |D_{pred}|, c.Cardinality\}$

        ##Add them to the predicted deck, if possible
        D_{pred}.AppendMultiple(c,k)

This algorithm is identical to algorithm a_4, except that duplicates are calculated based on the number of copies of each card seen in the generated rules rather than our cardinality data. When a rule is determined to be active, instead of checking rule metadata to determine the number of cards to add to the predicted deck, we instead determine the total number of the consequent item that already exist within the predicted deck, and if the required number specified by the rule already exist, we take no action. If the required number is not yet in the deck, we add a single consequent item. For example, if the rule $\{A, B, C\} \rightarrow B$ becomes active, we check to see if 2 or more B are included in the predicted deck. If so, we add nothing. If not, we add a single B.

6.1.3.6 Prioritising by rulesize (a_6)

This modification attempts to give priority to rules which contain more items, as these rules will be less rarely active due to their specificity. However, when these rules are active for an observed card set, they will likely tell us more about the content of the deck than smaller rules. This algorithm is identical to a_4, except that the rules are sorted by descending rule size, and then a_4 is performed using the set of rules which are the largest size, then descending through the rules until we have completed the deck.

6.1.3.7 Making confident predictions (a_7)

This modification is identical to a_6, however when we predict a card, we add it to the observed card set and check all rules again. So any card we predict to appear in the deck, we assume we are correct for the purposes of further predictions. This final version is shown in algorithm 8.

Algorithm 8 GetPredictedDeck(...) for a_7

```
1: function GETPREDICTEDDECK(D_obs, R_inf, R_noinf, C, n)
2:     D_pred ← D_obs
3:     for all r ∈ R_inf do
4:
5:         ##Initialise all cards with inf rule support
6:         InitCardRuleCounts(D_pred, C, R_inf)
7:
8:         ##Sort cards desc by rule support
9:         sort(C, rulecount, 0)
10:
11:        ##Spend influence first
12:        for all c ∈ C do
13:
14:            ##Take the required number of cards
15:            k = min{⌊(maxinf − inf(D_pred))/c.inf⌋, c.Cardinality}
16:
17:            ##Add them to the predicted deck, if possible
18:            D_pred.AppendMultiple(c,k)
19:        for all r ∈ R_noinf do
20:
21:            ##Initialise all cards with non-inf rule support
22:            InitCardRuleCounts(D_pred, C, R_noninf)
23:
24:            ##Sort cards desc by rule support
25:            sort(C, rulecount, 0)
26:
27:            ##Fill out deck with non-influence
28:            for all c ∈ C do
29:
30:                ##Take the required number of cards
31:                k = min{n − |D_pred|, c.Cardinality}
32:
33:                ##Add them to the predicted deck, if possible
34:                D_pred.AppendMultiple(c,k)
35:    return D_pred
```

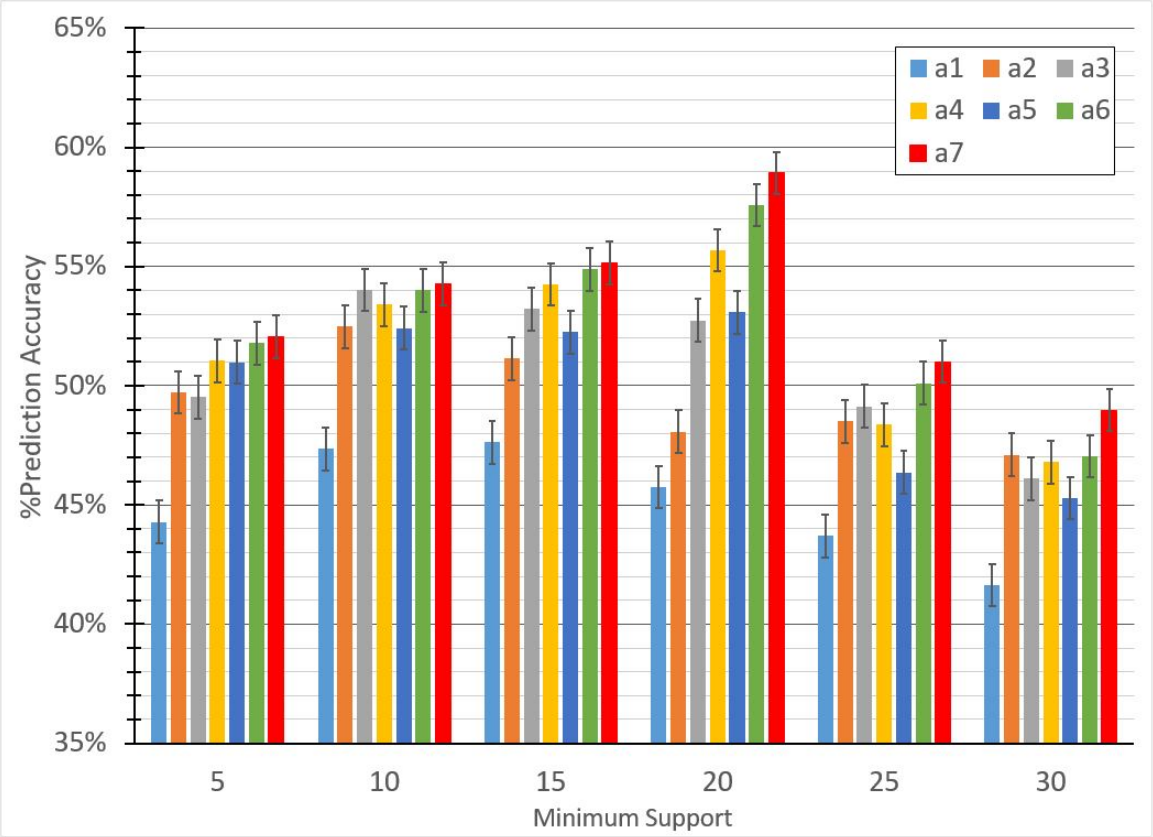


Figure 6.1: Percentage match accuracy of apriori prediction using agents a_1 - a_7 across different values of minimum support.

6.2 Results

Prediction results are shown in figure 6.1. Use of the mentioned techniques to generate deck predictions is generally successful, completing decks with an match accuracy of up to 59% from viewing only 5 cards (roughly 8-10% of the actual deck). However there are some general trends which can be observed. Firstly, as each card (or set of cards) is added to the deck sequentially, we do not take into account new patterns which may emerge between originally observed cards and cards more recently added. This means that all predictions are based on the original set of observed cards, whereas we would likely have a different effect on prediction if we considered predicted cards to be part of the observed set when making further predictions. We suggest that some of the difference in prediction may be

a tendency to form into familiar deck archetypes, as predicted cards would likely support larger patterns already recognised as frequently played decks. This is somewhat consistent with human deck construction however, as players often use existing archetypes to construct decks.

In order to provide a control for experimentation, random selection was tested (a_0). Generated decks were still required to observe deck construction rules, but other than that cards were selected randomly from the set of available cards. All predictions using a_0 had an match accuracy in the range 0% - 6%, and due to this low match accuracy, results are not shown below.

We also attempted to test prediction across a range of different numbers of observed cards. In each of these cases, the identity card was always observed, then an additional number of cards were added. This means in the case of the number of observed cards being zero, only the identity card was observed. In all previous experiments the size of the set has been five, which represents what a player might expect from two complete turns of play. We tested prediction with sets of up to ten viewed cards. We also tested prediction with a set of zero observed cards, which represents the game before play has begun.

6.2.1 Default Apriori (a_1)

Default Apriori allows for predictions of up to 48% match accuracy, and while this is somewhat effective, it can be improved upon significantly by the later algorithms which incorporate heuristic knowledge. Different values of minimum support were used to determine the optimum value, which lies close to 15. All of these tests were run on a dataset of size 200 (30-fold cross-validation on a total set of size 6000), so larger values of minimum support will likely cause smaller detail of the dataset to be lost during rule generation. Examination of the decks generated with a_1 also reveals that almost every card is included in triplets,

further speaking of the necessity of a modification to address the number of duplicates included.

6.2.2 Apriori with duplicates (a_2)

The modification to consider inclusion of duplicates in the predicted deck results in a significant increase in match accuracy. The most significant value of minimum support now appears between 10-15, both options resulting in a prediction accuracy of 53%, an increase in accuracy of 5%. This increase in match accuracy is certainly related to more accurate predictions on sets of duplicate cards, as due to the nature of the game, certain cards are more often played in sets of 2 or 3, and certain cards are almost always played without duplicates. This modification largely makes the effect that there are no longer automatic inclusions of cards in groups of 3, however it can still be further improved with respect to heuristic data.

6.2.3 Apriori with Influence Priority (a_3)

While prioritising the inclusion of cards which cost influence has a positive effect, the effect is marginal, increasing match accuracy by less than $\sim$2% at the optimal value of minimum support 10. It is surprising that the effect is so marginal, but upon examining further it is apparent that most (92%) of decks predicted with a_1 and a_2 already include the maximum permitted influence for those decks, so the modification is perhaps not as important to prediction as originally proposed.

Examinations of the individual card selections shows that the influence spends are somewhat inappropriate however, and are somewhat to blame for the inaccuracies of this prediction algorithm.

6.2.4 Apriori with Influence Filtering (a_4)

There are several interesting effects in these results. Firstly, the highest match accuracy has risen to 57%, an increase of $\sim$4%. Secondly, the optimal value of minimum support has changed to a higher value of 20.

A review of the cards selected by influence spends reveals that they are much more appropriate to the acknowledged deck archetypes, presumably due to the specific use of rules generated entirely from influence spend patterns.

We also start to observe some occasional single-card influence inclusions which are well established in the appropriate archetypes.

6.2.5 Rule Generation including duplicate cards (a_5)

We can see from the results for a_5 that attempting to determine the number of duplicate cards in a deck from generated rules appears to be less effective than using our data on the normal set count of that card. This is believable, as the number of duplicate cards included is likely to be much more dependent on the nature of the card than on the nature of the deck itself. As our information relates to patterns between cards, we do not necessarily have a good understanding of the nature of the card itself.

It is worth noting however that for some values of minimum support, a_5 is approximately as effective as a_3 and a_2, meaning that it is still an effective technique, and alternative methods to predict duplicate cards in the deck could be investigated.

6.2.6 Prioritising by rulesize (a_6)

Giving priority to larger rules has also had a positive effect on match accuracy. We can see this effect particularly when minimum support is 20. We attribute this effect to larger rules being more rarely satisfied unless they are highly informative about the configuration

of decks. As such, activated large rules should be given priority over activated smaller rules.

6.2.7 Making confident predictions (a_7)

By adding all predictions to our observed set, we are assuming that all our predictions are correct, and biasing future predictions by this information. This has a positive effect on match accuracy at higher values of minimum support, however it has almost no effect at values of 15 and below. This could be explained by some subtly of rules that are activated with a support of 15 or less, however in this case we would expect the match accuracy to be positively affected also, and yet we see that this is not the case.

The extension of our observed set also has another less obvious effect on prediction, which is that it allows activation of rules with larger item sequences, as more items appear in the observed set. This means as D_{obs} expands, we may observe decks activating larger rules, and effectively falling into archetypes.

6.2.8 Varied Size Observation Set

The results for predictions made with varied observation sets are shown in figure 6.2.

We can see that the overall change in match accuracy across the total range of tested values is approximately 20%, which while a large change, might be less than we expect from such a change in source data. This illustrates the importance of the identity card which is always viewed, it speaks deeply of the construction of the deck, mostly because the identity card is always active during play, and a substantial portion of the cards included will have some synergy with that identity. This also speaks of the nature of deck construction in Netrunner, which largely consists of modifications to existing archetypes, likely due to smaller synergies between groups of cards. It is also worth noting that at almost all values of observed set size and minimum support, our algorithms which incorporate heuristic

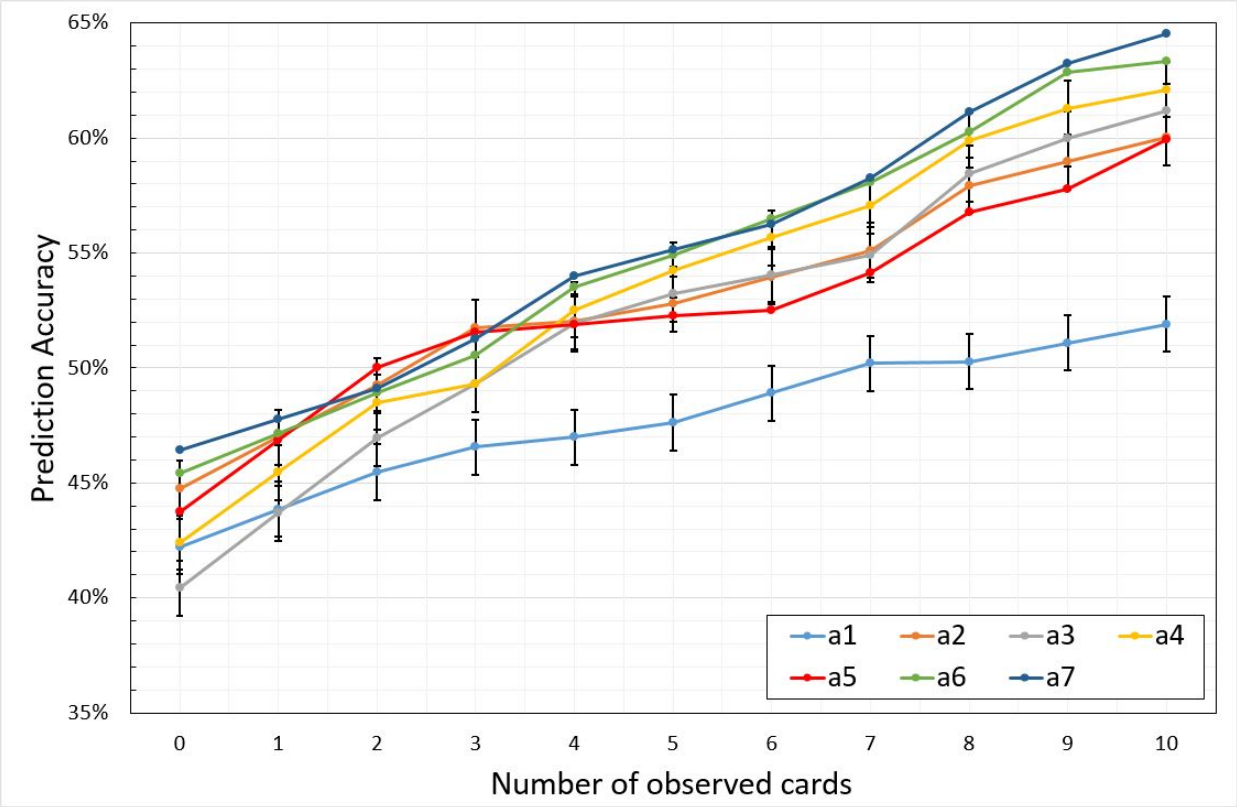


Figure 6.2: Percentage match accuracy of apriori prediction using agents a_1 - a_7 when using different numbers of observed items.

information perform significantly better than default apriori.

We see an understandable increase in match accuracy as we increase the size of the observed set, as there are both fewer cards to predict, and also more information is available on the set content. Rules with a higher number of antecedents are also activated, which likely provides more accurate information on the set content.

We can also observe that a few of our own techniques (a_3&a_4) perform very poorly when the observed set is very small or empty. As a_3 and a_4 both focus on influence inclusions, this is likely due to a lack of corroborating information from other observed cards to distinguish correct influence selections. As such, the initial influence selections are almost unguided, and as these cards are selected from a much larger set of available cards than regular selections, the picks are more likely to be incorrect without guidance.

There is also an interesting plateau in match accuracy around set size 3-6 with algorithm a_5. This is likely due to the estimation for duplicate cards struggling on smaller set size. As the cards in the observed section of the deck are selected randomly during each test, it is possible that duplicate cards are selected, and as such less information is exposed in certain cases. This might cause a decrease in accuracy when only a small number of unique cards are observed. This calculation is not included in any other algorithm, as it was not effective in increasing accuracy overall, possibly due to this complication.

The results across all experiments grouped by algorithm are shown in figure 6.3.

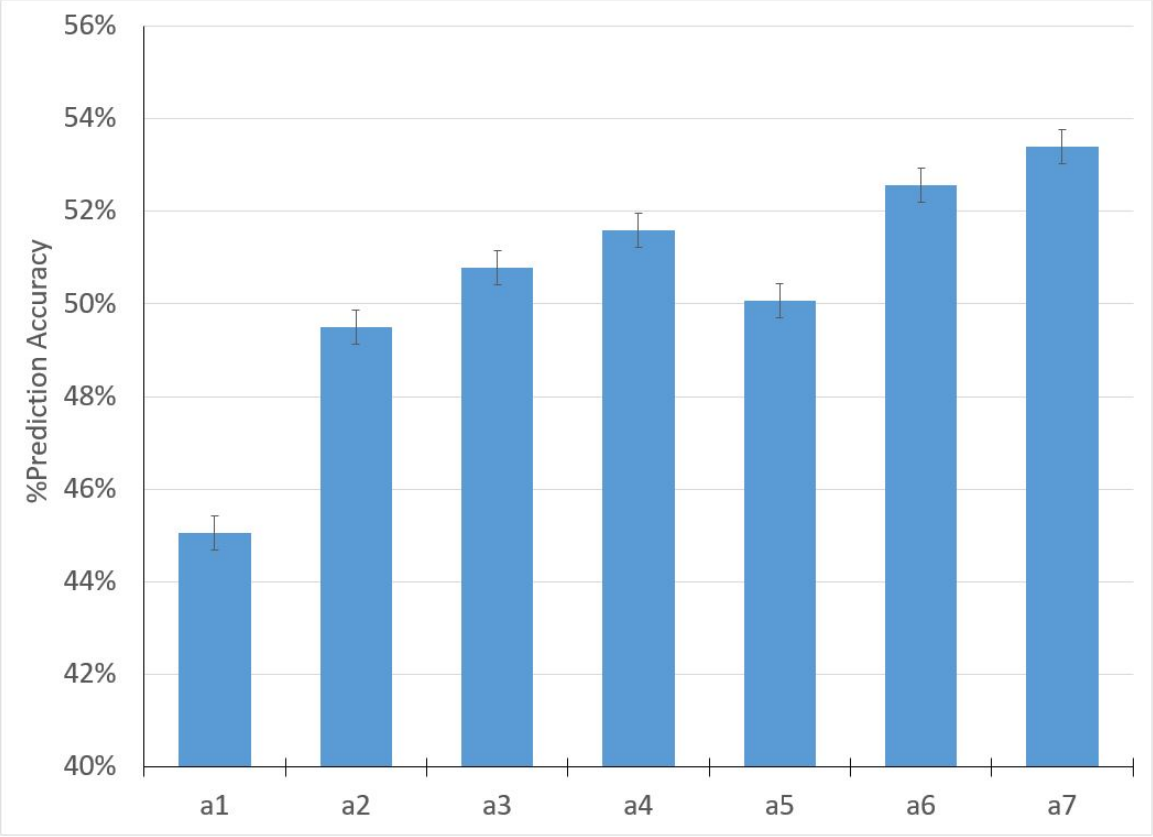


Figure 6.3: Cumulative match accuracy of apriori prediction using agents a_1 - a_7 across all experiments.

We can more clearly see a general rise in match accuracy here, with the exception of the a_5 algorithm for reasons mentioned above. This is to be expected, as each algorithm following a_1 includes specific heuristic improvements which are targeted to improve efficiency in

this specific domain.

Algorithm a_5 shows that our introduction of rule-based cardinality estimations have been unsuccessful in improving prediction efficiency, although this is something we would definitely want to address in future. The current cardinality estimations are unlikely to predict decks with 100% accuracy, for example it will always fail to predict decks that include a unusually small number of a card almost always seen in sets of 3.

6.2.9 Deck Creation

As a final experiment, we attempted to create a selection of novel decklists by providing a small set of cards and allowing the algorithms above to generate the decks. In each case, a single Identity card was provided, along with three other cards in the correct faction which are commonly played in tournament play. The generated decks were then shown to a group of experienced players, who were asked to criticise the decklists. All of the generated decks are listed in Appendix B.

In each case, the experienced players were content that the decks were reasonable for play in a casual or semi-competitive environment, but may struggle in a highly competitive tournament environment. They agreed that there were no completely inappropriate inclusions, however there were a few questionable choices, and in some cases it appeared that some card combinations had been left unfinished in the decks. When asked to provide modifications to the decks to improve their quality, the players each suggest 1-3 card exchanges, stating that those changes would likely make the decks worthy of play in a more competitive tournament.

6.3 Summary & Discussion

Our attempts to adequately predict the number of duplicate cards within a deck have been some what effective, but there is still work to be done here, as our best prediction is based on our heuristic knowledge of the specific card, rather than knowledge of the card in context. Successfully adding contextual heuristic knowledge into this process will surely lead to more accurate prediction.

Given that experienced players found our decks to be at least worthy of consideration for play in a tournament, there is definite potential in taking this technology forward, and potentially creating tools for the Netrunner community to generate and validate their decks. With a little adaptation, this can also be taken into other similar games.

As discussed earlier in this chapter, knowledge of the content of an opponent's deck represents a potentially powerful strategic knowledge which can be exploited to significant advantage. As such, we would expect an agent which can successfully predict the content of an opponent's deck to be significantly stronger than an agent which was unable to do so. It is also worth noting however, that a misprediction could cause a significant reduction in agent strength.

6.3.1 Contributions

Our principle research contribution from our work on Rule Association Mining is a substantial improvement in deck prediction from the default apriori algorithm. It can be seen that our modifications to the Apriori technique provide a significant improvement to prediction of decks in Netrunner, showing a maximum improvement of $\sim$13% between the default apriori algorithm (a_1) and our optimal modified algorithm (a_7).

We have shown that the apriori algorithm is applicable to NetRunner, Living Card Games, and also any card game that contains constructable decks. As discussed previ-

ously, the ability to predict opponent decks can be a powerful strategic advantage, but this can also be leveraged in other ways. An oppositional agent that attempts to predict the human player's deck rather than "cheating" and looking is likely to feel more fair (and more interesting) to the human player, assuming this is correctly measured. This contribution goes further than merely deck prediction, as these techniques can also be used as a form of deckbuilding aide, to help new players build decks and guide them as to what might be sensible inclusions in their first decks. Given that deck building is a challenging task for human players, any help provided by an machine learning agent would likely provide significant learning for new players and assistance for experienced players in the process of building a deck.

Similar principles could also be used in other computer games such as MOBAs (Multiplayer Online Battle Arena) and RTS (Real-time Strategy) games. Both these archetypes of games use highly customisable and configurable *Build Orders* which dictate the sequence in which actions should be taken for optimal performance. Determining build order could be viewed as a similar process to building a deck, and addressed in a similar manner.

Chapter 7

Conclusions & Further Work

In this thesis, we have investigated the application of specific artificial intelligence techniques to creating differing playstyle in games. Our principle body of work has been the modification of MCTS in order to affect play style, and thus create entertaining or interesting differences in play between a modified agent and a unmodified agent. In order to do so, we have considered a variety of techniques including heuristic pruning and the modification of the action selection mechanism. This work is motivated in large part by the needs of the commercial games industry and particularly those of my sponsoring company, Stainless Games. Commercial games need entertaining AI opponents, not merely strong ones, and this is an area ripe for research, with little work done to date.

