SCOUT: A Case-Based Reasoning Agent For Playing Race For The Galaxy

Michael Woolford
mwoo119@aucklanduni.ac.nz
UoA ID: 4713606

Supervisor: Associate-Professor Ian Watson

A dissertation submitted in partial fulfilment of the requirements for the degree of Bachelor of Science (Honours) in Computer Science, University of Auckland, 2016

Abstract

Game AI is a well-established area of research. Classic strategy board games such as Chess and Go have been the subject of AI research for several decades, and more recently modern computer games have come to be seen as a valuable testbed for AI methods and technologies. Modern board games, in particular those known as German-Style Board Games or Eurogames, are an interesting mid-point between these fields in terms of domain complexity, but AI research in this area is more sparse. This dissertation discusses the design, development and performance of a game-playing agent which uses the Case-Based Reasoning methodology as a means to reason and make decisions about game states in the Eurogame Race For The Galaxy. We have named this system SCOUT. The purpose of this research is to explore the possibilities and limitations of Case-Based Reasoning within the domain of Race For The Galaxy and Eurogames in general.

Contents

Chapter 1

Introduction

1.1 Game AI, Modern Board Games, and Race For The Galaxy

Historically, the most prominent examples of game AI research have focussed on achieving and exceeding human skill levels of performance in classic board games [9, 30, 31], while others have used those games as a testbed for experimenting with specific technologies and methodologies, or within the bounds of various limitations such as avoiding utilising domain knowledge [32, 11, 23, 29].

Laird argued that despite impressive successes in their specific domains, this research had done little to progress the field towards development of a general human-level AI, and that modern computer games of many different genres, including computer strategy games, provided a superior testbed for human-level AI [19].

Race For The Galaxy (RFTG) falls into a category of modern board games known as Eurogames. These games typically involve more complex rule sets than traditional card and board games, have mixtures of hidden and open information and deterministic and stochastic elements, and are less abstract. Because of this, they bear more similarities to computer strategy games than do traditional board games. In recent years several agents for playing various Eurogames have been developed, both in academic contexts [13, 17, 36] and otherwise [18]. In general the approach to creating these agents has been more in keeping with the approaches taken in classic strategy board game AI systems. In contrast, a key aim of this project is to train SCOUT from examples of games played by a human player, through the Case-Based Reasoning methodology. Our hope is that if SCOUT can successfully mimic the decisions made by a human player, then it can implicitly gain some of the benefit of the human's strategic reasoning, and demonstrate a style of playing the game which resembles that of the human. Of course the primary goal of playing a game like RFTG is to win, so that remains our main focus in terms of results, but we would also like to observe the way in which SCOUT goes about winning, and try to encourage diverse and adaptive play styles. Additionally, where possible we aim to limit our use of RFTG domain knowledge in developing the system, with an eye toward exploring methods which could be generalised to other Eurogames.

1.2 Case-Based Reasoning and k-Nearest Neighbour

Case-Based Reasoning (CBR) is a methodology [38] in which a system adapts and reuses solutions to past problems in solving new problems [26]. The essential principle behind it is the assumption that similar problems should have similar solutions, and that a problem can be captured and stored as a *case*, which describes the features of the problem and its solution. Features in a case may be *indexed* or *unindexed*, meaining they are either used in terms of measuringsimilarity or unused. A case-based reasoner will explore its case base of previous problems and compare them in terms of the current problem situation, and utilise them in what is classically described as the 4-stage CBR Cycle, described by Aamodt and Plaza as follows [2]:

1. RETRIEVE the most similar case or cases
2. REUSE the information and knowledge in that case to solve the problem
3. REVISE the proposed solution
4. RETAIN the parts of this experience likely to be useful for future problem solving

The most common algorithm for retrieving cases, and the one which SCOUT utilises, is the k-Nearest Neighbour (k-NN) algorithm [40]. The k-NN algorithm examines the case base in its entirety for every problem and identifies those k cases which have the smallest Euclidean distance to the current problem situation in an n-dimensional space, were n is the number of indexed features in the cases.

The distance between a problem P and case C is defined as

$$dist(P, C) = \sum_{i=1}^{n} d(P_i, C_i) \times w_i \tag{1.1}$$

where each feature has a unique distance function $d(P_i, C_i)$ and w_i is a predefined weighting for that feature.

SCOUT's system implements the full CBR cycle, although the primary focus is on the Retrieval and Reuse stages, as these are the key stages which impact SCOUT's ability to play RFTG. The final version of SCOUT includes the capacity to revise and retain cases and to perform case base maintenance This is shown to give some limited performance gains over time.