Our work in heuristic pruning and action selection mechanisms (see chapter 5) showed that we could create different behaviour in an artificial agent without diminishing play strength. If our approximation of entertainment through complexity holds, then we have also shown that we can make agent play more entertaining to humans through these modifications, which has real implications for game AI.

We have also explored the application of data mining techniques to the prediction of hidden information in complex card games, an AI agent with more information can operate

more efficiently, and thus can make better, more interesting decisions (whether the metric for those decisions be play strength or complexity of play.)

Below we present and summarise our research contribution, and consider their generalisation. Then we outline a plan for future work based on some of the topics explored in our work here. It is not our plan to execute all these plans, but we leave them here as a indication to others who may want to continue this work. Work already under way is some of that detailed in section 7.3.4, as the application of our techniques to other games has already provided interesting results.

7.1 Research Contributions

7.1.1 Parallelization of Monte Carlo Tree Search and Information Set Monte Carlo Tree Search

In section 4.3, we present our contribution towards the parallelization of Monte Carlo Tree Search (MCTS) and Information Set Monte Carlo Tree Search (ISMCTS). This work was published at the 2014 IEEE Congress on Evolutionary Computation (CEC) [123]. It is of significance to the research community, as at the time of writing it is the only work surveying the effect of parallelization techniques upon ISMCTS. It also provides a complete survey of commonly used parallelization techniques for MCTS.

Given the increase in the number of processors in modern hardware, parallelization is of growing importance, and research into the operation and effectiveness of artificial intelligence techniques on such hardware is therefore also important. Our work shows that effective use of parallel processing will directly result in a more effective MCTS and ISMCTS artificial agents, and measures the effect of adding processes on speed up and playing strength.

This research is particularly of importance to the games industry, (and in particular our partner, Stainless Games) as in the past AI has been assigned a low priority in game development, and thus the access to additional resources provided by advanced parallelization techniques will in the future allow more effective artificial agents to be implemented.

A significant weakness of our current work here is that it does not sufficiently investigate play strength of the resultant agents, only focusing on decision time. While we did some cursory experiments on play strength, more thorough investigation is required.

There are several potential forward directions for this work. The most obvious addition would be an investigation of the differences in play style and play strength of both UCT and ISMCTS agents across all four parallelization techniques. While we extensively explored the efficiency of the agents in UCT/ISMSTC, there is more room to explore the effect of parallelization upon the behaviour of other flavours of MCTS, especially those that reuse information between iterations and hence may be more difficult to parallelize [141].

Another promising avenue would be to explore the parallelization across the GPU, which has become a more available target for non-graphical processing over the last few years. There is some research into this field already [14, 113, 114], and following up on this work in combination with our work here would be valuable.

7.1.2 Modification of Monte Carlo Tree Search using Heuristic Hard Pruning

In section 5.5.1, we present our contribution towards applying Heuristic Hard Pruning to the modification of Monte Carlo Tree Search agents. This work was published at the 2014 Conference on Computational Intelligence and Games (CIG) [122]. This work is of significance to the research community because it demonstrates the effect that Heuristic Pruning has upon the strength and style of play from an artificial agent. This establishes that prun-

ing techniques can be used to improve playing strength, and offers a number of avenues of research going forward.

This work is also of significance to the games industry (and of our industry partner, Stainless Games), as heuristic pruning is used to optimise artificial intelligence agents, and our work both explores these techniques on a new domain, and also highlights that the pruning may have unforeseen consequences upon the style of play.

Our work here explores hard heuristic pruning in Lords of War, but given more time, we could have further explored the usability of the heatmap heuristics that are based on the geography of the playing board, and which showed some promise in combination with other heuristics. It is likely that this style of heuristic has a strong effect on positional board games, and further investigation would likely prove valuable [31]. It would also be possible to determine heatmaps from human versus human play data and thus have more advanced interpretations of the board than those we use in our work here. Similarly, further research into Lords of War play would likely yield more interesting and effective non-heatmap heuristics which could be used to advance our research.

While our target here has been a specific card game, this work is generally applicable to any card game or positional board game with a similar branching factor. This work can also be applied to any MCTS decision making process which would benefit from the addition of heuristic information.

7.1.3 Modification of Monte Carlo Tree Search play style using Action Selection Mechanisms

In section 5.5.2, we present our contribution on Modification of Monte Carlo Tree Search play style using Action Selection Mechanisms. This work was published at the 2015 IEEE Conference on Computational Intelligence and Games (CIG) [124]. This work is significant

to the research community because it demonstrates the effect of modifying the standard MCTS action selection mechanism, and through doing so creates a variety of agents which have very different play styles, but maintain roughly the same win rate against a specific opponent. This highlights that agent behaviour in such games has many more dimensions than a simple win percentage.

This work is very important to the games industry (and of our industry partner, Stainless Games), as it demonstrates a simple method to modify an artificial agent to create different behaviour. This could be used in a game to simply create a variety of opponents that played in different manners, and demonstrated different strengths and weaknesses while maintaining a similar strength.

Our target in this case was the Lords of War game, but this work can very easily be used in any game or process which is using MCTS, as none of the technology uses heuristic information specifically designed for Lords of War. Effectively we were just using Lords of War as a demonstration of a aheuristic modification to MCTS, which is applicable in any domain where MCTS would be applicable, and a variation in style of decision-making is relevant.

Further work in this area would continue the exploration of the differences in play between the modified agents. Our complexity measures were a good initial indicator of difference in play style, but there is more to be examined in the specific effects of the action mechanisms upon play style. Given that our complexity measures were primarily measures of randomness in play rather than measures of a specific style of play, further research could determine the actual patterns encouraged by each action selection method, and whether there is a specific character which can be encouraged through use of each. It is also worth noting that further exploration into the effectiveness of our complexity measures might result in advanced measures which allow incorporation of user preference and experience, allowing tailored play experiences and learned behaviours that are liked or disliked across a range of

players.

7.1.4 Application of Data Mining Techniques to prediction of opponent decks in card games

In section 6.3, we present contribution on the Application of Data Mining Techniques to prediction of opponent decks in card games. This work was published at the 2016 IEEE Conference on Computational Intelligence and Games (CIG) [121]. This work is significant to the research community because it applies the well established data mining technique *a priori* to a new domain for which it is well suited. We then modify the default a priori using heuristic information specific to the domain in question and show significantly better rates of deck and card prediction.

Over the last decade, there has been a substantial increase in the number of card games which have been adapted to the digital environment. Many of these games now have large player bases which create huge amounts of data on their play. The techniques presented in our work provide a variety of opportunities both for these players and for the industries which cater to these players. While we have used the a priori algorithm to predict deck content, it can also be used to generate new deck designs, and likely to predict upcoming trends in the data which correspond to new set releases and rotating cycles of cards, and hence assist in the design and make up of new card sets.

The work here holds a variety of potential applications, the most obvious of which would be a software project which provides players with access to the knowledge generated from the game community's play. For example, a Netrunner player would be very interested in both accessing the crowd knowledge on what cards their opponent is likely playing, and also on receiving crowd knowledge advising cards to include in their latest deck.

There is a massive potential for expansion in this work, as if an engine for interpreting

card text (or a game forward model) became available, then this work could be used to determine likely patterns of play for unprinted cards, and therefore as an aide to the publishing companies when they are considering new cards. Of course this requires some heuristic knowledge, but the techniques that made the prediction could remain aheuristic, relying upon the card interpretation engine to provide heuristic data on the individual cards.

7.2 Summary and General Observations

Our work here combines a variety of techniques which modify or improve artificial agents with the aim to improve human experience when playing digital games. We have worked in several different areas, all of which are described and summarised below.

Our work on Action Selection Mechanisms represents the culmination of our research upon creating an agent which has varied play, and yet is still competitive. The development of more entertaining, varied and possibly human-like artificial intelligence is of critical importance to the games industry. As mentioned previously, the continued improvement of AI techniques will cause a stronger demand for more human-like play from AI agents. This is somewhat due to demand for AI to make more human-like decisions, but also to provide a better environment for game learning and a more friendly interface. The objective of any AI in a commercial game should be to provide a challenging, entertaining experience, which may include defeating the player, but should not be confined to that sole objective. Our work here has shown that there are multiple configurations of AI agent that behave differently but maintain roughly the same win rate, which would be of significant interest to any industrial game producer.

Our work on Association Rule Mining represents an exciting new application of technology to a new domain provided by an widening market in online gaming. While our research here is confined to a single game domain, collective card games (CCGs) are rising

in popularity following the success of games such as Hearthstone[1], and all of our work on this topic is applicable across different card games assuming that specific heuristic modifications can be found for each. This work could also represent new areas of interest for our industrial sponsor, Stainless Games, as deck generation and prediction technology could potentially find a home in Stainless Games' Duels of the Planeswalkers series of games, providing crowd-sourced knowledge on deck construction. It is also important to recognise that these techniques could be applied by Stainless themselves (or the publisher, Wizards of the Coast,) to determine the popularity of specific cards, decks, or archetypes.

Further work related to this paper includes the recent winner of the best paper award at the IEEE Artificial Intelligence and Interactive Digital Entertainment 2016 conference, on which we are co-authors [52].

7.3 Future Work

7.3.1 Parallelization of MCTS

There are a number of interesting potential areas for future research in parallelization of MCTS. During the parallelization experimentation, the MCTS iterations were split evenly and statically assigned to the working threads. If a thread had finished early, it simply ended and did no further work. If iterations could be assigned dynamically as threads became available, then the process could be more efficient, and this is likely to have differing effects on each type of parallelization, and various approaches to dealing with collision between threads could be tried.

The branching factor of the game (or state) under examination may be relevant to the effectiveness of tree parallelization, as a higher branching factor should result in less thread

[1] `http://us.battle.net/hearthstone/en/`

waiting time. Experimenting with games or states with different branching factors would be interesting follow up work.

Future work on parallelization should also consider playing strength directly, in addition to the number of iterations performed in a given time.

7.3.2 Heuristic Pruning

It would be of interest to look at other methods of performing state extrapolation, more specifically other methods of searching the sub-tree that is traversed before the state is analysed. In other games where this sub-tree is not as simple, more advanced techniques may be appropriate to ensure reasonable decisions are being made.

We would expect heuristics which considered availability of squares, specifically those around the edges of the board, would be good candidates for creating a strong agent for Lords of War and similar games, and it would be of interest to explore such heuristics in future work. The possibility of evolving heat maps rather than designing them by hand would also be of interest [112].

It would be of interest to investigate the manner in which moves are selected by heuristics, particularly in multi-heuristic agents. Perhaps a move could be prioritised if it was selected by multiple heuristics, or perhaps moves that are only selected by a single heuristic could be soft-pruned until later stages of the search. Also, examining the total number of moves returned by multi-heuristic agents (and the difference from the maximum of $n \times HPL$) could be interesting.

The application of progressive techniques to heuristic agents in Lords of War would also be of interest, as it is entirely possible that the success of certain agents is being limited by regular exclusion of promising moves, which would be otherwise reintroduced at a later point in the search by a progressive technique.

7.3.3 Action Selection Mechanisms

For continuation of the Action Selection Mechanisms work, the most obvious extension is the creation of new online tuning methods, which would likely be interesting, as only one method of tuning was used during our work here. Also the method in which tuning was determined ($w_i/v_i > 0.5$) could be modified to create different conditions for tuning. A modified tuning method would allow for more careful and specific adjustment, and thus potentially create an agent that more readily adapted to opponent behaviour.

Further action selection mechanisms could also be created, particularly those that have multiple configuration parameters, which could then lead to an interesting multi-objective optimisation problem in combination with online tuning. An interesting suggestion for future work would appear to be modification of RobustRoulette$_n^k$, as the variations of this selection mechanism attempted here showed it to be promising. It is possible that the tuning of n and k values are specific to each game domain, and may require some fine tuning depending on the domain. Possible avenues forward included employing a mixed strategy across multiple action selection mechanisms, or the creation of a meta-heuristic which selects action selection mechanisms online.

7.3.4 Rule Association Mining

There are several other opportunities for future work which could be explored. For example, the technique used to separate rules in a_4 (Apriori with Influence Filtering) could also be applied to identity cards, using only rules generated for each identity to select either the entire deck, or the influence-spend portion of the deck. However this would require a large amount of data, as certain identities are unpopular and may appear only rarely within our current data set, so there would be fewer useful rules generated for these identities. It may also be worth looking at generation by *Faction*, which might yield more interesting results.

Also, as our observed cards were randomly selected, they may not adequately represent the real order cards are observed during a game (as it is more common to play certain cards earlier than others.) Biasing generation of the observed set of card may provide a more realistic scenario.

We can also look to applying these techniques to other domains, specifically games such as Hearthstone and Magic the Gathering. Magic the Gathering has a much larger set of active cards, and less stringent deck construction rules, so while this would represent a more challenging target, there is also a much larger amount of data available due to the larger player community and history of the game. Hearthstone likely represents a point of medium complexity, as the card pool is between the other two games mentioned here (approximately 450), and the deck construction rules are more restrictive than Magic: The Gathering, and thus provide more guidance.

A further avenue of research which could be pursue is that of pattern matching within the decks, in order to draw out common patterns which occur within multiple decks, and then using that information to further bias the prediction. We have already begun this work on Netrunner and Magic the Gathering, and have some interesting preliminary results which we expect to publish in the near future.

Appendices

Appendix A

Experimental Results

A.1 Chapter 4

nAgents	1	2	3	4	5	6	7	8	16
Root	505.74	253.47	191.71	154.77	135.94	113.21	104.41	91.74	82.14
Tree	514.09	299.13	207.51	163.61	133.19	109.64	105.74	98.05	84.11
Tree (VL)	511.58	297.60	198.75	152.99	131.45	110.64	105.86	98.42	92.73
Leaf	1226.18	676.75	615.35	618.23	630.62	619.71	626.06	634.27	627.83

Figure A.1: Results of four different parallelization techniques when applied to UCT. Results are expressed in milliseconds taken to complete a single decision.

nAgents	1	2	3	4	5	6	7	8	16
Root	1061.07	533.23	376.74	341.36	273.27	259.20	220.66	194.92	187.23
Tree	1057.69	768.74	528.83	432.33	348.83	291.88	282.98	259.58	248.13
Tree (VL)	1056.35	751.91	531.36	444.21	344.75	295.64	283.22	264.05	246.13
Leaf	1885.77	1087.50	981.75	1019.97	975.29	987.03	1049.48	1005.02	970.34

Figure A.2: Results of four different parallelization techniques when applied to ISMCTS. Results are expressed in milliseconds taken to complete a single decision.

nAgents	1	2	3	4	5	6	7	8	16
Root	50.2%	54.6%	56.2%	57.8%	59.2%	60.8%	61.2%	62.4%	65.3%
Tree	49.1%	58.2%	62.8%	65.3%	66.8%	68.0%	69.2%	71.9%	72.1%
Tree (VL)	48.8%	57.6%	61.7%	66.0%	64.3%	67.5%	70.7%	72.4%	73.2%
Leaf	47.2%	49.7%	50.6%	51.7%	50.9%	52.1%	52.9%	53.4%	52.8%

Figure A.3: Win Percentage when applying four different parallelization techniques to a UCT agent. The opponent in each case was an unparallelized UCT agent with the same iteration budget.

nAgents	1	2	3	4	5	6	7	8	16
Root	50.3%	52.9%	55.4%	56.4%	60.1%	61.0%	62.7%	63.7%	66.2%
Tree	47.1%	58.4%	61.9%	64.7%	66.2%	66.3%	67.9%	70.6%	71.8%
Tree (VL)	46.8%	56.2%	60.4%	63.9%	65.7%	66.7%	68.3%	72.1%	74.3%
Leaf	45.6%	48.2%	49.7%	51.3%	52.4%	52.7%	52.4%	53.1%	52.9%

Figure A.4: Win Percentage when applying four different parallelization techniques to a ISMCTS agent. The opponent in each case was an unparallelized ISMCTS agent with the same iteration budget.

A.2 Chapter 5

Table A.1: Number of wins (out of 1000) of single heuristic agents h_1 - h_{11} playing against a standard UCT agent with 10000 iterations at varying Hard Pruning Limits (y-axis)

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}
1	60	0	24	116	62	0	0	0	0	0	0
2	220	0	78	248	158	8	22	6	12	32	28
5	580	0	188	608	500	38	38	40	34	0	0
10	776	0	356	826	776	138	144	170	170	68	112
15	868	2	478	900	814	308	258	296	288	142	166
20	854	0	476	886	836	256	318	286	336	290	334
25	800	0	586	832	836	396	410	354	342	442	278
30	794	8	560	790	828	380	388	420	404	500	416

Table A.2: Number of wins (out of 1000) of multi heuristic agents h_4h_1 - h_4h_{11} playing against two different opponents.

	h_4h_1	h_4h_2	h_4h_3	h_4h_5	h_4h_6	h_4h_7	h_4h_8	h_4h_9	h_4h_{10}	h_4h_{11}
vs h4	559	117	288	644	526	534	455	475	536	441
vs Plain UCT	863	546	747	875	842	801	897	882	855	892

Table A.3: Number of wins (out of 1000) of multi heuristic rollforward agents h_4h_1R - $h_4h_{11}R$ playing against two different opponents.

	h_1R	h_2R	h_3R	h_4R	h_5R	h_6R	h_7R	h_8R	h_9R	$h_{10}R$	$h_{11}R$	h_4h_5R
vs H4	514	0	111	835	0	67	42	54	76	35	66	524
vs Plain UCT	831	3	493	920	79	337	291	284	303	281	274	842

Table A.4: Number of wins (out of 1000) of multi heuristic rollforward agents h_4h_1R - $h_4h_{11}R$ and multi-heuristic non-rollforward agents.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}
Rollforward	831	3	493	920	837	337	291	284	303	281	274
No Rollforward	868	2	478	900	814	308	258	296	288	142	166

Appendix B

Generated Netrunner Decks

Below are a list of example decks generated by providing an Identity card, and 3 other cards that are commonly used in the faction of the identity.

Listing B.1: Noise: Hacker Extrordinaire

```
Noise: Hacker Extraordinaire
2 x Aesop's Pawnshop ●●
3 x Cache ●
1 x Crypsis
3 x Cyberfeeder
3 x Daily Casts
2 x Darwin
2 x Datasucker
3 x Deja Vu
2 x Djinn
2 x Grimoire
1 x Hades Shard ●
2 x Hivemind
2 x Imp
2 x Incubator
3 x I've Had Worse
2 x Medium
2 x Parasite
2 x Progenitor
3 x Sure Gamble
3 x Virus Breeding Ground
```

```
Cards: 45, Influence Spent: 8/15
```

Listing B.2: The Foundry: Refining the Process

```
The Foundry: Refining the Process
3 x Accelerated Beta Test
2 x Adonis Campaign
2 x Architect
2 x Eli 1.0
3 x Galahad ●
2 x Gila Hands Arcology
3 x Hedge Fund
3 x Jackson Howard ●
3 x Lancelot ●
3 x Merlin ●
3 x NAPD Contract
3 x NEXT Bronze
3 x NEXT Silver
3 x Project Vitruvius
2 x Self-destruct
3 x The Twins ●
2 x Viktor 2.0

Cards: 45, Influence Spent: 15/15, Agenda Points: 20/[20-21]
```

Listing B.3: Quetzal: Free Spirit

```
Quetzal: Free Spirit
2 x Account Siphon ●●●●
2 x D4v1d
1 x Daily Casts
2 x Datasucker
3 x Day Job
3 x Deja Vu
3 x Dirty Laundry
2 x e3 Feedback Implants ●●
1 x Femme Fatale ●
1 x Hades Shard ●
3 x Inject
3 x I've Had Worse
2 x Kati Jones
2 x Knifed
2 x Medium
1 x Mimic
2 x Parasite
```

```
3 x Prepaid VoicePAD
2 x Same Old Thing
2 x Scrubbed
3 x Sure Gamble

Cards: 45, Influence Spent: 14/15
```

Listing B.4: Leela Patel: Trained Pragmatist

```
Leela Patel: Trained Pragmatist
2 x Account Siphon
2 x Cerberus "Rex" H2
2 x Corroder ●●
2 x Daily Casts
3 x Dirty Laundry
2 x Emergency Shutdown
2 x Faerie
1 x Femme Fatale
2 x Inside Job
2 x Kati Jones
2 x Legwork
3 x Logos
3 x Lucky Find ●●
1 x Mimic ●
1 x Passport
2 x Plascrete Carapace
2 x R&D Interface ●●
2 x Same Old Thing
2 x Security Testing
1 x Sneakdoor Beta
3 x Special Order
3 x Sure Gamble

Cards: 45{45}, Influence Spent: 15{15}/15
```

Listing B.5: Blue Sun: Powering the Future

```
Blue Sun: Powering the Future
2 {1} x Adonis Campaign ●●
2 {0} x Archer ●●
2 {2} x Caduceus
1 {0} x Crisium Grid
2 {2} x Curtain Wall
2 {0} x Enigma
2 {0} x Hadrian's Wall
```

3 {3} x Hedge Fund
2 {3} x Hive
3 {0} x Hostile Takeover
3 {2} x Ice Wall
3 {3} x Jackson Howard ●
1 {3} x NAPD Contract
1 {0} x Orion
3 {3} x Oversight AI
3 {1} x Priority Requisition
3 {3} x Project Atlas
3 {3} x Restructure
3 {3} x Scorched Earth
2 {1} x SEA Source ●●
1 {1} x Taurus
2 {2} x Tollbooth ●●

Cards: 45, Influence Spent: 15/15, Agenda Points: 20/[20-21]

Appendix C

Lord of War Rules

Below are the Lords of War Rules, taken from the website of the publisher, Black Box Games [1].

[1] https://boardgamegeek.com/boardgamepublisher/24619/black-box-games-publishing

Lords of War

Lords of War is a game of skill, daring and chance – a game full of tactics, betrayal, action and cunning. Plus, it's set in its own fictional universe, with stories and lore expanded upon at lords-of-war.com.

Lords of War is more than just a game. It is a community, a society, a club. More than that, it is a family. And so, welcome to the Lords of War family – it's ugly, bloody and brutal, and we think you're going to fit in just fine.

To find out why these armies are fighting, and to discover the history behind the various Command Cards and Generals in your decks, please head over to the website where you will also find video guides, tutorials and much, much more.

Otherwise, best of luck with your battles – and remember, fortune favours the bold!

Martin Vaux & Nick Street
Black Box Games

Contents: **Two core army decks of 36 cards each and a Battle Mat.**

Game time: **30 – 45 minutes**

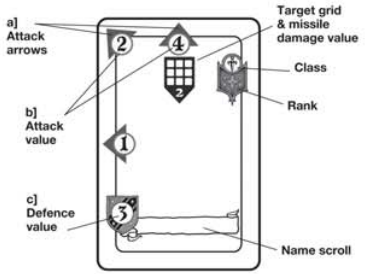
1) The cards:

Each card, also termed as a "unit" in these rules, has a number of features as shown in the card diagram below, with the 3 main features being:

a] Attack arrows - The direction/s the unit can attack in when it is in play. The unit attacks simultaneously in all the directions its attack arrows point.

b] Attack value - The strength of the attack in a given direction.

c] Defence value - The strength of attack/s the unit can withstand without being eliminated.



Class and Rank:

Each unit has a symbol to denote its class & rank:

CLASS:

Infantry, Cavalry, Spear, Ranged, Berserker

RANK:

Recruit, Regular, Veteran, Elite, Special, Command, General

Most units can be placed directly into **combat** on the battlefield. A unit that can engage in combat has one or more attack arrows around its border. The more attack arrows around a card's border, the more flexible the unit is on where it can be placed and the directions it can attack at the same time. The higher the numbers on the card, generally the better the unit is in combat.

A ranged weapon unit with the rank symbol has a target grid and missile damage value on its card as shown:

This type of unit can shoot at enemy units from a distance and are usually best played behind friendly units in order to keep them out of combat and allow effective use of their ranged weapon.

All spear units and ranged weapon units are **support units.** A support unit *does not* have to be placed directly into combat, but can be placed adjacent to a friendly unit. Support units can be used to occupy a space on the Battle Mat to prevent an enemy unit from being placed there or placed next to a friendly card to prevent an opponent attacking from a certain direction.

The six **Command Cards** represent the leaders of the army and are represented by the rank symbol on their card.

Loss of four of these command units will result in the collapse of the entire army and the immediate loss of the battle. One of the command cards is the army general which is represented by the rank symbol.

2) Set-up: The Core game

1) Decide which army each person will play.

2) Determine who is player 1 and player 2 by tossing a coin etc.

3) Place the Battle Mat face-up with one of the shorter sides in front of each player, so that the number **1** on the Battle Mat is facing and readable by player 1 & the number **2** is facing and readable by player 2.

4) Each player takes their General card (with the rank symbol on) from their army deck and places it within easy reach.

5) Each player shuffles their army deck with the cards face down and deals the top five cards face down in front of themselves. Each player then places the remainder of the deck face down within easy reach.

6) Each player takes the five dealt face down cards and their General card to form their "hand" of six cards. Each player then holds their six cards so only they can see them and should not show them to their opponent.

Winning the game:

The first player to eliminate **any** 20 of the opposition's cards **or** 4 of the opposition's Command Cards is the winner.

3) Issuing the challenge:

1) Player 1 chooses one card from their hand of 6 and places it face up onto the square on the Battle Mat with the number 1 on it. The unit is laid so that it is facing and readable by Player 1, not the opposing player. Player 1 then takes the top card from his army deck (not showing it to his opponent) to refresh his hand back up to six cards.

2) Player 2 then chooses a card from his hand of 6 and places it face up onto the square on the Battle Mat with the number 2 on it. Player 2 then takes the top card from his army deck (not showing it to his opponent) to refresh his hand back up to six cards.

The battle is ready to begin...

Each player, starting with Player 1, now takes it in turns to follow the 3 phase turn sequence:

Phase 1 - Deployment: Play a unit from your hand onto the battlefield according to the "Engagement Rule" & its exceptions.

Phase 2 - Elimination: Determine whether any unit has been eliminated by combat or ranged attack or a combination of both.

Phase 3 - Reinforcement: Increase the number of cards in hand back up to six.

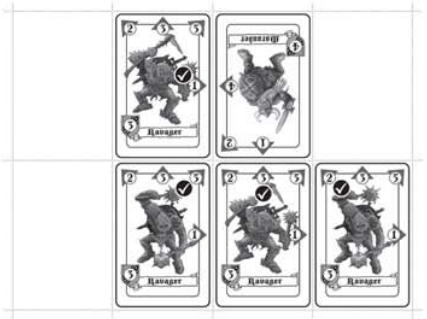
4) Phase 1: Deployment

The player chooses one card from their hand of six and lays it face up onto any **empty square** on the Battle Mat. The card is laid so that it is facing and readable by the placing player, not the opposing player.

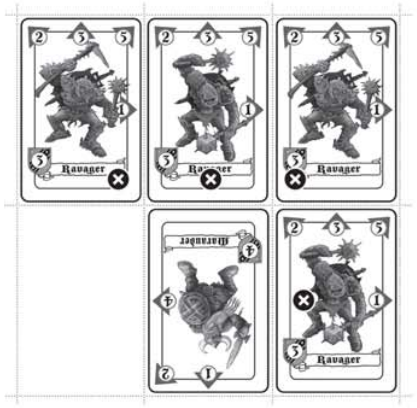
The unit *must* be placed adjacent to any of the opponent's units on the Battle Mat, so that at least one of its attack arrows is pointing at a corner or edge of at least one of the opposing units on the Battle Mat. This is known as the ***"engagement rule".***

Note: The card you are placing to engage an enemy unit does not have to be adjacent or next to any of your friendly units already on the Battle Mat.

Examples of deployment: The Dwarf player has a Marauder on the Battle Mat and it is the Orc player's turn. The Orc player chooses to play his Ravager. It must be played according to the "engagement rule", so can be placed in any of the following positions:



As indicated below, the Orc Ravager **cannot** be placed behind the Dwarf Marauder or to the right hand side of it since it will not be following the "engagement" rule as none of its attack arrows would be pointing at a corner or edge of the opposing Dwarf Marauder card.



There are two exceptions to the ***engagement rule:***

a) The deployment of support units. A support unit (a spear or ranged weapon unit) may be placed so that it engages in the usual way or ***instead*** it can be placed adjacent to any friendly unit on the Battle Mat. When placed adjacent to a friendly unit the card doesn't have to have any of its attack arrows pointing at the friendly unit or have any of the adjacent friendly unit's attack arrows pointing at it.

Note: A support unit gives no + attack or defence bonus of any kind to the friendly unit it is placed adjacent to.

b) If for any reason there are no enemy units on the Battle Mat at the beginning of a player's turn, the player can deploy ANY unit from his hand adjacent to any other friendly unit on the battlefield.

If there are no cards on the Battle Mat at the start of a player's turn, the player can place a card from their hand onto any square on the Battle Mat.

Note: If you cannot lay a card on your turn in accordance to the engagement rule and its exceptions, you must place a card of your choosing from your hand adjacent to any enemy card on the Battle Mat and then show your hand to your opponent (to prove you cannot go).

5) Phase 2: Elimination

Once the player places a card onto the Battle Mat, they check to determine whether one or more units have been eliminated by combat or ranged attack or a combination of both.

Combat is a simultaneous process and **ALL** units on the Battle Mat attack in **ALL** directions of their attack arrows in the same turn. This means no matter whose turn it is all units on the Battle Mat of both players are fighting in combat, which may result in units from either side being eliminated. Therefore, when a new unit is placed onto the battlefield, check the attack and defence values of all the units on the Battle Mat, including ranged weapon attacks, to determine whether a unit has been eliminated.

A unit is eliminated when the total combined attack value of one or more of the opponent's cards *engaging* and /or "range attacking" the unit, is *greater* than the unit's defence value.

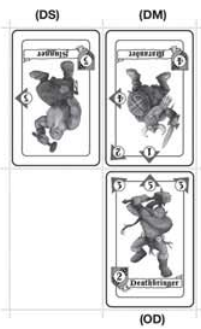
If the (total combined) attack value is equal to or less than the opponent's defence value, then there is no effect and the defending card stays in play on the Battle Mat.

Eliminated units are not removed until the results for all the possible engagements have been checked. An enemy's eliminated units are kept face up in a discard pile to one side of the Battle Mat by the opposing player to keep track of the number of eliminated units needed to win.

Note: Friendly units **never** cause damage to each other, even if engaging each other.

A one-on-one combat example:

The Orc player has just placed the Orc Deathbringer (OD) to engage the Dwarf Marauder (DM) and Slugger (DS).



1) The Orc Deathbringer has eliminated the Dwarf Marauder because it is engaging it for an attack value of 5, which is greater than the Marauder's defence value of 4.

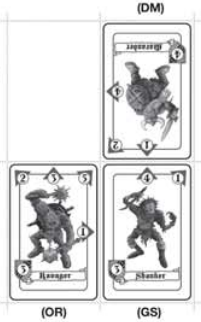
2) The Orc Deathbringer has NOT eliminated the Dwarf Slugger because it is engaging it for an attack value of 3, which is NOT greater than the Slugger's defence value of 3.

3) The Orc Deathbringer, with a defence value of 2, has NOT been eliminated by the Dwarf Marauder who is only engaging it for an attack value of 1.

4) The Dwarf Marauder is removed from the Battle Mat and put in the opposing player's discard pile.

A two against one combat example:

The Orc player has just placed the Orc Ravager (OR) to engage the Dwarf Marauder (DM).



1) The Orc Ravager and Goblin Shanker (GS) have jointly eliminated the Dwarf Marauder. The Goblin Shanker is engaging it for an attack value of 4 and the Orc Ravager is engaging it for an attack value of 5. This combined attack of 9 is greater than the Dwarf Marauder's defence value of 4.

2) The Orc Ravager, with a defence value of 3, has NOT been eliminated by the Dwarf Marauder who is only engaging it for an attack value of 2.

3) The Goblin Shanker, with a defence value of 3, has NOT been eliminated by the Dwarf Marauder who is only engaging it for an attack value of 1.

4) The Dwarf Marauder is removed from the Battle Mat and put in the opposing player's discard pile.

Elimination situations:

Sometimes placing a unit onto the Battle Mat may cause one or more of the following situations, which are all legal moves:

i) the elimination of more than one enemy unit simultaneously, either by itself or in combination with friendly units already on the Battle Mat.

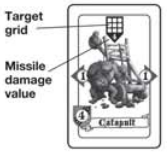
ii) the placed unit and any number of enemy units being eliminated at the same time; this is known as a suicide move.

iii) the placed unit being instantly destroyed and no opposing units being eliminated.

A player that eliminates the enemy's General immediately takes an extra turn, after having refreshed their hand back up to six cards. Killing an enemy General is the only time when the normal turn sequence is disrupted. If both army generals are eliminated simultaneously, neither player gets the option of an extra turn.

6) Elimination involving ranged units

All ranged weapon units, such as The Orc Catapult (shown) have a target grid with a missile damage value.



The target grid on the card shows which squares on the Battle Mat the ranged unit can target when deployed. The Orc catapult, (card **C**), according to its target grid, can range attack any **one** of the 9 squares to its **front**, as represented by the grey squares shown in Figure 1

The missile value is the damage caused to a chosen single enemy unit every time the ranged unit shoots. The Orc catapult displayed above causes 4 points of ranged attack missile damage.

This Dwarf Axethrower is a ranged unit that range attacks to its rear. The Axethrower, (card **A**), according to its target grid, can range attack any **one** of the 6 squares to its **rear** when on the Battle Mat, as represented by the grey squares shown in Figure 2. The Axethrower causes 3 points of ranged attack missile damage.

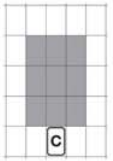


Figure 1

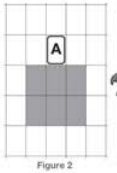
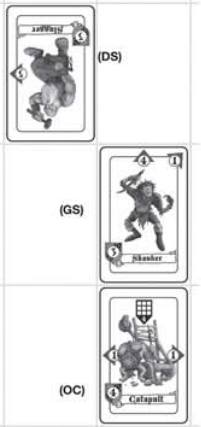


Figure 2

A ranged attack combat example:

The Orc player has just placed the Orc catapult (OC) (a support unit) adjacent to (behind) his Goblin Shanker (GS).



1) The Goblin Shanker is not engaging the Dwarf Slugger (DS), so is not attacking it.

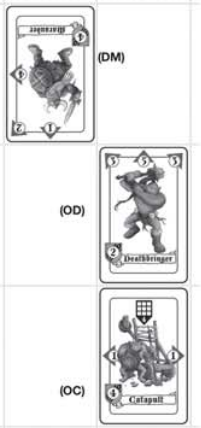
2) The Dwarf Slugger is not engaging the Goblin Shanker, so is not attacking it.

3) The Orc Catapult has eliminated the Dwarf Slugger because it is shooting it for a missile damage value of 4, which is greater than the Slugger's defence value of 3.

4) The Dwarf Slugger is removed from the Battle Mat and put in the opposing player's discard pile.

A combined combat involving a ranged attack example:

The Orc player has just placed the Orc catapult (OC) (a support unit) adjacent to his Orc Deathbringer (OD).



1) The Orc Deathbringer and Orc Catapult have jointly eliminated the Dwarf Marauder (DM). The Orc Deathbringer is engaging it for an attack value of 3 and the Orc Catapult is shooting the Dwarf Marauder for a missile damage value of 4. This combined attack of 7 is greater than the Dwarf Marauder's defence value of 4.

2) The Dwarf Marauder is not engaging the Orc Deathbringer, so is not attacking it.

3) The Dwarf Marauder is removed from the Battle Mat and put in the opposing player's discard pile.

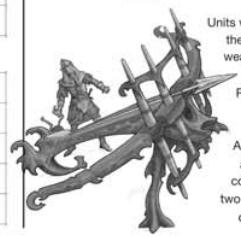
Ranged weapon situations:

Units with ranged weapons can make a ranged attack in the same turn they are placed on the Battle Mat **and**, once on the Battle Mat, they can make a ranged attack **every time** it is their controlling player's turn.

Ranged attack units can shoot over any units and do not need a clear line of sight to the enemy unit they are shooting at.

Ranged units **cannot** shoot if "engaged" by an enemy unit, e.g. when an adjacent enemy unit has one or more of its attack arrows pointing at it.

A ranged unit **can** shoot if it has one or more of its attack arrows pointing at adjacent enemy unit/s (engaging them), but the adjacent enemy units have no attack arrows engaging back.



Units with ranged weapons which find that they have the opportunity to both fire their weapon and fight in combat must choose to do only one during the turn.

Ranged weapon units are never permitted to target friendly units.

A ranged weapon unit can only ever shoot at a **single** enemy unit (chosen by the controlling player) within its target area, but two or more friendly ranged weapon units can shoot at a single enemy unit in a turn.

7) Phase 3: Reinforcement

After deploying his chosen unit onto the Battle Mat and determining whether any units have been eliminated or not, the player completes his turn by refreshing his hand back up to six cards.

In order to refresh his hand back up to six cards, a player can choose to either take the top card from his face down army deck and place it into his hand (so only he can see it) or "recall" a unit from the Battle Mat.

To be able to "recall" a unit back from the Battle Mat and into the hand, the unit:

1) must not be engaged by an enemy card (does not have an enemy card adjacent to it that has at least one of its attack arrows directly pointing towards it).

2) must not have been placed down that turn.

3) must not have just taken part in a combat that has eliminated an enemy unit (this includes ranged weapon units as well).

If a unit is not subject to any of the three conditions, it can be withdrawn from the battlefield and placed into the player's hand to refresh the hand back up to six cards.

8) To the bitter end:

After player 1 has completed his 3 phase turn, player 2 then takes his turn following the three phase turn sequence:

Phase 1 - Deployment: play a unit from your hand onto the battlefield according to the "engagement rule" & its exceptions.

Phase 2 - Elimination: determine whether any unit has been eliminated by combat or ranged attack or a combination of both.

Phase 3 - Reinforcement: increase the number of cards in hand back up to six.

Players continue to take turns following the above three phase turn sequence until one player wins the game.

For a full video tutorial of the above rules visit www.lords-of-war.com

TACTICAL TIPS

Deploy ranged units so they are safely shielded from combat behind other friendly units and take advantage of the reach of their weapons.

Do not just play weaker units for the sake of it; they will be eliminated very easily. Deploy a weaker unit to "engage" an opposing stronger unit in one of its vulnerable positions where it does not have an attack arrow. This will stop it from being recalled and make it more vulnerable to elimination.

A weaker unit can also be used to deliver the final attack to eliminate an opposing stronger unit. Even if used in a suicide move; the loss of a weaker unit for the elimination of a stronger unit, such as a command unit, is worth it.

WHAT NEXT?

A) Battles with no boundaries:

Play the game exactly the same way, but ignore the boundaries of the Battle Mat or play the game without the Battle Mat at all. Playing this way will present new opportunities and challenges to an experienced player.

B) Game length:

To shorten or lengthen the game time both players agree to change the number of units and / or command units that have to be eliminated to win.

C) Personalise your army:

Every core army deck has a number of units of each rank:

6 Command units | 4 Special units | 4 Elite units

6 Veteran units | 8 Regular units | 8 Recruit units

To personalise your army you can take any card out of your army deck and replace it with any card of a similar rank from another deck.

Example: as an Orc player I would like some more ranged firepower, so I take out two Orc Ravagers (of veteran rank) from my army deck and replace them with two Orc catapults (of veteran rank).

Personalising your army by rank swapping in this way can be done with as many cards as you wish.

Designing your own army in this way provides the chance to try a wide range of different combinations to play against your opponents, as well as creating an army that appeals to your own personal style of play.

D) Building a Mercenary army:

Not happy just playing one race? Using the same rank swapping system, personalise your army even further by taking any card out of your army deck and replacing it with any card of a similar rank from any other army deck.

Example: as an Orc player I would like some more archers with a better ranged attack than my Goblin Bow Pests, so I take out two Orc War hogs (of Elite rank) from my army deck and swap them with two Lizardman Quillshots (of Elite rank).

Additional guidance on army building is available on the website.

Lords of Game Concept & Design: **Nick Street & Martin Vaux**
Lord of Testing: **Alan Williams**
Lord of Art & Graphic Design: **Steve Cox**

Thanks to: Dnyan Keni, Carl Smith, Niyati Keni, Rachel Vaux, Luke Maddox, Aidan Taylor, Brenda Alvares, Tom Bell and Sarah O'Regan.

Printed in England by Richard Edwards

Appendix D

NetRunner Rules

Below are the NetRunner Core Rules, taken from the website of the publisher, Fantasy Flight Games [1].

[1] https://images-cdn.fantasyflightgames.com/ffg_content/android-netrunner/support/android-netrunner-core-rules.pdf

Android: Netrunner

The Card Game

Rules of Play

ANDROID™

NETRUNNER™

THE CARD GAME

Over the course of 18 hours, the runners hit Jinteki, Haas-Bioroid, and Weyland Consortium with DOS attacks, datatheft, and a truly vulgar piece of cyber-vandalism. These attacks cost each megacorp millions upon millions of credits. NBN put together a holo-report inside half an hour. Thirty minutes after the third megacorp node went dark, Lily Lockwell was standing in front of the Beanstalk gravely lecturing on the evils of unregulated networks and the rise of cybercrime worldwide. Five minutes later, the runners had struck again; now Lockwell was reading out the Anarch's Manifesto. They hadn't bothered to make her lips synch with the new audio track. One in three feeds got a special bonus: Lockwell's head grafted onto a sense-star's scantily-clad body.

The talking heads said it was a legion of organized cybercriminals, Tri-Maf activity, Martian terrorists. They were wrong. It was three people–a g-mod from Heinlein, a cyborg New Angelino, and a baseline woman from BosWash–who knew one another by reputation only. But the heads were right about one thing: it was the start of a cyber war, one that neither side could afford to lose.

The Living Card Game

Android: Netrunner is a two-player game that can be played using only the contents of this box, known as the core set. However, *Android: Netrunner* is also a Living Card Game (LCG®) that evolves over time with regularly released expansions. Each expansion offers players many additional cards that add variety, new customization options, and rich themes to the game. Unlike most collectible card games, all LCG expansions have a fixed distribution–there is no randomization to their contents.

Introduction

Welcome to *Android: Netrunner*. It is the future. Humanity has spread itself across the solar system with varying degrees of success. The Moon and Mars are colonized. A plan to terraform the Red Planet is well underway, hindered only by a civil war that has broken out and locked down many of its habitation domes. On Earth, a massive space elevator has been built near the equator in the sprawling megapolis of New Angeles, stretching up into low orbit. It is the hub of trade in the solar system, and most people refer to it as the "Beanstalk."

Computers have continued to advance along with discoveries in the field of neurobiology. This has led to brain-mapping, a method by which a human mind can be stored electronically in sophisticated mind-machine interface devices. The physical mouse and keyboard are archaic relics; gestural interfaces and virt displays are commonplace. Elite users "jack in," plugging the computer directly into their brains.

Enormous megacorporations, called corps by most, influence every facet of daily life: food, threedee, music, career choices. Jinteki and Haas-Bioroid redefine life itself, making clones and bioroids with braintaped, artificially-intelligent minds. The Weyland Consortium owns a piece of everything that goes up or down the Beanstalk, and *everything* goes up or down the Beanstalk. And NBN shapes what you think and dream, with the most extensive media network ever conceived on Earth under their control.

Everyone relies on the network, the all-seeing, all-hearing grid that surrounds Earth and reaches out into the solar system beyond. More data flows through the network every second than was ever expressed in the first five thousand years of written language. It is a surveillance network, a financial system, a library–it is the backbone of modern civilization. And it is also the only weakness the corps have.

The network is forever evolving and moving, impossible to completely pinpoint or lock down. Rogue operators–computer specialists with the hardware, software, and raw talent to challenge the system–use the sprawl of the net to their advantage. Some want to expose the rot that lies at the heart of the system, and to awaken the teeming billions to the hypocrisy of their corporate masters. Others just want to earn a profit, or express themselves in the ultimate medium. Whatever their motivation, the actions of these individuals intersect in a common cause: that of digital independence. They are runners.

Game Overview

Android: Netrunner is a card game for two players set in the dystopian future of the Android universe. One player assumes the role of a **Runner**, a rogue hacker armed with bleeding-edge gear and software, while the other player controls a powerful **Corporation** that will stop at nothing to achieve its goals.

In *Android: Netrunner*, players alternate taking turns, beginning with the Corporation. During the Corporation's turn, he has three **clicks** to spend. The Corporation can spend his clicks to perform a variety of actions, including gaining credits, drawing cards, installing cards, and advancing agendas. The Corporation must carefully divide his efforts between defensive actions, such as protecting his servers from the Runner, and offensive actions, such as tracing the Runner or advancing agendas.

The Runner has four clicks to spend during his turn. The Runner can also spend his clicks to perform a variety of actions, including gaining credits, drawing cards, installing cards, and making runs. During a **run**, the Runner attempts to hack into the Corporation's servers in an effort to hinder the Corporation and steal his agendas. The Runner has several different targets to choose from when initiating a run; choosing where and when to run is a key part of an effective Runner strategy.

Object of the Game

The objective for both players is to score seven agenda points. The Corporation scores agenda points by advancing agendas; the Runner scores agenda points by stealing agendas from the Corporation. Agendas are cards that only appear in the Corporation's deck.

The Corporation also wins if the Runner is **flatlined** (see "Damage" on page 20) and the Runner wins if the Corporation must draw a card from his empty draw deck.

An agenda card worth 2 agenda points.

Corporate Factions

In *Android: Netrunner* there are four different Corporate factions to choose from. Corporate factions are important for deckbuilding (see "Deckbuilding" on page 24) and each Corporate faction has certain cards affiliated with it. These factions are:

- Haas-Bioroid
- Jinteki
- NBN
- Weyland Consortium

Runner Factions

In *Android: Netrunner* there are three different Runner factions to choose from. Factions are important for deckbuilding and each Runner faction has certain cards affiliated with it. These factions are:

- Anarch
- Criminal
- Shaper

Neutral Cards

Some Corporation and Runner cards have no faction affiliation. These cards are called **neutral cards** and can be used in any deck of the corresponding side.

Component Overview

The *Android: Netrunner* core set includes the following components:

Corporation Cards (134)

- 28 Haas-Bioroid Cards
- 28 Jinteki Cards
- 28 Weyland Consortium Cards
- 28 NBN Cards
- 22 Neutral Corporation Cards

Runner Cards (114)

- 33 Anarch Cards
- 33 Criminal Cards
- 33 Shaper Cards
- 15 Neutral Runner Cards

Reference Cards (2)

These cards show the actions a player can perform during his turn.

Click Tracker Tokens (2) & Cards (2)

Together these are used to track how many clicks a player has left to spend during his turn. The reference card with four spaces is the Runner's. The reference card with three is the Corporation's.

One-Credit \\ Advancement Token (51)

One side of this token represents one credit. Credits are the basic currency of *Android: Netrunner*.

One credit

The other side of this token is an advancement token. The Corporation uses advancement tokens to track the advancement of his installed cards.

Advancement

Five-Credit Token (8)

This token represents five credits.

Brain Damage Token (6)

This token represents one brain damage. The Runner can get brain damage through various card effects.

Bad Publicity \\ Tag Token (12)

One side of this token represents one point of bad publicity. The Corporation can get bad publicity through various card effects.

Bad publicity

The other side of this token represents one tag. The Runner can get tags through various card effects.

Tag

Generic Tokens (23)

One side of this token is purple, and the other side is red. Players use these tokens to track counters on cards as necessary. The most common counters are agenda counters, power counters, and virus counters.

Setup

To prepare a game of *Android: Netrunner*, carefully follow the steps below.

1. **Choose Sides:** The players decide who will play as the Runner and who will play as the Corporation. Then, each player places his identity card faceup in his play area and takes a corresponding deck.

   Note: New players should use the Shaper and Jinteki starter decks for their first game.

2. **Create Token Bank:** Gather the credits, advancement, brain damage, tag, bad publicity, and generic tokens into piles. Keep these piles within reach of both players.

3. **Collect Starting Credits:** Each player takes five credits from the bank.

4. **Shuffle Decks:** Each player shuffles his deck. After shuffling, each player offers his deck to his opponent for further shuffling.

5. **Draw Starting Hands:** Each player draws five cards from the top of his deck to form his starting hand. After drawing starting hands, the Corporation may choose to take a **MULLIGAN** by shuffling his hand back into his deck and then drawing a new starting hand. After the Corporation decides whether to mulligan, the Runner decides whether to mulligan as well. If a player takes a mulligan, he must keep his second hand as his starting hand. When the players are satisfied with their starting hands, each player places his deck facedown in his play area.

The Golden Rule

If the text of a card directly conflicts with the rules in this book, the card text takes precedence.

Starter Decks

The game can be enjoyed straight out of the box by building starter decks to play with.

To make a starter deck, take all the cards of a single Corporate or Runner faction and shuffle in all of the neutral cards for the chosen side. Starter decks are quick to build and are legal for tournament play.

Below are the card numbers for the Corporate factions, Runner factions, and Neutral cards that appear in the core set:

Corporation:

Haas-Bioroid Cards #54-66

Jinteki Cards #67-79

NBN Cards #80-92

Weyland Cards #93-105

Neutral Cards #106-113

Runner:

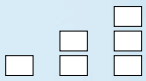
Anarch Cards #1-16

Criminal Cards #17-32

Shaper Cards #33-48

Neutral Cards #49-53

This symbol identifies cards included in the core set. Every card in the core set has this symbol next to its card number.



These boxes represent the quantity of a card in the core set and appear to the left of the core set symbol.

Important Vocabulary

Players should become familiar with the following terms before reading the rest of the rules. Refer to the "Glossary" on page 30 to look up other terms as needed.

ACTIVE: An active card's abilities affect the game and can be triggered.

INACTIVE: An inactive card's abilities do not affect the game and cannot be triggered.

INSTALL: This is the game term for playing a card onto the table.

CREDIT: This is the basic unit of wealth, represented by .

CLICK: This is the basic unit of work, represented by .

REZ: This is the act of flipping a facedown card faceup. The Corporation installs his cards facedown and must rez them in order to use them.

Play Areas

In *Android: Netrunner*, the play areas for the Corporation and the Runner differ significantly from one another. However, both players have a credit pool, identity card, score area, and click tracker.

Credit Pool
Each player has a credit pool where he keeps the credit tokens he has available to spend. Spent credits are returned to the token bank.

Identity Card
Each player has an identity card that is placed faceup in his play area. The identity card does not count toward his maximum hand or deck size, and is always active during the game.

Score Area
Each player has a score area that holds his scored or stolen agendas. Agendas in a score area add their agenda points to a player's score.

Click Tracker
Each player has a click tracker that he uses to track the number of clicks left that he has to spend on his turn. This is a game aid only and its use is optional.

Corporation Play Area

In addition to his credit pool, identity card, score area, and click tracker, the Corporation's play area includes his servers and his ice. There are two types of servers: **central servers** and **remote servers**.

Central Servers
The Corporation has three central servers: **Headquarters**, **Research and Development**, and **Archives**. Each central server also has a **root**.

Headquarters (HQ)– This is the Corporation's hand of cards. Cards in HQ are inactive. The Corporation begins the game with a maximum hand size of five cards. The Corporation identity card represents HQ for the purposes of card installation.

Research and Development (R&D)–
This is the Corporation's draw deck. R&D is kept facedown within reach of the Corporation. Cards in R&D are inactive.

Archives– This is the Corporation's trash pile. Archives is kept adjacent to R&D. This is where Corporation cards are placed when they are **trashed** or **discarded**. Cards in Archives are inactive.

Some cards enter Archives faceup, and some cards enter Archives facedown. Facedown cards in Archives should be oriented horizontally so that the Runner can easily see them. Both the Corporation and Runner may look through the faceup cards stored in Archives at any time, and do not need to maintain the order of its cards while doing so. The Corporation can also look at the facedown cards in Archives at any time; the Runner cannot.

Root– This is the area of a central server where **upgrades** for the server are installed. When an upgrade is installed in the root, it should be placed below the server. If a root has no cards installed in it, it is considered to be empty.

Remote Servers
The Corporation has no remote servers at the beginning of the game. The Corporation creates remote servers by installing cards. Cards in remote servers are active if rezzed and inactive if unrezzed.

There is no limit to the number of remote servers the Corporation can have at any given time.

Ice
The Corporation installs ice to protect his servers. Installed ice is always dedicated to a particular server and placed in front of that server. Ice can protect an empty server. Ice is active if rezzed and inactive if unrezzed.

Runner Play Area

In addition to his credit pool, identity card, score area, and click tracker, the Runner's play area includes his **grip**, his **stack**, his **heap**, and his **rig**.

Grip
This is the Runner's hand of cards. The Runner begins the game with a maximum hand size of five cards. Cards in the grip are inactive.

Stack
This is the Runner's draw deck. The stack is kept facedown within reach of the Runner. Cards in the stack are inactive.

Heap
This is the Runner's trash pile. The heap is kept adjacent to the Runner's identity card. This is where Runner cards are placed when they are trashed or discarded. Cards in the heap are faceup and inactive. Both the Runner and Corporation may look through the heap at any time, but must maintain the order of its cards.

Rig
This is where the Runner installs his cards. The rig is separated into three rows: one for programs, one for hardware, and one for resources. Cards in the rig are active.

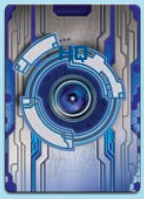
CORPORATION PLAY AREA

SCORE AREA

CLICK TRACKER

HQ

ROOT



IDENTITY CARD

CREDIT POOL

ARCHIVES

R&D

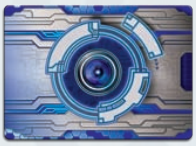
REMOTE SERVER

REMOTE SERVER

ICE

ICE

ICE



TOKEN BANK

RUNNER PLAY AREA

Program Row

Hardware Row

RIG

Resource Row

IDENTITY CARD

STACK

HEAP

CREDIT POOL

CLICK TRACKER

SCORE AREA

Corporation Cards

There are six types of Corporation cards: identities, operations, agendas, ice, upgrades, and assets. All cards except the identity card are shuffled into the Corporation's deck at the beginning of the game. Corporation cards are installed facedown, and are inactive unless rezzed (see "Rezzed and Unrezzed Cards" on page 12).

Corporation Card Anatomy Key

1. Play cost
2. Title/Subtitle
3. Card type: Subtype
4. Text box
5. Faction affiliation
6. Influence value
7. Set info
8. Minimum deck size
9. Influence limit
10. Advancement requirement
11. Agenda points
12. Rez cost
13. Trash cost
14. Strength

Corporation Identity Card

Identity cards indicate which identity the Corporation has assumed.

The Corporation identity card defines the Corporation's faction and describes the identity's special ability. It also provides a minimum deck size that must be observed when deckbuilding **(8)** and the amount of influence available for spending on out-of-faction cards **(9)**. See "Deckbuilding" on page 24 for more information.

Note: The Corporation's identity card also represents his HQ for the purposes of card installation: ice protecting HQ is installed in front of the Corporation's identity card, and upgrades installed in the root of HQ are installed behind the Corporation's identity card.

Operations

Operations represent singular occurrences and are always trashed after being played.

The Corporation pays credits equal to the play cost **(1)** of an operation to play it. When played, an operation's abilities as listed in its text box **(4)** are resolved. Then, the operation is immediately trashed. Operations are never installed.

Agendas

Agendas are valuable pieces of the Corporation's data, and are the only cards in *Android: Netrunner* that are worth agenda points.

The Corporation installs agendas in remote servers. Agendas are the only cards in the game worth agenda points **(11)**. Agendas have an advancement requirement **(10)** that must be met before the Corporation can score them (see "Advancing a Card" on page 14).

Agendas cannot be rezzed and are only active while in a score area. There can be only one agenda **or** one asset installed in a remote server at a time.

Ice

Ice defends the Corporation's servers against intrusions by the Runner.

The Corporation installs ice in front of any server. Ice is not active until it is rezzed by paying credits equal to its rez cost **(12)**.

A piece of ice has one or more **SUBROUTINES** (↳) in its text box **(4)** that the Runner must break during a run or suffer their effects (see "Ice" on page 16) if the ice is rezzed.

Upgrades

Upgrades are improvements to a server that provide the Corporation with a wide variety of benefits and bonuses.

The Corporation installs upgrades in remote servers or the roots of central servers. Upgrades are the only card type that can be installed in the root of a central server. An upgrade is not active until it is rezzed by paying credits equal to its rez cost **(12)**.

There is no limit to the number of upgrades that can be installed in a server. When the Runner accesses an upgrade, he can trash it by paying credits equal to its trash cost **(13)**.

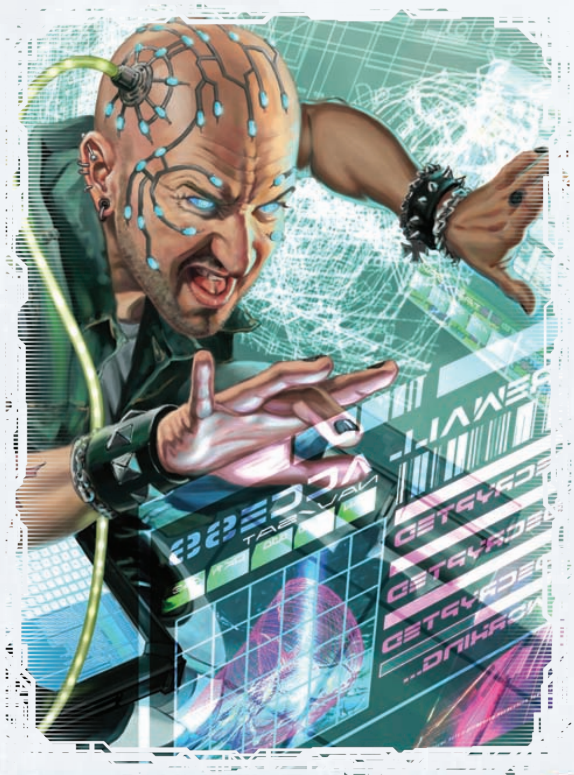
Assets

Assets provide the Corporation with resources and connections that help him advance and score his agendas.

The Corporation installs assets in remote servers. An asset is not active until it is rezzed by paying credits equal to its rez cost **(12)**.

Some assets can also be advanced, giving them the appearance of agendas and potentially misleading the Runner. When the Runner accesses an asset, he can trash it by paying credits equal to its trash cost **(13)**.

There can be only one agenda **or** one asset installed in a remote server at a time.



Runner Cards

There are five types of Runner cards: identities, hardware, resources, programs, and events. All cards except the identity card are shuffled into the Runner's deck at the beginning of the game. Runner cards are always active while installed.

Runner Card Anatomy Key

1. Title/Subtitle
2. Faction affiliation
3. Base link
4. Card type: Subtype
5. Text box
6. Minimum deck size
7. Influence limit
8. Set info
9. Install cost
10. Influence value
11. Memory cost
12. Strength
13. Play cost

Runner Identity Card

Identity cards indicate which identity the Runner has assumed.

The Runner identity card defines the Runner's faction and describes the identity's special ability. It also provides a minimum deck size that must be observed when constructing a deck **(6),** and the amount of influence available for spending on out-of-faction cards **(7)**. See "Deckbuilding" on page 24 for more information.

Hardware

Hardware is the array of physical tools at the Runner's disposal.

The Runner installs hardware in his rig by paying an install cost **(9).**

There is no limit to the amount of hardware the Runner can install in his rig.

Resources

Resources are a wide variety of connections, assets, and skills that aid the Runner.

The Runner installs resources in his rig by paying an install cost **(9).**

There is no limit to the number of resources the Runner can install in his rig.

When the Runner is **TAGGED** (see "Tags" on page 20), resources may be trashed by the Corporation.

Programs

Programs are digital tools at the Runner's disposal, primarily used as a means of intrusion.

The Runner installs programs in his rig by paying an install cost **(9).**

Programs are the only card type that have a memory cost **(11).** The memory cost of his installed programs can never exceed his current memory limit (see "Programs" on page 15).

The Runner uses a program subtype called an **ICEBREAKER (4)** to break ice subroutines during runs (see "Icebreakers" on page 16). An icebreaker's strength **(12)** must be equal to or greater than the ice it is interacting with.

Events

Events represent singular occurrences and are always trashed after being played.

The Runner pays credits equal to the play cost **(13)** of an event to play it. When played, an event's abilities as listed in its text box are resolved. Then, the event is immediately trashed. Events are never installed.

Paid Abilities

Some card abilities have trigger costs that a player must pay before the effect of the ability can be resolved. These abilities are called **PAID ABILITIES**. A card's trigger cost is always listed in its text box before the effect, following the format "cost: effect."

The most common costs are spending clicks (), spending credits (), trashing the card (), and spending hosted counters. Some effects feature a combination of costs.

Example: The Runner card Datasucker has the text "Hosted virus counter: Rezzed piece of ice currently being encountered has –1 strength until the end of the encounter." The Runner must spend 1 of the virus counters on Datasucker (returning it to the token bank) in order to trigger this ability, after which the strength of the chosen ice is lowered by 1.

If the player cannot pay the full cost of an ability, he cannot trigger it.

Unique Cards

Some cards have a unique symbol (♦) in front of their title. There can be only one unique card of the same title active at a time. If a card with a unique title becomes active, any other card that shares its title is immediately trashed. This trashing cannot be prevented.

Playing the Game

In *Android: Netrunner,* the Corporation and the Runner alternate taking turns. *Android: Netrunner* is unusual in that the Runner and the Corporation are governed by different rules. Players should familiarize themselves with the rules for both sides.

The Corporation always takes the first turn of the game.

Turn Overview

Each player, during his turn, takes **ACTIONS** by spending clicks. A player can only spend his clicks during his own Action phase, and he **must** spend all of his clicks in each Action phase. The Corporation begins his turn with three clicks (Ⓐ Ⓐ Ⓐ) and the Runner begins his turn with four clicks (Ⓐ Ⓐ Ⓐ Ⓐ).

Corporation's Turn

The Corporation's turn consists of three phases, which he performs in the following order:

1. **Draw Phase:** The Corporation draws one card from R&D.
2. **Action Phase:** The Corporation has Ⓐ Ⓐ Ⓐ with which to perform actions.
3. **Discard Phase:** The Corporation discards down to his maximum hand size, if necessary.

1. Draw Phase

The Corporation draws the top card of R&D. This does not cost the Corporation any clicks.

Note: If the Corporation's R&D is empty when he attempts to draw a card, the Runner immediately wins the game.

2. Action Phase

In his Action phase, the Corporation takes actions by spending Ⓐ Ⓐ Ⓐ. He can only take actions during his Action phase, and he must spend all three of his clicks during his Action phase.

The Corporation can perform any of the following actions as many times as he likes, and in any combination, provided he can pay for them. These are listed in the format of "cost: effect."

- Ⓐ: Draw one card from R&D.
- Ⓐ: Gain 1⓪ (one credit).
- Ⓐ: Install an agenda, asset, upgrade, or piece of ice.
- Ⓐ: Play an operation.
- Ⓐ, 1⓪: Advance a card.
- Ⓐ, 2⓪: Trash a resource in the Runner's rig if the Runner is **TAGGED**.
- Ⓐ Ⓐ Ⓐ: Purge virus counters.
- Trigger a Ⓐ ability on an active card (cost varies).

Whenever the Corporation spends clicks on one of these actions, he is considered to be taking an action and cannot take another action until the current action fully resolves.

When the Corporation has spent all of his clicks, his Action phase ends and his Discard phase begins.

Rezzed and Unrezzed Cards

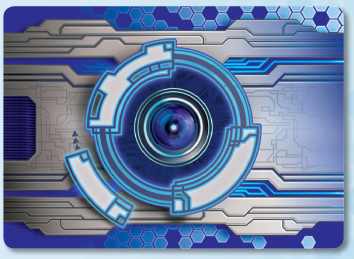
The Corporation's installed cards have two play states: **REZZED**, which means that the card is faceup and active, and **UNREZZED**, which means that the card is facedown and inactive. The Corporation can look at his unrezzed cards at any time. To rez an installed card, the Corporation pays its rez cost and turns the card faceup.

Note: Rezzing a card does not cost the Corporation a click.

To organize this hidden information for both players, it is important that the Corporation observes the following rules for card orientation:

- Agendas, assets, and upgrades are always installed in a vertical orientation.
- Ice is always installed in a horizontal orientation.

Installed Asset (rezzed)



Installed Ice (unrezzed)

Drawing One Card

For ⓐ, the Corporation draws the top card of R&D and adds it to HQ.

Gaining One Credit

For ⓐ, the Corporation takes 1 ⓒ from the bank and adds it to his credit pool.

Installing Cards

For ⓐ, the Corporation installs a single agenda, asset, upgrade, or piece of ice from HQ, placing it facedown on the table.

Note: When an asset or upgrade is installed, the Corporation can pay its rez cost to rez it at almost any time (see the "Timing Structures" on pages 32-33). Ice can only be rezzed when the Runner approaches it during a run (see "Approaching Ice" on page 17).

When installing a card in a server, the Corporation can first trash any cards already installed in that server. Trashed cards go to Archives faceup if they are rezzed, and facedown if they are unrezzed.

If the Corporation chooses to create a remote server when installing a card, he installs the card by placing it facedown in a discrete location in his play area. Agendas, assets, upgrades, and ice can all be used to create a new remote server. If the Corporation creates a remote server by installing ice, the server exists, but is considered to be **EMPTY**. An empty server can still be run against by the Runner.

Note: Installed cards cannot be rearranged or mixed-up by either player except through card effects.

The following entries describe the installation restrictions and associated costs of each card type:

Agendas– An agenda can only be installed in a remote server. After an agenda is installed, the Corporation can advance and ultimately score it (see "Advancing a Card" on page 14).

Note: A remote server can have only one agenda or asset installed in it at a time.

If the Corporation wants to install an agenda in a remote server that has an asset or an agenda already installed in it, he can install the card but ***must*** trash the existing card first as part of the install action. The Corporation does not have to trash upgrades in order to install an agenda or an asset.

Assets– An asset can only be installed in a remote server.

If the Corporation wants to install an agenda in a remote server that has an asset or an agenda already installed in it, he can install the card but ***must*** trash the existing card first as part of the install action.

Upgrades– An upgrade can be installed in any server. When an upgrade is installed in a central server, it is installed in the central server's root.

Unlike an agenda or asset, there is no limit to the number of upgrades the Corporation can install in any server, central or remote.

Note: The Corporation can only have one upgrade with the **region** subtype installed per server or server root, as listed in the text box of these cards.

Ice– Ice can be installed in front of any server in order to protect that server. After a piece of ice is installed in front of a server, it is dedicated to that server and cannot be moved or rearranged.

When the Corporation installs a piece of ice, he ***must*** install it in the outermost position in front of the server and pay an install cost equal to the number of pieces of ice already protecting that server. The outermost position is the position farthest from the server, in front of any other pieces of ice that are protecting the server.

When installing ice, the Corporation can first trash any ice protecting that server in order to reduce the install cost. Then, he installs the new piece of ice in the outermost position in front of the server.

Install Example

This remote server has a rezzed asset installed in it, protected by two pieces of ice. If the Corporation wants to install a third piece of ice to protect this server, he will have to pay 2 ⓒ (one for each piece of ice already installed) and place it in front of Ice Wall in the outermost position. The Corporation can trash one or both pieces of ice before installing to lower this cost.

Playing Operations
For ⓐ, the Corporation plays an operation from his hand by paying its play cost. He then places it faceup in his play area, immediately resolves the effects of the operation, and trashes it.

Advancing a Card
For ⓐ and 1⟲, the Corporation adds one advancement token to an installed card. Agendas can always be advanced while installed. Cards other than agendas can only be advanced if their text box allows it. There is no limit to the number of times a card can be advanced.

Note: If a card's text box says that the card can be advanced, the card can be advanced even when the card is unrezzed.

Scoring Agendas– When the number of advancement tokens on an agenda is equal to or higher than its advancement requirement, the agenda is **FULLY ADVANCED** and the Corporation can score it. The only times the Corporation can score an agenda is right before his turn begins, or after he completes an action.

To score an agenda, the Corporation turns it faceup and places it in his score area, resolving any conditional abilities on the agenda that use the language "When you score." The Corporation cannot score an agenda until it is fully advanced. Scoring an agenda does not cost a click and is not an action.

While an agenda is in the Corporation's score area, it is active and adds its agenda points to his score.

Advancing Assets
Some assets can be advanced. Advancing assets gives them the appearance of being agendas. This can be useful in bluffing the Runner into making runs which are not beneficial to him.

Delayed Scoring– An agenda sometimes has an ability that rewards advancement beyond the agenda's advancement requirement, or an ability that encourages the Corporation to delay scoring the agenda. The Corporation is not required to score an agenda immediately upon satisfying its advancement requirement. He may instead advance it more, or wait for a more opportune time to score it.

Trashing a Runner's Resource
If the Runner is tagged, the Corporation can spend ⓐ and 2⟲ to choose one of the Runner's resources and trash it (see "Tags" on page 20).

Purging Virus Counters
For ⓐ ⓐ ⓐ, the Corporation removes all virus counters **HOSTED** (see "Hosting" on page 22) on cards, returning them to the token bank.

Triggering ⓐ Abilities
Some cards have abilities with trigger costs that require the Corporation to spend one or more clicks. These abilities list the ⓐ icon in their trigger cost, and the Corporation can trigger these abilities only during his Action phase.

3. Discard Phase
The Corporation begins the game with a maximum hand size of five cards, but card effects can increase or decrease this limit. If the cards in HQ exceed the Corporation's current maximum hand size at the beginning of the Discard phase, he must **DISCARD** down to his maximum hand size.

Trashing and Discarding
A discarded card is not considered to have been trashed, and vice versa. Cards that prevent a card from being trashed cannot prevent a card from being discarded.

If the Corporation must discard more than one card from HQ, he chooses and discards cards from HQ one at a time until he is no longer above his current maximum hand size.

Cards discarded from HQ are always sent to Archives facedown, regardless of whether they have been previously accessed by the Runner.

After the Corporation completes his Discard phase, the Runner begins his turn.

Runner's Turn

The Runner's turn is divided into two phases, which he performs in the following order:

1. **Action Phase:** The Runner has [click][click][click][click] with which to perform actions.
2. **Discard Phase:** The Runner discards down to his maximum hand size, if necessary.

1. Action Phase

In his Action phase, the Runner takes actions by spending [click][click][click][click]. He can only take actions during his Action phase, and he must spend all four of his clicks during his Action phase.

The Runner can perform any of the following actions as many times as he likes, and in any combination, provided he can pay for them. These are listed in the format of "cost: effect."

- [click]: Draw one card from the stack.
- [click]: Gain 1[credit].
- [click]: Install a program, resource, or piece of hardware.
- [click]: Play an event.
- [click], 2[credit]: Remove one tag.
- [click]: Make a run.
- Trigger a [click] ability on an active card (cost varies).

Whenever the Runner spends clicks on one of these actions, he is considered to be taking an action and cannot take another action until the current action fully resolves.

When the Runner has spent all of his clicks, his Action phase ends and his Discard phase begins.

Drawing One Card

For [click], the Runner draws the top card from his stack and adds it to his grip.

Gaining One Credit

For [click], the Runner takes 1[credit] from the bank and adds it to his credit pool.

Installing Cards

For [click], the Runner installs a single program, resource, or piece of hardware faceup in his rig. An installed Runner card is active and does not have to be rezzed like a Corporation card.

Note: The Runner's cards are always installed faceup and in a vertical orientation.

Programs– To install a program, the Runner pays the program's install cost and places it in his program row. Each program also has a memory cost. The Runner cannot have programs installed that have a combined memory cost greater than his available **MEMORY UNITS** (MU). The Runner begins the game with four MU, though certain card effects can increase or decrease this value.

If the MU costs of the Runner's installed programs ever exceed his available MU, he must trash his installed programs until he is no longer exceeding his available MU.

The Runner can choose to trash any number of his installed programs at the beginning of an install program action.

Resources– To install a resource, the Runner pays the resource's install cost and places it in his resource row.

There is no limit to the number of resources a Runner can have installed.

Hardware– To install a piece of hardware, the Runner pays the hardware's install cost and places it in his hardware row.

There is no limit to the amount of hardware a Runner can have installed.

Note: The Runner can only have one piece of hardware with the **console** subtype installed at a time, as listed in the text box of these cards.

Playing Events

For [click], the Runner plays an event from his hand by paying its play cost. He then places it faceup in his play area, immediately resolves the effects of the event, and trashes it.

Removing Tags

For [click] and 2[credit], the Runner removes one of his tags.

Making a Run

For [click], the Runner initiates a run against the Corporation (see "Runs" on page 16) in order to steal the Corporation's agendas and trash his cards.

Triggering [click] Abilities

Some cards have abilities with trigger costs that require the Runner to spend one or more clicks. These abilities list the [click] icon in their trigger cost, and the Runner can trigger these abilities only during his Action phase.

2. Discard Phase

The Runner begins the game with a maximum hand size of five cards, but card effects can increase or decrease this limit (see "Brain Damage" on page 20). If the cards in the Runner's grip exceed his current maximum hand size at the beginning of the Discard phase, he must discard down to his maximum hand size.

If the Runner must discard more than one card from his grip, he chooses and discards cards from his grip one at a time until he is no longer above his current maximum hand size.

After the Runner completes his Discard phase, the Corporation begins his turn.

Runs

Runs are the heart of *Android: Netrunner*, and provide opportunities for the Runner to steal the Corporation's agendas and trash his cards. In a run, the Runner attacks one of the Corporation's servers in an attempt to access cards, using his installed programs to help him pass the Corporation's ice.

Because most runs pit the Runner's installed icebreaker programs against the Corporation's installed ice, it is vital that both players understand the functions and subtypes of the Corporation's ice and the Runner's icebreakers.

Ice

Ice is defensive software the Corporation installs in front of his servers to protect his valuable data. There are four main subtypes that can appear on a piece of ice: **sentry**, **barrier**, **code gate**, and **trap**. Ice also has separate abilities called **subroutines**.

A piece of ice

Subroutines

Subroutines are abilities of a piece of ice marked by the ↳ symbol. If the Runner encounters a piece of rezzed ice and does not or cannot break its subroutines, the unbroken subroutines trigger and resolve one by one.

In addition to preventing the Runner's access to the Corporation's servers by ending his run, subroutines can pose other hazards if allowed to trigger, such as damaging the Runner or initiating trace attempts (see "Traces and Tags" on page 20).

Icebreakers

Icebreakers are programs with the **icebreaker** subtype that the Runner can use to overcome ice encountered during a run. Each icebreaker has a strength, an install cost, and one or more subtypes that reflect which kind of ice subroutine it is designed to break.

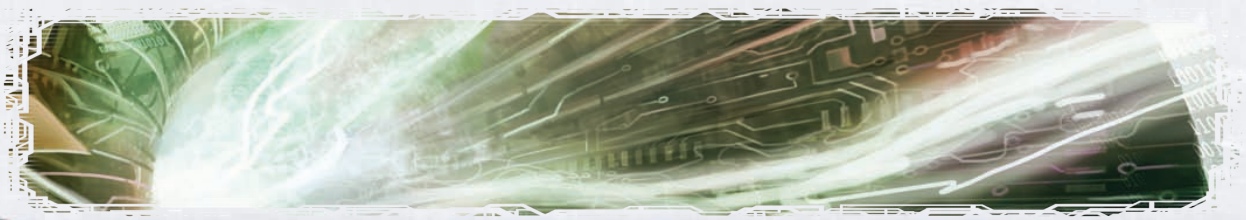
The Runner uses icebreakers to interact with and break subroutines on ice. An icebreaker can only interact with ice that has equal or lower strength than the icebreaker.

In addition to this strength requirement, many icebreaker abilities can only be used to break subroutines on particular subtypes of ice. For example, an icebreaker that has the ability "1: Break **barrier** subroutine" can only use this ability to break subroutines on a piece of ice with the barrier subtype. It does not matter if the ice has additional subtypes, provided it has ***any*** subtypes referred to by the icebreaker's ability. If an ability does not restrict itself to a subtype (i.e., "Break ice subroutine"), it can be used against any piece of ice.

An icebreaker

Increasing an Icebreaker's Strength

Many icebreakers allow the Runner to temporarily increase the icebreaker's strength by spending credits. This helps the Runner deal with stronger pieces of ice, provided he has enough credits to spend. This strength increase lasts ***only*** while the current piece of ice is being encountered, unless otherwise noted by card abilities. After an encounter with a piece of ice, the icebreaker's strength returns to the value shown on its card. This applies to any other strength modifiers given by icebreakers as well.



Phases of a Run

Runs typically transpire in three phases. Not every run will include all of these phases. Players are encouraged to use the following text in combination with the "Timing Structure of a Run" diagram on page 33 in order to fully understand the intricacies of runs.

1. **Initiation Phase**

2. **Confrontation Phase**

3. **Access Phase**

Bad Publicity

Some cards and events in *Android: Netrunner* give the Corporation bad publicity. For each point of bad publicity the Corporation has, the Runner gains 1[credit] at the beginning of each run. The Runner may spend these credits during his run as if they were in his credit pool, but any unspent bad publicity credits return to the bank at the end of the run. Bad publicity always generates revenue for the Runner at the beginning of a run, even when the Runner makes multiple runs in a single turn.

1. Initiation Phase

To initiate a run, the Runner declares the server that he is attacking. The Runner can only initiate a run against a single server per run action.

After the Runner declares the server he is attacking, he gains 1[credit] to spend during the run for each point of bad publicity the Corporation has. Then, both players check to see if there is ice protecting the attacked server.

If there is ice protecting the server, the run proceeds to the Confrontation phase.

If there is no ice protecting the server, the run proceeds to the Access phase.

2. Confrontation Phase

The Confrontation phase consists of **APPROACHING** a piece of ice and then potentially **ENCOUNTERING** that ice. A Runner approaches each piece of ice protecting the server one at a time, starting with the outermost piece. The Runner must **PASS** each piece of ice in order to approach the next piece of ice protecting the server, continuing until all pieces of ice have been passed or until the run ends. If the Runner passes all pieces of ice protecting the attacked server, the run proceeds to the Access phase.

Approaching Ice

When the Runner approaches a piece of ice, he must first decide whether he wishes to continue the run or **JACK OUT**. If he decides to jack out, he ends his run and the run is considered unsuccessful. **The Runner cannot jack out while approaching the first piece of ice during a run.**

If the Runner decides to continue instead of jacking out, the Corporation has the opportunity to rez the approached piece of ice and any other non-ice cards.

Note: The Corporation can only rez ice when it is approached.

If the approached piece of ice is rezzed after the Corporation has the opportunity to rez cards, then the Runner encounters it.

If after rezzing cards the approached piece of ice is not rezzed, then the Runner passes it. He then continues the run by either approaching the next piece of ice protecting the server or proceeding to the Access phase if there is no more ice to approach.

Approach Example

Encountering Ice

When the Runner encounters a piece of ice, he has the opportunity to break any subroutines on that piece of ice. After the Runner finishes breaking any subroutines that he wishes to break, each unbroken subroutine on that ice triggers in the order as listed on the card. If a subroutine ends the run, then the run ends immediately and no further subroutines on that piece of ice trigger.

Breaking Subroutines– To break a subroutine, the Runner uses abilities on his installed icebreakers. The Runner can break the subroutines on the encountered ice in any order he chooses. There is no limit to the number of installed cards a Runner can use to interact with the encountered ice, but he generally only needs one icebreaker. Remember that before an icebreaker can interact with a piece of ice, the icebreaker's strength must be equal to or higher than the encountered ice's strength.

Note: Breaking all subroutines on a piece of ice does not mean the ice is trashed. A passed piece of ice remains installed and is approached during every subsequent run against the server it protects.

After the Runner breaks all of the ice's subroutines and/or any effects from unbroken subroutines resolve without ending the run, he has passed that piece of ice. He then continues the run by either approaching the next piece of ice protecting the server or proceeding to the Access phase if there is no more ice to approach.

3. Access Phase

After the Runner has passed all of the ice protecting the attacked server, he has one final opportunity to jack out. If he chooses to continue, the Corporation has one final opportunity to rez cards. After rezzing cards, the run is considered to be successful and the Runner **accesses** the Corporation's cards by looking at them. The type of server attacked determines the degree and method of access, and the Runner must access cards according to the following rules:

- **R&D:** The Runner accesses the top card of R&D, and any upgrades in its root. Unless the Runner scores, trashes, or is forced by a card's text to reveal the card, he does not show cards accessed from R&D to the Corporation.
- **HQ:** The Runner accesses one random card from HQ and any upgrades in its root. Any cards the Runner does not score or trash return to HQ.
- **Archives:** The Runner accesses all cards in Archives and any upgrades in its root. The Runner turns all cards faceup when accessing them, and does not need to keep them in order. The Runner steals all agendas in Archives and cannot trash cards that are already in Archives. After accessing Archives, all cards in Archives return to Archives faceup.
- **Remote Server:** The Runner accesses all cards in the server.

Note: Installed ice is not in a server and is never accessed.

Stealing Agendas

If the Runner accesses an agenda, he steals it and places it faceup in his score area, resolving any conditional abilities on the agenda that use the language "When you steal." While an agenda is in the Runner's score area, it adds its agenda points to his score. The Runner cannot decline to steal agendas he accesses.

Trashing Cards

If the Runner accesses a card with a trash cost, he may pay credits equal to its trash cost in order to trash it to Archives faceup.

Accessing Multiple Cards

When accessing multiple cards, the Runner accesses them one at a time in any order he likes. For example, the Runner may access a card from HQ, then an upgrade installed in the root of HQ, and then another card from HQ, if he has the ability to do so.

When accessing multiple cards from R&D, the Runner must draw them in order from the top of the deck, and must return any cards not scored or trashed in reverse order, so as to preserve their positions in R&D.

The Runner must fully resolve his access to a card (steal it, pay to trash it, etc.) before accessing the next card. If the Runner scores an agenda that gives him seven or more points, he immediately wins the game, even if he would otherwise access more cards.

Concluding the Run

After the Runner has accessed all required cards, he returns any cards not stolen or trashed to their original play states. For example, an unrezzed card in a remote server returns facedown to that server, and a card accessed from HQ returns to HQ.

After a Runner finishes accessing cards, the run ends. The Runner returns any unspent bad publicity credits to the token bank, and the Runner resumes his Action phase.

Run Example

Spending his last click, Bart the Runner initiates a run against Olivia's remote server. Bart has a Gordian Blade, Crypsis, Sacrificial Construct, and The Toolbox installed. He has 5[credit]. The remote server has two unrezzed cards in it and three pieces of ice protecting it, one rezzed. One of the cards has an advancement counter on it. Olivia has 7[credit].

Since the first piece of ice protecting the attacked server is rezzed, Bart must encounter it. The Gordian Blade is already at strength 2, and Bart spends 1[credit] from The Toolbox to break Enigma's second subroutine, "End the run," and declares he is finished breaking subroutines **(1)**. The first subroutine, "The Runner loses [click], if able" resolves, but Bart has no clicks to lose.

Since the ice was passed, Bart approaches the next piece of ice protecting the server and can either continue the run or jack out. He still has 5[credit] in his credit pool and 1[credit] on The Toolbox, and decides to continue. Olivia has the opportunity to rez cards, but declines to do so. Bart then passes that piece of ice and approaches the innermost piece of ice protecting the server.

Bart once again chooses to continue the run, feeling confident with his credits and his programs in play. Olivia, with 7[credit], again has the opportunity to rez cards. She decides to rez the upgrade installed in the server by spending 1[credit], and flips over Akitaro Watanabe **(2)**. This leaves her with only 6[credit]. Her third piece of ice is a Wall of Thorns. While normally this ice would be too expensive for her to rez, Akitaro Watanabe lowers the rez cost of ice protecting that server by 2. She rezzes the piece of ice by paying 6[credit], leaving her with no credits **(3)**.

Bart encounters Wall of Thorns, spending 1[credit] from The Toolbox and 4[credit] from his pool to boost the strength of Crypsis to 5 **(4)**. With only 1[credit] left he cannot break both subroutines on the Wall of Thorns. He breaks the "End the run" subroutine by spending 1[credit] **(5)**, and then must either remove 1 hosted virus counter from Crypsis or trash it. Since there are no virus counters on Crypsis, Bart decides to use his Sacrificial Construct and triggers its prevent effect, trashing it instead of Crypsis **(6)**.

The first subroutine on Wall of Thorns then triggers and resolves, doing 2 net damage. Bart must trash two random cards from his grip. He does so, leaving him with a single card.

Now that Bart has passed every piece of ice protecting the server, he has one last opportunity to jack out. He once again decides to continue the run. Olivia can now rez cards. Since the unrezzed card in the server is an agenda, she declines.

The run is then considered to be successful and Bart gets to access cards. The Runner chooses the order in which cards are accessed in, and Bart chooses the unrezzed card first. He flips over the agenda, steals it, and adds it to his score area **(7)**, and then takes 1 net damage from Jinteki's identity card ability. This trashes the last card from his grip. He then accesses the upgrade, but since he cannot pay the trash cost, Akitaro Watanabe remains installed. The run then ends.

Traces and Tags

Though the Corporation spends much of the game repelling the Runner's intrusions, traces and tags give the Corporation opportunities to attack the Runner.

Traces

Some card abilities initiate a trace on the Runner. Traces are marked by the language "**Trace**[X]" on a card, with X equaling the base trace strength of the trace. Traces pit the Corporation's trace strength against the Runner's link strength, both of which are increased by spending credits.

The Corporation acts first during a trace, openly spending any number of credits to increase his **TRACE STRENGTH** by one point for each credit he spends. There is no limit to the number of credits the Corporation can spend on the trace.

After the Corporation spends his credits, the Runner has the opportunity to openly spend credits to increase his **LINK STRENGTH**. The Runner's base link strength is equal to the number of **LINKS** (⧉) he has in play. The Runner increases his link strength by one point for each credit he spends. There is no limit to the number of credits the Runner can spend on the trace.

After the Runner finishes increasing his link strength, it is compared to the Corporation's trace strength. If the trace strength exceeds the link strength, the trace is successful and any "If successful" effects associated with the trace are resolved. If the link strength is equal to or greater than the trace strength, then the trace is unsuccessful, and any "If unsuccessful" effects associated with the trace are resolved.

Trace Example

A Runner encounters Data Raven, and is unable to break the trace subroutine. The Runner's identity card is Kate "Mac" McCaffrey (link of 1) and he has one copy of Access to Globalsec (link of 1) in his rig, for a base link strength of 2. The Data Raven has a base trace strength of 3, and the Corporation decides to spend 2⟐, increasing the Data Raven's trace strength to 5. This means that the Runner would need to spend 3⟐ in order to make the trace unsuccessful. The Runner has 7⟐ in his pool and decides to spend 3⟐, matching the Corporation's trace strength. Because the trace was unsuccessful, no power counter is placed on Data Raven.

Tags

Certain card effects result in a tag being placed on the Runner. As long as the Runner has at least one tag, he is considered to be **TAGGED**. While the Runner is tagged, the Corporation may, as an action, spend ⓐ and 2⟐ to trash one of the Runner's resources. Certain card effects can also trigger off of the Runner being tagged, and it is usually dangerous for the Runner to remain tagged for very long.

While tagged, the Runner may, as an action, spend ⓐ and 2⟐ to remove the tag, returning it to the token bank. The Runner can repeat this action as many times he likes, provided he has the clicks and credits to pay its cost, and as long as he has a tag to remove.

Damage

Many cards and ice subroutines inflict damage on the Runner. The Runner can receive the following three types of damage:

- **MEAT DAMAGE:** The Runner randomly trashes one card from his grip for each point of meat damage done to him.
- **NET DAMAGE:** The Runner randomly trashes one card from his grip for each point of net damage done to him.
- **BRAIN DAMAGE:** The Runner randomly trashes one card from his grip for each point of brain damage done to him, and his maximum hand size is permanently reduced by one card. The Runner takes a brain damage token to track this.

Note: The only differences between net and meat damage are the cards that inflict and prevent them.

When the Runner trashes multiple cards for damage, the cards are placed in his heap in the order they were randomly trashed.

If the Runner takes more damage than the number of cards in his grip, or if he has a maximum hand size of less than zero at the end of his turn, then he is **FLATLINED** and the Corporation wins the game.

Winning the Game

If at any time a player has seven agenda points in his score area, he immediately wins the game.

If R&D contains no cards and the Corporation attempts to draw a card, the Runner immediately wins the game.

If the Runner is flatlined (see "Damage" above), the Corporation wins the game.

Additional Rules

The following sections describe additional important rules and information not addressed in the previous sections.

Card Abilities

There are two different types of card abilities in *Android: Netrunner*: **CONSTANT ABILITIES** and **TRIGGERED ABILITIES**. The following information explains how these abilities function in the game.

Constant Abilities

Constant abilities continually affect the game as long as the card they appear on is active and any other specified conditions are met. They are not triggered and do not have costs associated with them. An example of a constant ability is the card Experiential Data, which reads, "All ice protecting this server has +1 strength."

Triggered Abilities

In order to use a triggered ability a prerequisite must be met. This prerequisite is either a trigger cost that must be paid (**PAID ABILITY**) or a trigger condition that must be met (**CONDITIONAL ABILITY**). Once an ability is triggered, its effect is resolved immediately and can only be stopped by **PREVENT** or **AVOID** effects. Players must follow all restrictions on the cards when triggering abilities.

Paid abilities– In order to trigger a paid ability, a trigger cost must be paid. The most common trigger costs are spending clicks, credits, or hosted counters, and trashing cards. A card's trigger cost is always listed in its text box before the effect, following the format "cost: effect." A paid ability can be triggered an unlimited number of times as long as the cost is paid and any restrictions specified by the effect are observed. Paid abilities can be triggered at the beginning of each turn, before and after each player action, and at certain points during a run, unless the ability requires a click, in which case it must be triggered as an action. An example of a paid ability is the card Magnum Opus, which reads, ": Gain 2."

Conditional abilities– In order for a conditional ability to trigger, a trigger condition must be met. A conditional ability can only be resolved ***once*** per trigger condition. Trigger conditions commonly use the terms "When" or "Whenever" in their card text. An example of a conditional ability is the card PAD Campaign, which reads, "Gain 1 when your turn begins."

If a conditional ability uses the word "may" in its description, it is an optional conditional ability. The decision to trigger the ability belongs to the player who controls the card, provided the ability's trigger condition is met. If a conditional ability does not use the word "may" in its description, it is a required conditional ability. It must be triggered when its trigger condition is met, although the exact time of resolution may vary (see "Simultaneous Effects" on page 22).

Note: Ice subroutines are required conditional abilities that can be broken, in which case they do not resolve.

Other Terms and Concepts

There are several other terms and concepts that players should know when resolving abilities.

Timing Priority

Whenever there is an opportunity to trigger paid abilities, rez cards and/or score agendas (usually at the beginning of a turn and after each action), the player who is currently taking his turn gets the first opportunity to act. He can trigger as many abilities, rez as many cards, and/or score as many agendas as he wishes in the order of his choosing. When he is finished, the other player gets the opportunity to act. When that player is finished, the first player gets the opportunity to act once again.

After both players have had at least one opportunity to act and a player declines to act, then the players cannot trigger more abilities, rez more cards, or score more agendas until the next opportunity to do so.

For more information on the intricacies of triggering card abilities, rezzing cards, and scoring agendas, consult the timing diagrams on pages 32-33.

Prevent or Avoid

Some card abilities use the words "prevent" or "avoid." Prevent or avoid effects are the only effects which can disrupt another effect. A prevent or avoid effect states what it is preventing or avoiding, and an effect that is prevented or avoided is not resolved. Prevent or avoid effects can be triggered whenever the effect they are preventing or avoiding is resolving.

Self-referential Language

Unless otherwise noted, a card with text that refers to its own card title only refers to itself and does not refer to other copies of cards with that title.

Negative Effects

If an effect prohibits a player from doing something, usually by using the word "cannot," it always takes precedence over other effects unless another effect explicitly overrides it.

Trashing
When trashing a card as part of a trigger cost for its own paid ability (), the effect on that card will resolve even though the card is no longer active.

Expose
Some effects expose one or more cards. Generally, only unrezzed installed cards can be exposed, unless an ability specifies otherwise. An exposed card is revealed to all players, and then returned to its previous state. If multiple cards are exposed by one effect, they are considered to be exposed simultaneously.

Simultaneous Effects
When one or more abilities have the same timing trigger or can be triggered at the same time, each player chooses the order his own abilities trigger. A player can trigger an optional conditional ability before a required conditional ability if they both have the same trigger condition.

Simultaneous Effect Example
The Runner has Aesop's Pawnshop and Wyldside installed and both have the same trigger condition of "When your turn begins." The Runner begins his turn and can choose to trigger the optional conditional ability on Aesop's Pawnshop first, gaining 3 by trashing Wyldside. This stops Wyldside's required conditional ability from triggering, keeping the Runner from losing .

If players ever want to perform simultaneous effects at the same time, the player whose turn it is resolves all of his effects first.

Hosting
Some cards can only be installed on other cards; others allow cards to be installed on them. A card that has other cards installed on it is called the "host card," while the card installed on it is called the "hosted card." Hosted cards can leave play without affecting their host.

Cards can also host counters and tokens. Hosted counters or tokens can be spent, or leave play, without affecting their host. If a trigger cost requires one or more hosted counters, those counters must be spent (returned to the token bank) from the card that the ability appears on.

If a host leaves play, then all cards and counters hosted also leave play. This cannot be prevented.

Forfeiting Agendas
Some card abilities require the Corporation or Runner to forfeit an agenda. When a player forfeits an agenda, he selects any agenda in his score area and permanently removes it from the game (it does not go to Archives or the heap). He no longer scores points for the forfeited agenda.

Symbols
The following symbols appear on cards:

: This symbol stands for **CREDIT**. It always appears with a numeral, such as 1, which means "one credit," or 3, which means "three credits."

: This symbol stands for a single **CLICK**. Multiple clicks are represented by multiple symbols, such as , which means "two clicks."

: This symbol stands for **RECURRING CREDIT**. It always appears with a numeral, such as 1, which means "one recurring credit," or 3, which means "three recurring credits." Any recurring credits a player spends are replaced on their host card at the beginning of that player's turn. A player can only spend these credits as instructed by their host card.

: This symbol stands for **LINK**. It is always used with a quantity, such as +1, which means "plus 1 link."

: This symbol stands for **MEMORY UNIT**. It always appears with a quantity, such as + which means "plus 2 memory units."

: This symbol stands for **SUBROUTINE** and only appears on ice. Each symbol marks a single subroutine on a piece of ice.

: This symbol stands for **TRASH**. It is used as a self-referential trigger cost in a card text, such as ": Draw 2 cards," which means "trash this card to draw 2 cards."

Deckbuilding

In a Living Card Game, players can customize their decks by adding and removing cards, creating a unique play experience.

Why deckbuild?

Deckbuilding is a great way to experience the game in a completely new way. Instead of adapting to the game, you can force the game to adapt to you. Deckbuilding opens up new strategies, new experiences, and ultimately can lead to more exciting games where you feel more invested. When you deckbuild, you do not just participate in the game; you actively shape how the game is played.

When first building a deck, it is usually a good idea to modify one of the starter decks rather than start from scratch. After playing *Android: Netrunner* a few times with different decks, you should have a general idea of what the various cards do. Pick your favorite faction, and then modify that faction's starter deck by switching out some cards for cards from other factions. In most cases you will want to build a deck at the minimum deck size, as it makes your deck more efficient. Don't worry about building the perfect deck–enjoy the process and try out cards that are appealing to you and seem fun to play with.

When building a deck from scratch, it is generally helpful to sort your cards by faction. Once you have sorted the factions, choose one and separate those cards by card type. You will want to make sure you have a good mix of card types in your deck. Adding cards from a second core set greatly increases the number of options you will have.

One thing to consider when building a deck in *Android: Netrunner* is how to spend your influence. It is a good idea to use as much of it as possible, since there are many powerful cards in other factions. If you aren't sure what to add, look for broadly applicable cards like icebreakers or ice. For the Corporation, a surprise rez of an out-of-faction ice can be an important turning point in the game!

Another thing the Corporation should consider is how much ice you have in your deck. You will want to make sure you put in enough to stop the Runner. We recommend building about 17-20 pieces of ice into a 45-49 card deck. Also make sure you have enough ways to generate credits quicker than the regular "[click] for 1[credit]" action. Having a strong economy will give you plenty of credits to spend and put a lot of pressure on your opponent.

Once you've built your deck, it is time to play some games! This is where you will begin to understand whether or not your deck is working. Do you have enough resources? Is your ice too expensive? Are you drawing your icebreakers fast enough? Figure out what the weak points of your deck are, and try switching out some cards. Looking through your cards again, you may even have another idea for a different deck!

Restrictions

When building a deck for organized play, players must observe the following restrictions:

- A deck must be associated with a single identity card, and cannot contain fewer cards than the minimum deck size value listed on the chosen identity card. There is no maximum deck size, but the deck must be able to be sufficiently randomized in a short period of time. Identity cards, reference cards, and click tracker cards are never counted as part of a deck and do not count against the minimum deck size.
- A deck cannot have more than three copies of a single card (by title) in it.
- A deck associated with a Runner identity can never contain Corporation cards, and vice versa.
- A deck cannot contain out-of-faction cards with a total influence value that exceeds the influence limit listed on the chosen identity card (see "Influence" below). Cards that match the faction of the identity card do not count against this limit.
- A Corporation deck must have a specific number of agenda points in it based on the size of the deck, as follows:

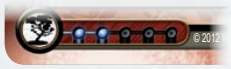
  - 40 to 44 cards requires 18 or 19 agenda points. (**Note:** Identities in this set have a 45 card minimum)
  - 45 to 49 cards requires 20 or 21 agenda points.
  - 50 to 54 cards requires 22 or 23 agenda points.

  For decks larger than this, add 2 additional agenda points to the 54 card deck requirements **each time** the number of cards in the deck reaches a multiple of 5 (55, 60, 65, etc.).

  For example, a 66 card deck requires 6 additional agenda points (2 at 55, 2 at 60, and 2 at 65 cards). This gives a final requirement of either 28 or 29 agenda points.

Influence

A player may wish to include cards in his deck that do not match the faction of his identity card. He is restricted, however, by the influence limit on his identity card. The combined influence value of out-of-faction cards in his deck cannot exceed this limit. Each card's influence value is represented by small blue orbs near the bottom of the card.



A card with 2 influence.

Neutral cards are not part of any faction, can be used in any deck of the side they are affiliated with, and generally have an influence value of zero.

Note: Some cards do not have any influence value (this is different than a card that has an influence value of zero). These cards are identified by their lack of an influence box. A card without an influence value cannot be used with an identity card that has a different faction affiliation.

Deckbuilding Example

Jenny has decided to build a Criminal deck by modifying the Criminal starter deck. She lays out all of the Criminal and Runner neutral cards, and knows that she currently has 47 cards and is spending 0 of Gabriel Santiago's 15 influence.

Jenny really likes a lot of the Shaper cards, and so she decides to browse through those and pick out some to add to her deck. After browsing, she adds the following cards to her potential card pool:

- 3x Diesel, 3x Gordian Blade, 1x The Toolbox, 2x The Maker's Eye, 2x Akamatsu Mem Chip, 3x Tinkering, 2x Magnum Opus

These cards have a combined influence of 35, and she has only 15 influence to spend. Looking at the Criminal cards, Jenny decides that she needs Gordian Blade the most, since it is a decoder, something her deck currently does not have. She puts in 2x Gordian Blade for six influence, figuring she doesn't need 3x with Special Order in the Criminal deck.

Gordian Blade

After adding in Gordian Blade, Jenny does the math and finds that in order to install all of the programs she wants to install, she will need more memory. She decides to add in 2x Akamatsu Mem Chip and 1x The Toolbox. This takes her up to 10 influence. She has only five influence left to spend.

Looking at her deck again, Jenny feels like she really wants more card draw. Since Diesel is only two influence, she could add 2x Diesel, which would put her at 14 influence. Adding in 3x Diesel would be too much influence. Looking at the cards she has already spent influence on, Jenny notices that the Akamatsu Mem Chip is only worth one. She decides to drop one of the Akamatsu Mem Chips in order to add in 3x Diesel and reach 15 influence. She really wants to add in The Maker's Eye, but decides to try the deck with just Diesel first. Having spent the 15 influence, she has now added the following cards:

Diesel

- 2x Gordian Blade, 1x Akamatsu Mem Chip, 1x The Toolbox, 3x Diesel

After maxing out her influence, Jenny counts up the current number of cards she has in her deck. She currently has 54 cards, and Gabriel Santiago has a minimum deck size of 45. She now wants to cut nine cards from the deck to reach the minimum deck size, as this makes the deck more efficient.

The first card Jenny decides to cut is 1x Desperado, since she wants to play with The Toolbox and a player can only ever have one console installed at a time. Next, Jenny decides to cut out 1x Data Dealer, since she doesn't like forfeiting agendas.

Now the decisions get tougher. Since Jenny now has a decoder, she feels like 3x Crypsis is no longer necessary, but she doesn't want to get rid of all of them. She decides to cut 2x Crypsis. Looking over her programs and icebreakers, she decides she is happy with them and sets them aside.

This leaves her with resources, hardware and events. Looking at her hardware, Jenny decides that she doesn't need 2x Lemuria Codecracker with 3x Infiltration to expose cards. She considers dropping both Lemuria Codecrackers, but one of her friends she plays with *does* like to use ambush cards. She decides to keep one in the deck, just in case. Counting up the cards she has cut, she finds that she has cut five cards, putting her current deck size at 49. If she wants to get to 45, she must cut out four more cards!

Jenny then takes a look at her resource cards. She definitely wants to keep 3x Armitage Codebusting, but she is unsure about the 2x Crash Space and 2x Decoy. These cards are great against Weyland and NBN, but not so good against Jinteki or Haas-Bioroid. She wants to cut them, but remembers the card Scorched Earth and decides to leave them in and cut 2x Access to Globalsec instead, since she feels like the Decoys will better protect her from tags.

Access to Globalsec

This leaves events. She needs to cut two more cards, and looking at the events she decides Forged Activation Orders is the weakest of the bunch. She removes two of them and breathes a sigh of relief. She has removed the following cards:

- 1x Desperado, 1x Data Dealer, 2x Crypsis, 1x Lemuria Codecracker, 2x Access to Globalsec, 2x Forged Activation Orders

She is now ready to play some games with her new deck!

Haas-Bioroid

"Effective. Reliable. Humane."

With headquarters in New Angeles and major branch offices in Chicago, Cologne, Heinlein, Johannesburg, and Sydney, Haas-Bioroid is the world leader in cybernetics and artificial intelligence. The most iconic and recognizable products made by Haas-Bioroid are the bioroids themselves, androids built with cybernetic technology and with artificially-intelligent minds designed around sophisticated imaging of human brains.

Bioroids are a new technology but have already changed humanity forever. As android labor becomes cheaper and more widely available, ordinary humans, mostly in the lower class, find themselves unemployed and replaced by a bioroid or clone. Although bioroids are less controversial than the humanlike clones, they attract a good deal of vitriol, hatred, and even violence.

The "labor solutions" market is controlled by Haas-Bioroid and their chief competitor, Jinteki. Both corporations have become enormously wealthy through their joint monopoly. Haas-Bioroid holds the patent on bioroids and most of the necessary technology for developing a proper artificial intelligence. They aggressively protect their patents and their market position through any legal means available–and, if certain alarmist watchdog organizations and fringe elements are to be believed, any illegal means available as well.

In addition to the creation of artificially-intelligent bioroids, Haas-Bioroid has been experimenting with specialized bioroids dedicated to network security and other tasks that are traditionally the role of software agents (so-called "weak" AI). Bioroids tasked for purely network usage have a proven ability to interact with the brains of users employing a neural interface, with occasionally lethal results. There are also some indications that these bioroids are less "well-adjusted" than others who possess a body and may interact with human beings in a more traditional manner. Haas-Bioroid denies any allegations that their software-purposed bioroids are unstable or have ever been implicated in the brain damage of human users.

Haas-Bioroid prides itself on quality craftsmanship and superior design. In addition to bioroids, Haas-Bioroid and its subsidiaries produce commercial-grade and medical cybernetics, prosthetics, industrial robots and machinery, mind-machine interface devices, and consumer electronics.

Jinteki

"When you need the human touch."

The traditionally conservative Jinteki corporation is now being led by an aggressive new chairman of the board, Chairman Hiro, through a series of upheavals and transitions. Alongside rapid developments in the field of cloning and biotechnology in the last decade, the corporation has relocated its headquarters from Tokyo, Japan to New Angeles, acquired or built laboratories on Mars, and shifted its recruitment policies to diversify its research and sales forces. Branch offices have also been granted more autonomy and localized marketing has increased sales of consumer-model clones (though most clone sales are still business-to-business).

This upheaval mirrors unrest in society at large in the past decades, and the cause is the same: androids. Jinteki owns the patent on the process that creates humanlike clones, biological androids tailor-made by the "genegineers" of Jinteki. As this controversial technology becomes cheaper and more robust, more and more humans find themselves replaced in the workforce by cheaper android labor. While some Jinteki corporation products (such as the vacuum-tolerant "turtleback" clones sometimes seen in Heinlein or on the Beanstalk) bear only a faint resemblance to human beings, others are virtually indistinguishable, marked only by barcode tattoos on the backs of their necks.

Jinteki markets its clones as more personable and person-like than the robotic bioroids built by their chief competitor. Clones are inherently adaptable and intuitive, just like a real person, and are able to establish empathy with real humans more easily than other androids. They excel in service industry positions, although heavy-labor and industrial-process clones are also readily available. Rumors exist of clone projects that explore the potential of human psionic ability, but such claims are dismissed by serious scientists. Jinteki has performed extensive research on the human brain and mind-machine interface technologies, but this is because so-called "braintaping" technology is essential to their production process.

The new, sleeker, more modern Jinteki prides itself on adaptability, aesthetics, and a connection to the natural world. Jinteki is proud of its heritage as a Japanese corporation and embraces a traditional aesthetic as part of its corporate identity. In addition to clones, Jinteki and its subsidiaries specialize in biotechnology, cloned organs, pharmacology, agriculture, and medical equipment.

NBN

"Someone is always watching."

The largest media conglomerate in the world is NBN, which at various times in the company's history has stood for Network Broadcast News, Net Broadcast Network, and Near-Earth Broadcast Network. Now simply known as NBN, the corporation is headquartered right on Broadcast Square in New Angeles after relocating from SanSan in the early 30s. NBN also has offices and broadcast equipment along the entire length of the New Angeles Space Elevator, particularly at Midway Station and the terminal space station known as the Castle.

NBN owns or operates five of the ten top-rated content streams worldwide. From music to threedee, news broadcasting to sitcoms, classic movies to interactive sensies, NBN does it all. NBN produces or licenses more content every day than a human being could consume in a year and boasts sophisticated secretary software agents to aid the consumer in locating the highest-quality content that best matches his user profile.

NBN's revenue streams are as complex as the web of network and broadcast infrastructure it owns. Its broad array of content and sophisticated, user-friendly delivery systems have garnered NBN an enormous number of subscribers at various membership levels in a variety of media markets. By collecting and collating viewership information and habits, NBN is also the world's leading media and marketing research firm, with zettabytes of information on such subjects as the buying habits of thirty-year-old college-educated single mothers. NBN can sell this data to other corporations, and also provide precision-targeted advertising to that same subscriber list. NBN-produced advertising uses psychographic profiling and the latest neuroscience and braintaping techniques to promote message penetration and brand retention.

The market dominance of NBN means that in most markets even non-subscribers must use NBN-owned infrastructure to access the network at all. As a result, a large percentage of data and media in all of human society passes through NBN. Privacy advocates worry that NBN has too much access and control over communications and media, and condemn NBN for its cooperation with repressive Mediterranean regimes. Some worry that NBN is using its wealth of data for purposes more nefarious than advertising, and that there is a reason why no antitrust laws were ever enforced against the corporation by U.S. or world governments.

NBN is a model of corporate efficiency, agile and responsive to an ever-changing marketplace. It does more than simply read the market; it steers it.

Weyland Consortium

"Moving Upwards"

Aside from its dramatic and public association with the New Angeles Space Elevator, better known as "Jack's Beanstalk" or simply "the Beanstalk" after designer Jack Weyland, the extent of the Weyland Consortium's holdings is little known among the general population. This shadowy organization owns or invests in other corporations, leveraging the enormous assets granted them by the Beanstalk to buy and sell smaller megacorps at an alarming rate.

For the past several decades, the Weyland Consortium's obvious specialty has been construction, a legacy of its involvement in the Space Elevator project. Many of its subsidiaries are construction companies, often on a local level, or suppliers for construction companies. By some estimates, half the arcologies in New Angeles were built by a Weyland Consortium-controlled company, and cunning accounting and business practices ensure that even when the client companies fold, the Consortium somehow comes out ahead.

Part of the secret of the Weyland Consortium's success lies in its ability to secure government contracts and lobby for favorable legislation, especially in the United States and China. It is often a war profiteer, securing lucrative reconstruction bids in the Mediterranean, United Korea, and the Sub-Saharan League nations. In the wake of the Lunar War, Weyland snatched up almost 70% of the orbital reconstruction contracts on Earth and nearly all of the Heinlein contracts. Unfortunately for Weyland, its apparent magic with local governments does not appear to extend to the Martian separatists, who consider the Weyland Consortium a corporate extension of Earth's hegemony.

Still, Weyland remains confident that the bright future of the human race is in outer space. The Consortium is a major source of funding for space exploration and continues to acquire aerospace and orbital construction companies. Some suggest that the Weyland Consortium seeks a monopoly in outer space, that it wants to control all human habitation outside Earth's atmosphere. Many of these alarmists are Martians who distrust the Weyland Consortium on principle.

Given the Weyland Consortium's proclivity for operating in war-torn regions, it should be no surprise the corporation is comfortable playing hardball. While little has been proven, some mysterious deaths are blamed on elements within the Consortium. Weyland favors a brute-force approach to most problems, using its vast resources to get their way.

Runners

Runners are a fractious and varied group. It's nearly impossible to generalize about them, except to say that individuality is core to their identity. By definition they live outside the law, and as a consequence they mostly lead a solitary existence. They do not have overarching organizations or affiliations, or indeed much of anything that makes any one runner similar to another. They come from all walks of life, vary dramatically in skill sets, goals, and available resources, and don't even have a dress code.

Anarchs

Anarchs have strong contempt for the corporate oligarchs, the whole corrupt system, and often for society in general. Whatever the exact target of their rage, their unifying characteristic is their anger. At their worst, Anarchs just want to watch the world burn. At their best, Anarchs are tireless champions for the downtrodden and oppressed. They're very good at breaking things, spreading viruses, and trashing Corporation assets and programs.

Criminals

Criminals are in it for themselves. All runners are technically criminals, at least if you ask the corps, but these runners embrace it. They make self-interest an art form and don't care who gets hurt so long as they get ahead. Many Criminals engage in more traditional forms of crime as well, stealing data and money with equal gusto. Criminals are good at covering their tracks and employing a variety of dirty tricks to attack from an unexpected angle.

Shapers

To others, **Shapers** seem like idealistic naifs. They're not motivated by rage against the corporate injustice that is a daily fact of life for the underclass. They're not in it for the money. Many never understand why Shapers do what they do, but it's not actually very complicated. Shapers are motivated by curiosity and a certain amount of pride. A Shaper may orchestrate a data raid as underhanded and destructive as the most frothing Anarch, but his goals are different: the Shaper just wants to see if he can do it. Shapers are also tinkerers and builders, and they push their hardware and software beyond their limits.

Noise

Hacker Extraordinaire

"I guess I'm just the classic example of the man who had every advantage going tragically wrong." Reilly grinned a cocksure, infuriating grin that made him look ten years younger on his already young-looking face.

"You did," said Brady, flicking through the file that hung in the air between them. "Ji Reilly. Says here you're a g-mod from birth. Mommy and daddy must have loved you very much to tinker with your brain like that."

"Oh, sure," said Reilly. "They loved me so much they planned everything out. You know I was born on Heinlein, but I went to school downstalk. Mommy and daddy lived on the moon, but there I was, living in an arcology in New Angeles. I could look out the window at night and wave at my mom and dad."

"Don't sell me that line of bull."

"It's all there in the file, Detective." Reilly flicked one long-fingered hand at the shimmering virt display. "All planned out. Internship with Jinteki's AI research division. Management position by 25. VP by 30."

"You were not a Jinteki VP."

"Oh, hell no, can you imagine?" He laughed, a short, sharp bark of a laugh. "That company would never survive me." And then the grin again. "Still might not."

"So you had every goddamn advantage a boy could dream of," sneered Brady. "And yet here you are." Reilly shrugged, doing his best to look innocent. The cuffs spoiled both effects. "So why the life of crime, Reilly?"

"Why the life of crime." Reilly wrapped his mouth and lips around each word, tasting them, weighing the ideas contained within. "Why not?"

Brady sneered, dropping his PAD to the cold steel of the table. "I know why. You get plenty rich off these little capers."

"There's what you know, and what you think you know, Detective. And there's two things you should know before we go any further."

"Yeah?"

"The first thing is that arresting me over and over does not mean that I'm a criminal. Not until you can find a charge that'll stick."

"You're no good, Reilly. The techs are going to find that stolen data on one of your datacores, somewhere. You won't be a top-flight runner anymore; they're gonna pull the cyberware out of your head and if you so much as touch a PAD they'll break your fingers. You'll just be Ji Reilly. Nobody."

"That's the other thing, Detective." He stood, resting his hands on the table. All the flippant gestures, all the mocking smiles were gone. His eyes blazed like foxfire. "My name is Noise."

Brady held his gaze until his PAD chirped. He frowned and glanced down at it. The device tracked his eye line and the virt display bloomed to life. Orders.

"So," said Noise. "I guess I'll be going now?" He offered his cuffs and grinned. Brady scowled.

Gabriel Santiago
Consummate Professional

Gabriel was hungry. Nothing new. Grew up hungry. Grew up lean. Grew up mean on the streets of New Angeles. You don't get much schooling on the streets but you do get an education. Gabriel grew up speaking three languages and cursing in three more. He learned how to spot a cop or a spydrone, how to palm a PAD from a ristie's coat pocket, how to crack the case and burn out the auto-locator without scragging the valuable electronics inside.

Being hungry gave him an edge. Had to want it more. Had to need it. Had to be willing to do what it took to get ahead. So, yeah. Gabriel was hungry. He liked it that way. Kept himself hungry. Cracking PADs turned into cracking code, turned into cracking networks. Could've gone straight, worked for HB or one of the small software startups that bloom and die like mushrooms on a corpse all through New Angeles. Gotten fat. Complacent. Lost the edge.

"Better this way," Gabriel said, hanging upside down outside the 124th floor of the Hu-Jintao arcology. A green light blinked on the small black box affixed to the window by his head, claiming the alarm was successfully disabled. Gabriel ignored it; it was linked to his cortical implant and he'd know if it needed his attention. He focused on carefully removing the cut glass from the window before him. Couldn't drop it, couldn't let the wind snatch it away. He used his good hand, his flesh-and-blood hand, for the operation. Deftly he tucked the circle of glass, about the size of his palm, into the front pocket on his vest. Then the laser probe had to be placed just so, with the beam striking the optical port on the sarariman's desk inside, and then he was in to the network.

He pulled a cable from the laser probe and socketed it into his wrist–his bad wrist, his metal wrist. An optical connection established, his implant came to life, flooding his mind with data. His sense of his body fell away; he wasn't hanging upside down a mile above the street with the wind tearing at his clothes anymore. He was in a river of data, a bodiless phantasm, a ghost in the machine.

But he was still hungry.

Kate "Mac" McCaffrey
Digital Tinker

"I like to think of myself as an artist," she said. Said. Out loud. With her vocal cords. Unplugged, perched on the edge of a stool so old it was made of wood. Her "desk" was a polywood flat laid over two sawhorses and strewn with humming, glowing electronic devices. One of these devices projected a virt display of a girl's plastic doll face, fixed in a permanent plastic grin. The face spoke back from the small speaker at the base of the projector.

"An artist of…pixels? Qubits? Bytes?"

"Ideas," said Mac. She gestured and a virt screen, a luminous panel showing bricks of raw code, floated up in front of her face. "The bits and bytes and qubits aren't the data. It's just how it's written. Like a word isn't just a collection of letters. There's an idea behind it."

"So you use digital storage media as a means to express your ideas?"

Mac ignored the question. Her own anonymizer program was probably showing her as an old film or sensie star; she couldn't remember if she'd set it for Marilyn Monroe, Charlie Chaplin, or Miranda Rhapsody. The code was good. She rested her hand flat on an induction interface panel and let the device synch with the nanowiring implanted under her skin. "There's great potential in the network, ways for us to communicate with each other, maybe new ways to structure our society. I just want to reach out and see what it can do."

"And what are you doing tonight?"

"Reaching out." She sent one final command and lifted her hand. "Are you listening?"

"I'm listening," said the doll.

Mac turned on her stool, looking out the window at the New Angeles skyline. She grinned as her handiwork wrote itself across the sky. "Try looking out the window, Ms. Lockwell."

"Why do you think I'm–oh my god." The mile-high Gila Heights arcology, all its lights flickering according to Mac's design. They circled and streaked, bloomed and exploded in a pattern of light and dark. The cycle looped three times before someone at Gila Heights managed to return control and shut it down. "Did you just flash that to all of New Angeles?" came the voice from the projector.

"Maybe," said Mac. "It would be easy. As easy as tracing you back to your office at Broadcast Square."

"You cracked the NBN firewall!? You can't do that! This is why people think you're reckless criminals. This is why–" Mac killed the feed.

"Can't do that," she mused. She gestured and the virt displays clustered on her desk showed her a great big tower of data, the inaccessible NBN network, its spine running up the Beanstalk, its ports guarded by the best ice money could buy. The diagram spun slowly in front of her. Mac grinned and cracked open a new can of Diesel. "I wonder," she said, and called up a new window full of code.

Glossary

Accessing: The act of a Runner looking at a Corporation card as part of a successful run, which he can then trash or steal.

Action: What a player performs on his turn whenever he spends one or more clicks.

Active: A state in which a card's effects and abilities are able to be used and affect the game.

Advancing: The act of putting one advancement token on a card that can be advanced. Agendas can always be advanced.

Advancement Requirement: The number of advancement tokens that must be on an agenda before the Corporation can score it.

Agenda: A Corporation card type that is installed in remote servers and is worth agenda points.

Agenda Counter: A counter used to track various effects on agenda cards.

Agenda Points: A value on agenda cards. This value is how many points an agenda is worth while it is in a score area.

Anarch: One of the three Runner factions available to a player in *Android: Netrunner.*

Approach: The step of a run in which the Runner makes contact with a piece of ice and decides whether or not to continue the run.

Archives: The Corporation's trash pile. A central server.

Asset: A Corporation card type which is installed in his remote servers and grants him various benefits.

Avoid effect: An effect that stops another effect from resolving.

Barrier: One of the four subtypes of ice which the Corporation can use to defend his servers.

Click (⏀): The basic unit of work in *Android: Netrunner*. Players spend their clicks to perform actions and trigger abilities.

Code gate: One of the four subtypes of ice that the Corporation can use to defend his servers.

Constant Ability: An ability that continually affects the game provided its card is active.

Corporation: One of the two sides available to the player in *Android: Netrunner*; the opponent of the Runner. Referred to as "the Corp" on card text.

Credit (⌾): The basic unit of wealth in *Android: Netrunner*.

Credit, Recurring (⟲⌾): A credit that, when spent, returns to its host card at the start of that player's next turn. A player can only spend recurring credits as instructed by their host.

Credit bank: The supply of credits not yet in play.

Credit pool: The supply of credits currently available to a player for spending.

Criminal: One of the three Runner factions available to a player in *Android: Netrunner.*

Damage, Brain: A unit of damage that requires the Runner to trash one card from his grip at random, and reduces his maximum hand size by one card.

Damage, Meat or Net: A unit of damage that requires the Runner to trash one card from his grip at random.

Derez: The act of flipping a rezzed card facedown, inactive.

Discard: The act by which a player moves a card to his trash pile at the end of his turn if he has exceeded his maximum hand size.

Effect: The resolution of a card ability.

Event: A single-use card type that is played by the Runner during his turn and is trashed when its effects are resolved.

Expose: The act of revealing a card to all players. Only unrezzed installed cards can be exposed unless otherwise noted. An exposed card returns to its previous state after being exposed.

Flatline: A condition that results from the Runner being forced to trash more cards than he has in his grip, or from having a maximum hand size that is below zero at the end of his turn, and which causes the Runner to immediately lose the game.

Grip: The Runner's hand of cards.

Hardware: A Runner card type which is installed in the Runner's play area and grants him various abilities.

Haas-Bioroid: One of the four Corporation factions available to a player in *Android: Netrunner.*

Heap: The Runner's trash pile.

Host: A card that is currently holding other cards or counters.

Headquarters (HQ): The Corporation's hand of cards. A central sever.

Ice: A Corporation card type which protects his servers from the Runner.

Icebreaker: A program subtype which enables the Runner to break ice.

Inactive: A state in which a card's effects and abilities are ignored.

Influence: A value that appears on certain cards which is used in deckbuilding. Influence restricts the number of out-of-faction cards in a deck.

Install: The act of placing an agenda, asset, ice, upgrade, hardware, program or resource card onto the table. The Runner installs cards in his rig, the Corporation in his servers.

Install cost: The cost which must be paid in order for a card to be installed.

Jack out: The process by which a Runner voluntarily ends his own run.

Jinteki: One of the four Corporation factions available to a player in *Android: Netrunner.*

Link (⿻): A value that increases the Runner's link strength during a trace.

Link strength: The Runner's total strength during a trace; the sum of his links and the amount of credits the Runner spends on the trace.

Maximum Hand Size: The maximum number of cards a player can have in his hand during his discard phase.

Memory Unit (MU): A unit of space available to the Runner to install programs. The Runner begins the game with four memory units.

Mulligan: The act of drawing a new hand at the start of the game. Each player gets one mulligan per game.

NBN: One of the four Corporation factions available to a player in *Android: Netrunner.*

Operation: A single-use card type that is played by the Corporation during his turn and is trashed when its effects are resolved.

Power Counter: A counter used to track various effects on cards.

Program: A Runner card type that is installed and grants him various abilities.

Prevent effect: An effect that stops another effect from resolving.

Research and Development (R&D): The Corporation's draw deck. A central server.

Resource: A Runner card type that is installed and grants the Runner various benefits.

Rez: The process by which the Corporation reveals his installed cards and allows them to take effect; once rezzed, a card is turned faceup.

Rez cost: The credits that the Corporation must pay in order to rez a card.

Root: The portion of the central server where the Corporation installs upgrades.

Runner: One of the two sides available to the player in *Android: Netrunner*; the opponent of the Corporation.

Score (noun): The number of agenda points a player has on agendas in his score area.

Score (verb): The act of the Corporation turning an installed agenda faceup and adding it to his score area. An agenda must have at least as many advancement tokens on it as its advancement requirement to be scored.

Score Area: A place where each player places his scored or stolen agendas.

Sentry: One of the four subtypes of ice which the Corporation uses to defend his servers.

Server, Central: A type of server which includes R&D, HQ, and Archives.

Server, Remote: A server built by the Corporation. Assets and agendas can only be installed in remote servers.

Shaper: One of the three Runner factions available to a player in *Android: Netrunner.*

Stack: The Runner's draw deck.

Steal: The act of the Runner adding an accessed agenda to his score area.

Strength: An attribute of programs and ice.

Subroutine (↳): An ability of a piece of ice which interferes with the Runner if allowed to trigger during a run.

Subtype: A card descriptor.

Tag: An effect that, when acquired by the Runner, can allow the Corporation to trash the Runner's resources.

Tagged: A state which describes a Runner when he has one or more tags.

Trace: An attempt by the Corporation to tag or damage the Runner.

Trace Strength: The Corporation's total strength during a trace; the sum of the base trace strength on the card initiating the trace and the amount of credits the Corporation spends on the trace.

Trap: One of the four subtypes of ice which the Corporation can use to defend his servers.

Trash: The act of moving a card to its owner's trash pile.

Triggered Ability: An ability that has a prerequisite that must be met or paid before it is used.

Upgrade: A Corporation card type that is installed in any server and grants the Corporation various abilities.

Virus Counter: A counter used to track various effects on virus cards.

Weyland Consortium: One of the four Corporation factions available to a player in *Android: Netrunner.*

TIMING STRUCTURE OF TURNS

↻ = Paid abilities can be triggered = Cards can be rezzed = Agendas can be scored

1. Corporation's Draw Phase

↻

Turn begins *("When your turn begins" conditionals meet their trigger conditions)*

Draw one card

2. Corporation's Action Phase

Take actions

After each action: ↻

Phase ends after abilities are triggered following the last spent action

3. Corporation's Discard Phase

Discard down to maximum hand size

End of turn ↻

1. Runner's Action Phase

↻

Turn begins *("When your turn begins" conditionals meet their trigger conditions)*

Take actions

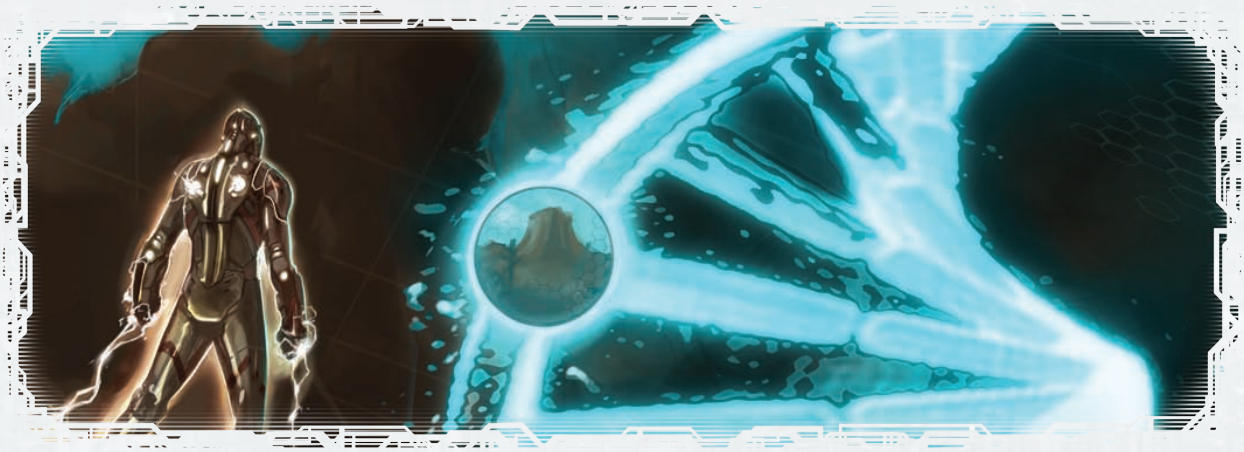
After each action: ↻

Phase ends after abilities are triggered following the last spent action

2. Runner's Discard Phase

Discard down to maximum hand size

End of turn ↻



TIMING STRUCTURE OF A RUN

↻ = **Paid abilities can be triggered** ✹ = **Cards can be rezzed**

1. The Runner initiates a **RUN** and declares the **ATTACKED SERVER**.

   • If the attacked server has one or more pieces of ice protecting it, go to [2]. If the attacked server does not have ice protecting it, go to [4].

2. The Runner **APPROACHES** the outermost piece of ice not already approached on the attacked server. ↻

   ...Either the Runner **JACKS OUT**: go to [5] *(cannot jack out if this is the first ice approached this run)*

   ...Or the Runner continues the run: if the approached ice is **REZZED,** go to [3]; if the approached ice is **UNREZZED,** go to [2.1].

   2.1 ↻ ✹ *(only time the approached ice can be rezzed)*

   ...Either the Corporation **REZZES** the approached ice: go to [3]

   ...Or the Corporation does not rez the approached ice and the runner **PASSES** it: go to [2] if there is another piece of ice protecting the server, go to [4] if there is not another piece of ice protecting the server.

3. The runner **ENCOUNTERS** a piece of ice. ↻

   3.1. The Runner can break **SUBROUTINES** on the encountered ice. ↻

   3.2. Resolve all subroutines not broken on the encountered ice.

   ...Either the run ends: go to [5]

   ...Or the run continues: if there is another piece of ice installed protecting the server, go to [2]; if there is not another piece of ice protecting the server, go to [4].

4. The Runner decides whether to continue the run. ↻

   ...Either the Runner **JACKS OUT**: go to [5]

   ...Or the Runner continues the run: go to [4.1].

   4.1. ↻ ✹

   4.2 The run is considered to be **SUCCESSFUL**.

   • Trigger any abilities resulting from successful run.

   4.3. Access cards, then go to [6].

   • If an **AGENDA** is accessed, the Runner **STEALS** it. If a card with a **TRASH COST** is accessed, the Runner may pay its trash cost to **TRASH** it.

   • All accessed cards not stolen or trashed are returned to the server in their previous states.

5. The run ends and is considered to be **UNSUCCESSFUL**.

   • Trigger any abilities resulting from unsuccessful run.

6. The run ends.

Index

Credits

Original Game Design: Richard Garfield

Game Development: Lukas Litzsinger

IP Development: Daniel Lovat Clark

Rules Text: David Hansen, Michael Hurley, and Lukas Litzsinger

Editing: David Hansen and Lukas Litzsinger

Graphic Design: Michael Silsby with Chris Beck, Shaun Boyke, Dallas Mehlhoff, Andrew Navaro, and Evan Simonet

Cover Art: Imaginary FS Pte Ltd

Art Direction: Zoë Robinson

Managing Art Director: Andrew Navaro

Licensing Coordinator: Deb Beck

Producer: Lukas Litzsinger

Production Manager: Eric Knight

Executive Game Designer: Corey Konieczka

Executive Game Producer: Michael Hurley

Publisher: Christian T. Petersen

Android Universe created by Kevin Wilson with Daniel Lovat Clark

Playtesters: Patrick John Haggerty, Brent Bartlett, Patrick Burroughs, Chris Long, Mendel Schmiedekamp, Brian Olmstead, Christopher "Tarnis Phoenix" Seefeld, WiL Springer, Mike Linnemann, Bryan Bornmueller, Maximum Deb, Joshua B. Grace, Andrew Fischer, Erik Dahlman, Nate French, Damon "Stormaggedon Dark Lord of All" Stone, Francesco Moggia, Andrew Baussan, Brad Andres, Nathan LeSueur, Kalar Komarec, Andy Christensen, "Lovey the Snake," Joseph Houff, Patrick Burroughs, John Laughlin, Brent Bartlett, Jon Waddington, Maya Waddington, Aiden Tanner, Chad Tanner, Becky Zamborsky, Steve Zamborsky, Kevin Ancell, Jimmie Hardin, jeremy barrett, James Abele, Michael Hurley, Michael Silsby, Julia Lewandowski, Mat Nowak, Josh Bailey, Johnny Anderson, Engoduun, Noshrok Grimskull, Emmanuel "Playful_EE" Estournet, Wormhole Surfer aka Ludovic Schmidt, JudgeWhyMe, The_Little_Bug, Hugo "Ophidian Lord," Will "Kennon" Lentz, Richard A. Edwards, William Edwards, Kathy Bishop, Eric Damron, Joe Becker, "The Shadow," and Nick Jordan.

Wizards of the Coast Licensing Approvals Team

Senior Creative Director: Jon Schindehette

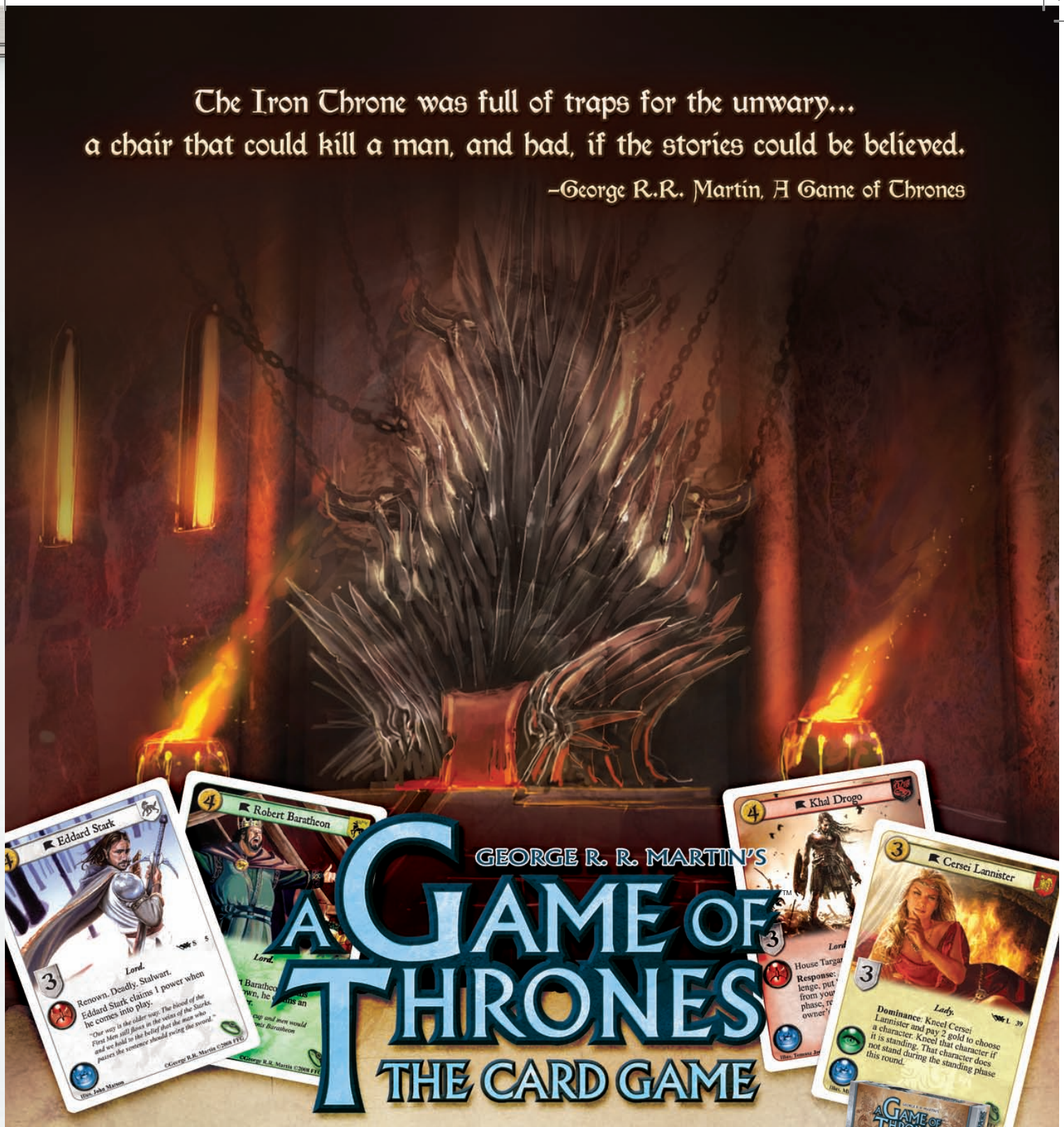
Editorial Review: Jennifer Wilkes Clark

Associate Brand Manager: Hilary Ross

www.FantasyFlightGames.com

The Iron Throne was full of traps for the unwary...
a chair that could kill a man, and had, if the stories could be believed.

–George R.R. Martin, A Game of Thrones



Can you survive the struggle for the Iron Throne?

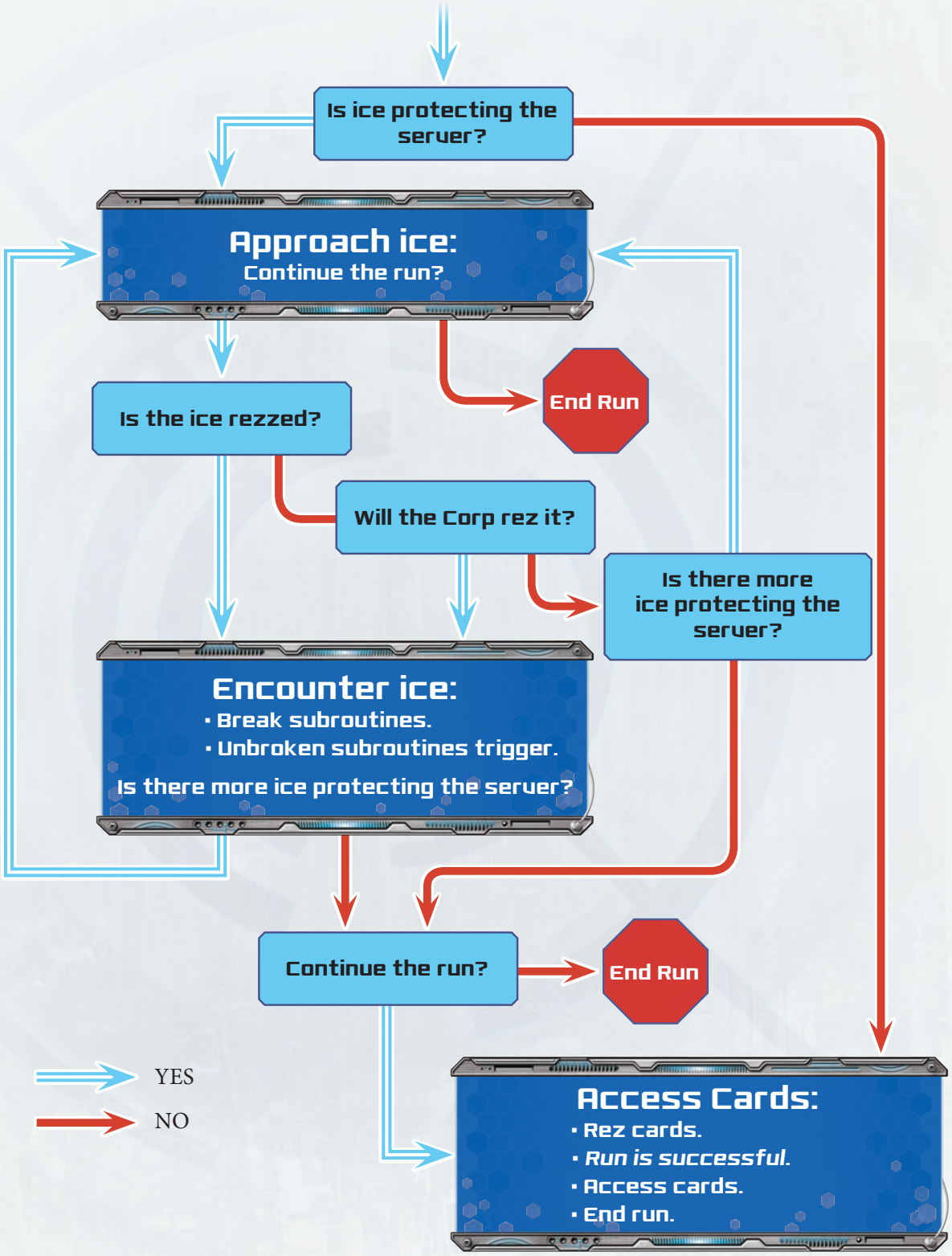
The Great Houses of Westeros war and scheme, sacrificing knights and pawns in their efforts to rule the Seven Kingdoms. *A Game of Thrones: The Card Game* allows you to relive your favorite moments from George R.R. Martin's *A Song of Ice and Fire* or rewrite history to win the Iron Throne for your favorite House.

FANTASY FLIGHT GAMES

With 220 cards, the *A Game of Thrones: The Card Game* Core Set provides you everything you need to play the only game that matters!

WWW.FANTASYFLIGHTGAMES.COM

INITIATE RUN AGAINST SERVER



List of References

[1] Bruce Abramson. Expected-Outcome: A General Model of Static Evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:182 – 193, 1990.

[2] R Agrawal, T Imielinski, and A Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(May):207–216, 1993.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. *Proceeding VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases*, 1215:487–499, 1994.

[4] Michael Albert, Richard Nowakowski, and David Wolfe. *Lessons in Play: An Introduction to Combinatorial Game Theory: An Introduction to the Combinatorial Theory of Games*. 2007.

[5] Atif M. Alhejali and Simon M. Lucas. Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 65–72, Niagara Falls, Ontario, Canada, 2013.

[6] Ethem Alpaydn. *Introduction to machine learning*, volume 1107. 2014.

[7] Ingo Althöfer. On the Laziness of Monte-Carlo Game Tree Search in Non-tight Situations. Technical report, Friedrich-Schiller Univ., Jena, 2008.

[8] Ingo Althöfer. Game Self-Play with Pure Monte-Carlo: The Basin Structure. Technical report, Friedrich-Schiller Univ., Jena, 2010.

[9] Apple.com. Siri, 2015.

[10] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte Carlo Tree Search in Hex. *IEEE Trans. Comp. Intell. AI Games*, 2(4):251–258, 2010.

[11] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2):235–256, 2002.

[12] V. Baez-Monroy and S. O'Keefe. An Associative Memory for Association Rule Mining. *2007 International Joint Conference on Neural Networks*, (2):2227–2232, 2007.

[13] Hendrik Baier and Peter D. Drake. The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go. *IEEE Trans. Comp. Intell. AI Games*, 2(4):303–309, 2010.

[14] Nicolas A. Barriga, Marius Stanescu, and Michael Buro. Parallel UCT search on GPUs. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2014.

[15] C Bauckhage, C Thurau, and G Sagerer. Learning human-like opponent behavior for interactive computer games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2781:148–155, 2003.

[16] József Beck. *Combinatorial Games: Tic-Tac-Toe Theory*. 2008.

[17] Amit Benbassat and Moshe Sipper. EvoMCTS : Enhancing MCTS-Based Players through Genetic Programming. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 57–64, Niagara Falls, Ontario, Canada, 2013.

[18] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The Challenge of Poker. (September):50–57, 1996.

[19] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings Of The National Conference On Artificial Intelligence*, volume pp, pages 493–499, 1998.

[20] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. Lower Bounding Klondike Solitaire with Monte-Carlo Planning. In *Proc. 19th Int. Conf. Automat. Plan. Sched.*, pages 26–33, Thessaloniki, Greece, 2009.

[21] Yngvi Björnsson and Hilmar Finnsson. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Trans. Comp. Intell. AI Games*, 1(1):4–15, 2009.

[22] Amine Bourki, Guillaume Maurice Jean-Bernard Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, Paul Vayssière, and Ziqin Yu. Scalability and Parallelization of Monte-Carlo Tree Search. In *Proc. Int. Conf. Comput. and Games, LNCS 6515*, pages 48–58, Kanazawa, Japan, 2010.

[23] Charles L. Bouton. Nim, A Game with a Complete Mathematical Theory. *The Annals of Mathematics*, 3:35–39, 1901.

[24] Bruno Bouzy. Associating domain-dependent knowledge and Monte Carlo approaches within a go program. *Inform. Sci.*, 175(4):247–257, nov 2005.

[25] Bruno Bouzy. Move Pruning Techniques for Monte-Carlo Go. In *Proc. Adv. Comput. Games, LNCS 4250*, pages 104–119, Taipei, Taiwan, 2005.

[26] Bruno Bouzy. Old-fashioned Computer Go vs Monte-Carlo Go (PPT). Technical report, 2007.

[27] D. M. Breuker, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Replacement schemes for transposition tables. *ICCA Journal*, 17(4):183–193, 1994.

[28] Cameron Browne. A Problem Case for UCT. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(1):69–74, 2013.

[29] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comp. Intell. AI Games*, 4(1):1–43, 2012.

[30] Bernd Brügmann. Monte Carlo Go. Technical report, Max-Planke-Inst. Phys., Munich, 1993.

[31] Michael Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1-2):85–99, 2002.

[32] Michael Buro, Jeffrey R. Long, Timothy Furtak, and Nathan R. Sturtevant. Improving State Evaluation, Inference, and Search in Trick-Based Card Games. In *Proc. 21st Int. Joint Conf. Artif. Intell.*, pages 1407–1413, Pasadena, California, 2009.

[33] Tristan Cazenave and Nicolas Jouandeau. On the Parallelization of UCT. In *Proc. Comput. Games Workshop*, pages 93–101, Amsterdam, Netherlands, 2007.

[34] Alex Champandard, Damian Isla, Daniel Kline, Michael Lewis, and Dave Mark. Never Mind Small Steps: What's the Giant Leap for AI? In *GDC 2013*, 2013.

[35] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. *Aiide*, pages 216–217, 2008.

[36] Guillaume Maurice Jean-Bernard Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Monte-Carlo Strategies for Computer Go. In *Proc. BeNeLux Conf. Artif. Intell.*, pages 83–91, Namur, Belgium, 2006.

[37] Guillaume Maurice Jean-Bernard Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel Monte-Carlo Tree Search. In *Proc. Comput. and Games, LNCS 5131*, pages 60–71, Beijing, China, 2008.

[38] Guillaume Maurice Jean-Bernard Chaslot, Mark H. M. Winands, H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. *New Math. Nat. Comput.*, 4(3):343–357, 2008.

[39] Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and Move Groups in Monte Carlo Tree Search. In *Proc. IEEE Symp. Comput. Intell. Games*, pages 389–395, Perth, Australia, 2008.

[40] Gobinda G. Chowdhury. Natural language processing. *Journal of the American Medical Informatics Association : JAMIA*, 18(5):544–51, 2011.

[41] Rémi Coulom. Computing Elo Ratings of Move Patterns in the Game of Go. *Int. Comp. Games Assoc. J.*, 30(4):198–208, 2007.

[42] Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proc. 5th Int. Conf. Comput. and Games, LNCS 4630*, pages 72–83, Turin, Italy, 2007.

[43] Peter Cowling, Sam Devlin, Edward Powley, Daniel Whitehouse, and Jeff Rollason. Player Preference and Style in a Leading Mobile Card Game. *IEEE Transactions on Computational Intelligence and AI in Games*, X(X):1–1, 2014.

[44] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information Set Monte Carlo Tree Search. *IEEE Trans. Comp. Intell. AI Games*, 4(2):120–143, 2012.

[45] Anna Cox, Paul Cairns, Pari Shah, and Michael Carroll. Not Doing But Thinking: The Role of Challenge in the Gaming Experience. *Proceedings of the CHI 2012*, pages 79–88, 2012.

[46] Chad Creighton and Samir Hanash. BIOINFORMATICS Mining gene expression databases for association rules. pages 79–86, 2003.

[47] Miguel Cristian and Greciano Raiskila. Dynamic Difficulty Adaptation for Heterogeneously Skilled Player Groups in Collaborative Multiplayer Games. In *GDC 2001*, 2001.

[48] Csikszenthmihalyi. Flow, 1990.

[49] Alena Denisova and Paul Cairns. Adaptation in Digital Games: The Effect of Challenge Adjustment on Player Performance and Experience. *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play - CHI PLAY '15*, (October 2015):97–101, 2015.

[50] Ethan Dereszynski, Jesse Hostetler, Alan Fern, Tom Dietterich, Thao-Trang Hoang, and Mark Udarbe. Learning probabilistic behavior models in real-time strategy games. *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 20–25, 2011.

[51] Heather Desurvire, Martin Caplan, and Jozsef a Toth. Using heuristics to evaluate the playability of games. *CHI'04 extended abstracts on Human factors in computing systems*, pages 1509–1512, 2004.

[52] Sam Devlin, Anastasija Anspoka, Nick Sephton, Peter I Cowling, and Jeff Rollason. Combining Gameplay Data With Monte Carlo Tree Search To Emulate Human Play. 2016.

[53] Peter D. Drake. The Last-Good-Reply Policy for Monte-Carlo Go. *ICGA Journal*, 32(4):221–227, 2009.

[54] Peter D. Drake and Steve Uurtamo. Heuristics in Monte Carlo Go. In *Proc. Int. Conf. Artif. Intell.*, pages 171–175, Las Vegas, Nevada, 2007.

[55] Sean C Duncan. Mandatory Upgrades: The Evolving Mechanics and Theme of Android: Netrunner. In *Well Played Summit*, 2014.

[56] Andrew Dyce. Alien: Isolation' Devs Claim Their Xenomorph Is Almost Sentient', 2013.

[57] Hilmar Finnsson and Yngvi Björnsson. Simulation-Based Approach to General Game Playing. In *Proc. Assoc. Adv. Artif. Intell.*, pages 259–264, Chicago, Illinois, 2008.

[58] Ian Frank and David Basin. Search in games with incomplete information: a case study using Bridge card play. *Artif. Intell.*, 100(1-2):87–123, 1998.

[59] Michael Freed, Travis Bear, Herrick Goldman, Geoffrey Hyatt, Paul Reber, and Joshua Tauber. Towards more human-like computer opponents. *Papers from 2000 AAAI Spring Symposium, Artificial Intelligence and Interactive Entertainment*, (1):22–26, 2000.

[60] Ada Wai Chee Fu, Renfew Wang-wai Kwong, and Jian Tang. Mining N-most Interesting Itemsets. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS),*, 2000.

[61] Gartner. Gartner Says Worldwide Video Game Market to Total $93 Billion in 2013.

[62] Sylvain Gelly and David Silver. Combining Online and Offline Knowledge in UCT. In *Proc. 24th Annu. Int. Conf. Mach. Learn.*, pages 273–280, Corvalis, Oregon, 2007. ACM.

[63] Sylvain Gelly and David Silver. Achieving Master Level Play in 9 x 9 Computer Go. In *Proc. Assoc. Adv. Artif. Intell.*, volume 1, pages 1537–1540, Chicago, Illinois, 2008.

[64] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.*, 175(11):1856–1875, jul 2011.

[65] Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *Proc. Adv. Neur. Inform. Process. Syst.*, Vancouver, Canada, 2006.

[66] Matthew L. Ginsberg. GIB: Imperfect Information in a Computationally Challenging Game. *J. Artif. Intell. Res.*, 14:303–358, 2001.

[67] Richard Guy. *Problem Books in Mathematics (Book 1)*. 2004.

[68] Thomas Hauk, Michael Buro, and Jonathan Schaeffer. Rediscovering *-Minimax Search. In *Proc. Comput. and Games, LNCS 3846*, volume 3846, pages 35–50, Ramat-Gan, Israel, 2004.

[69] Philip Hingston and Senior Member. A Turing Test for Computer Game Bots.pdf. 1(3):169–186, 2009.

[70] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining — a general survey and comparison. *ACM SIGKDD Explorations Newsletter*, 2(1):58–64, 2000.

[71] Jochen Hipp, Andreas Myka, Rudiger Wirth, and Ulrich Guntzer. A New Algorithm for Faster Mining of Generalized Association Rules. In *Principles of Data Mining and Knowledge Discovery*, pages 74–82. 2006.

[72] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[73] Feng-Hsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton Univ. Press, Princeton, New Jersey, 2004.

[74] Z. Hu, W.N. Chin, and M. Takeichi. Calculating a new data mining algorithm for market basket analysis. *Practical aspects of declarative languages: second International Workshop, PADL 2000, Boston, MA, USA, January 17-18, 2000: proceedings*, pages 169–185, 2000.

[75] Robin Hunicke and Vernell Chapman. AI for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI . . .* , pages 91–96, 2004.

[76] Kokolo Ikeda and Simon Viennot. Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 145–152, Niagara Falls, Ontario, Canada, 2013.

[77] Jaist. 2011 JAIST Cup Game Algorithm Competitions.

[78] JAIST. JAIST Cup 2012 Game Algorithm Competitions.

[79] J.Han, J.Pei, M.Kamber. *Data Mining: Concepts and Techniques*, volume 3. 2012.

[80] MB Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. *Masters Abstracts International*, 2007.

[81] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*.

[82] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Euro. Conf. Mach. Learn.*, pages 282–293, Berlin, Germany, 2006. Springer.

[83] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved Monte-Carlo Search. Technical Report 1, Univ. Tartu, Estonia, 2006.

[84] Raph Koster. *A Theory of Fun for Game Design*. 2004.

[85] Tomas Kozelek. *Methods of MCTS and the game Arimaa*. M.s. thesis, Charles Univ., Prague, 2009.

[86] David M. Kreps and Robert Wilson. Reputation and imperfect information. *Journal of Economic Theory*, 27(2):253–279, 1982.

[87] Chang-Shing Lee, Martin Müller, and Olivier Teytaud. Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go. *IEEE Trans. Comp. Intell. AI Games*, 2(4):225–228, dec 2010.

[88] Chang-Shing Lee, Mei-Hui Wang, Guillaume Maurice Jean-Bernard Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Trans. Comp. Intell. AI Games*, 1(1):73–89, 2009.

[89] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132, 1999.

[90] Jeffrey R. Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In *Proc. Assoc. Adv. Artif. Intell.*, pages 134–140, Atlanta, Georgia, 2010.

[91] Michael & Mateas and Phoebe Sengers. Narrative Intelligence. *AAAI Fall Symposium on Narrative Intelligence*, pages 1–10, 1999.

[92] Jean Méhat and Tristan Cazenave. A Parallel General Game Player. *Künstliche Intelligenz*, 25(1):43–47, 2011.

[93] Jean Méhat and Tristan Cazenave. Tree Parallelization of Ary on a Cluster. In *Proc. Int. Joint Conf. Artif. Intell.*, pages 39–43, Barcelona, Spain, 2011.

[94] Microsoft.com. What is Cortana, 2016.

[95] E.R. Miranda and J.A. Biles. *Evolutionary Computer Music*. 2007.

[96] Olana Missura and G Thomas. Player Modeling for Intelligent Di ffi culty Adjustment. pages 197–211, 2009.

[97] J Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[98] Sze Chung Ngan, Tsang Lam, Raymond Chi Wing Wong, and Ada Wai Chee Fu. Mining N-most interesting itemsets without support threshold by the COFI-tree. *International Journal of Business Intelligence and Data Mining*, 1(1):88, 2005.

[99] J. A. M. Nijssen and Mark H. M. Winands. Enhancements for Multi-Player Monte-Carlo Tree Search. In *Proc. Comput. and Games, LNCS 6515*, pages 238–249, Kanazawa, Japan, 2011.

[100] J. A. M. Nijssen and Mark H. M. Winands. Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard. *IEEE Trans. Comp. Intell. AI Games*, 4(4):282–294, dec 2012.

[101] Pierre Perick, David L. St-Pierre, Francis Maes, and Damien Ernst. Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 242–249, Granada, Spain, 2012.

[102] Marc Ponsen, Geert Gerritsen, and Guillaume Maurice Jean-Bernard Chaslot. Integrating Opponent Models with Monte-Carlo Tree Search in Poker. In *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, pages 37–42, Atlanta, Georgia, 2010.

[103] Marc Ponsen, Geert Gerritsen, and Guillaume Maurice Jean-Bernard Chaslot. Integrating Opponent Models with Monte-Carlo Tree Search in Poker. In *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, number February, pages 37–42, 2010.

[104] William Poundstone. *Prisoner's dilemma: John von Neumann, game theory and the puzzle of the bomb*. 1993.

[105] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 234–241, Granada, Spain, 2012.

[106] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 81–88, Niagara Falls, Ontario, Canada, 2013.

[107] Rob Price. Microsoft is deleting its AI chatbot's incredibly racist tweets, 2016.

[108] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On Adversarial Search Spaces and Sampling-Based Planning. In *Proc. 20th Int. Conf. Automat. Plan. Sched.*, pages 242–245, Toronto, Canada, 2010.

[109] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On the Behavior of UCT in Synthetic Search Spaces. In *Proc. 21st Int. Conf. Automat. Plan. Sched.*, Freiburg, Germany, 2011.

[110] Carlos Ramos, Juan Carlos Augusto Wrede, and Daniel G. Shapiro. Ambient IntelligenceThe Next Step for Artificial Intelligence. *Intelligent Systems, IEEE*, 23(2):15–18, 2008.

[111] Mark Richards and Eyal Amir. Opponent Modeling in Scrabble. In *Proc. 20th Int. Joint Conf. Artif. Intell.*, pages 1482–1487, Hyderabad, India, 2007.

[112] David Robles, Philipp Rohlfshagen, and Simon M. Lucas. Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 305–312, Seoul, South Korea, 2011.

[113] Kamil Rocki and Reiji Suda. Massively Parallel Monte Carlo Tree Search.

[114] Kamil Rocki and Reiji Suda. Large-scale parallel Monte Carlo tree search on GPU. *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 2034–2037, 2011.

[115] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artif. Intell.*, 175(5-6):958–987, apr 2011.

[116] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, third edition, 2009.

[117] Ayse Pinar Saygin, Ilyas Cicekli, and Varol Akman. Turing Test: 50 Years Later. *Studies in Cognitive Systems*, 30:23–78, 2003.

[118] Jonathan Schaeffer and Darse Billings. Learning to Play Strong Poker. ... *on Machine Learning* ..., pages 1–9, 1999.

[119] Jan Schäfer. *The UCT Algorithm Applied to Games with Imperfect Information*. Diploma thesis, Otto-Von-Guericke Univ. Magdeburg, Germany, 2008.

[120] Jürgen Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (19902010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, sep 2010.

[121] Nick Sephton, Peter I. Cowling, Sam Devlin, Victoria J. Hodge, and Nicholas H. Slaven. Using Association Rule Mining to Predict Opponent Deck Content in Android: Netrunner. *Proceedings of IEEE Computational Intelligence and Games Conference (CIG 2016)*, pages 102–109, 2016.

[122] Nick Sephton, Peter I Cowling, Edward Powley, and Nicholas H Slaven. Heuristic Move Pruning in Monte Carlo Tree Search for the Strategic Card Game Lords of War. In *IEEE Conference on Computational Intelligence and Games*, 2014.

[123] Nick Sephton, Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Parallelization of Information Set Monte Carlo tree search. In *IEEE Congress on Evolutionary Computation*, 2014.

[124] Nick Sephton, Peter I. Cowling, and Nicholas H. Slaven. An experimental study of action selection mechanisms to create an entertaining opponent. *2015 IEEE Conference on Computational Intelligence and Games, CIG 2015 - Proceedings*, pages 122–129, 2015.

[125] Noor Shaker and Julian Togelius. The Turing Test Track of the 2012 Mario AI Championship: Entries and Evaluation. *Proceedings of the . . .*, 2013.

[126] Brian Sheppard. World-championship-caliber Scrabble. *Artif. Intell.*, 134:241–275, 2002.

[127] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam,

Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[128] SB Skelton. Gaming as Culture: Essays on Reality, Identity and Experience in Fantasy Games. *The Journal of Popular Culture*, 2008.

[129] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' Bluff: Opponent Modelling in Poker. In *Proceedings of the TwentyFirst Conference on Uncertaintyin Artificial Intelligence UAI*, pages 550–558, 2005.

[130] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[131] Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Online adaptation of game opponent AI in simulation and in practice. *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)*, pages 93–100, 2003.

[132] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-ning Tan. Web usage mining: discovery and applications of usage patterns from Web data. 1(2):12–23, 2000.

[133] Jan A. Stankiewicz, Mark H. M. Winands, and Jos W. H. M. Uiterwijk. Monte-Carlo Tree Search Enhancements for Havannah. In *Proc. 13th Int. Conf. Adv. Comput. Games, LNCS 7168*, pages 60–71, Tilburg, The Netherlands, 2012.

[134] James P. Suttle and Daniel A. Jones. Poker game, 1989.

[135] Penelope Sweetser and Peta Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment*, 3(3):1–24, 2005.

[136] Gabriel Synnaeve and Pierre Bessière. A Bayesian model for opening prediction in RTS games with application to StarCraft. *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pages 281–288, 2011.

[137] Mandy J. W. Tak, Mark H. M. Winands, and Yngvi Björnsson. N-Grams and the Last-Good-Reply Policy Applied in General Game Playing. *IEEE Trans. Comp. Intell. AI Games*, 4(2):73–83, 2012.

[138] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Association Analysis: Basic Concepts and Algorithms. *Introduction to Data mining*, pages 327–414, 2005.

[139] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N. Yannakakis. The Mario AI Championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.

[140] Alan M Turing. COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, 59:433–460, 1950.

[141] Daniel Whitehouse. Monte Carlo Tree Search for games with Hidden Information and Uncertainty. 2014.

[142] Daniel Whitehouse, Peter I. Cowling, Edward J. Powley, and Jeff Rollason. Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. In *Proc. Artif. Intell. Interact. Digital Entert. Conf.*, pages 100–106, Boston, Massachusetts, 2013.

[143] Daniel Whitehouse, Edward J. Powley, and Peter I. Cowling. Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu. In *Proc. IEEE Conf. Comput. Intell. Games*, pages 87–94, Seoul, South Korea, 2011.

[144] John R. Koza William B. Langdon, Riccardo Poli, Nicholas F. McPhee. *Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications*. 2008.

[145] MHM Winands. The relative history heuristic. *Computers and . . .* , 2006.

[146] Wizards of the Coast. Magic: The Gathering.

[147] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret Minimization in Games with Incomplete Information. In *Proc. Adv. Neur. Inform. Process. Sys.*, pages 1729–1736, Vancouver, Canada, 2008.