Chapter 2

Game AI Literature Review

2.1 Classic Board Games

2.1.1 Chess

Computer Chess is one of the classic examples in game AI, particularly Deep Thought and Deep Blue [9]. Deep Thought was the first chess program to win a tournament match against a Grandmaster, in 1988. Deep Thought utilised specialised hardware and relied primarily upon high search speeds, evaluating 700,000 positions per second. From 1989, Deep Thought 2 was developed and improved upon its predecessor with multiprocessing, a more sophisticated evaluation function, new search software, and a larger move database. This machine served as a prototype for Deep Blue I, which played World Champion Garry Kasparov in 1996, by this stage with the power to search 50-100 million positions per second. Deep Blue was defeated 2-4, but a scaled-down version of Deep Blue did defeat other Grandmasters in preparation matches. The next iteration of the machine Deep Blue II, and improvements again focussed on increasing raw search speed and adding further sophistication to the evaluation function. Deep Blue II defeated Kasparov in their 1997 match by 3.5-2.5, a landmark event in the history of computation. Despite its success, there were still many improvements that could be made upon Deep Blue. Notably, Deep Blue did not significantly utilise pruning mechanisms to narrow the search space. While Deep Blue was a dedicated supercomputer, several chess programs are now able to achieve superhuman ability on only basic PC hardware.

Around the same time as Deep Blue II's success, Sinclair explored an approach using CBR, in an attempt to avoid excessive use of search algorithms [32]. As a motivating factor, Sinclair cited psychological research into human chess players' reasoning. Grandmaster players were not found to explore significantly more branches than amateur-level players. Also, grandmaster players would recognise familiar patterns in a board state and draw upon memories of previous examples of such patterns to provide clues as to appropriate moves. Using a knowledge base of 16,728 grandmaster games, the program attempted to predict the moves from a further 331 games. Board states were expressed as 11-dimensional vectors, and the results showed that good predictions were made only when of all of the dimensions matched. Increasing the tolerance for dissimilarity increased the retrieval rate but reduced the quality of predictions.

2.1.2 Checkers

A simpler game than Chess, Checkers is now a weakly solved game, as Schaeffer determined the strategy perfect-play from the start, which will yield a draw result [30]. A weakly solved game is one where the strategy for perfect-play is known from the start position, as opposed to a strongly solved game, where the perfect-play strategy is known from given position. The proof utilised two different searches, a backward search which enumerated every endgame position with fewer than 11 pieces on the board, a total of 39,271,258,813,439 different positions, and a forward search which evaluated and proved the outcome of moves from the opening to the endgame database. Additionally, expert knowledge was used to initially guide the search, thereby reducing the search space and enhancing performance. The solving process took 18 years of continuous calculation on multiple computers.

The completed proof was an extension of the Chinook program, which had been playing Checkers competitively since 1989. In 1990 Chinook had competed and achieved second place in the U.S. National Open, thereby earning the right to challenge the World Champion, Marion Tinsley, to a title match. The human player narrowly won in 1992, and a 1994 rematch with an improved iteration of Chinook had to be abandoned. By 1996, however, 11 years before the proven solution was published, Chinook was already established as stronger than all human players.

Like Deep Blue, Chinook's strength lay in its raw search power, large endgame database, and sophisticated evaluation function, entirely hand-crafted using expert knowledge. Other approaches utilised Checkers as a testing ground for alternative AI approaches that did not require expert knowledge or intensive computational power. Fogel developed a system using evolutionary neural networks and novice-level domain knowledge, which could, after 250 generations of evolution, eventually compete against skilled and even master human players [11]. Powell utilised an unconventional form of CBR in a system called CHEBR, which eschewed the usual process of manual knowledge-base construction from domain experts [23]. This was replaced with automatic case elicitation, whereby an initially empty knowledge base was populated by playing the program against itself and retaining records of successful moves. Because the program began with no prior knowledge of what constituted a good move or even a valid move, early cases were built entirely by trial-and-error. Through a case rating system the agent eventually learned to discriminate between valid and invalid moves, and then between effectual and ineffectual moves. Powell concluded that this process was a valid substitute for a hand-crafted knowledge base.

2.1.3 Go

Go is considered the most difficult of the classic strategy games in AI research, and recent developments have come in the form of AlphaGo, which became the first program to defeat a professional human player in 2015, and then the first to defeat a 9 dan ranked player in 2016 [31]. Prior to AlphaGo, the most recent major development in computer Go was the introduction of Monte Carlo Tree Search, and the strongest programs narrowed their search spaces using prediction policies trained on human expert moves. Preceding this development, relatively weak Go programs relied on temporal-difference learning. AlphaGo's advantage was to combine MCTS with deep neural networks for board evaluation and move selection. These networks allowed for deeper evaluation of board state and move, although they were still trained initially from human expert moves. The neural networks were trained through a pipeline of training stages. The first stage of the pipeline was to train directly from a database of 30 million human moves. Once trained, the policy network could predict expert human moves for a given state with 57.0% success. The network was then trained in the next stage using reinforcement learning to translate the ability to predict expert moves into the ability to predict winning moves. The use of human

expert moves to initially train the policy network meant that the network could learn high quality moves quickly and efficiently.

The MCTS rollouts used by AlphaGo to combine the policy and value networks were relatively simple compared to other top Go programs, placing emphasis on the depth of the evaluation functions. The performance gained from this approach was astounding: in a tournament of 495 games against other state-of-the-art Go programs, AlphaGo won 494. The creators argued that AlphaGo is a closer approximation to how humans play games than is Deep Blue, as it evaluates thousand of times fewer positions, but does so more intelligently.

2.2 Poker

Poker poses different challenges to researchers than do classical strategy board games, and has gained more attention since the successes of Deep Blue and Chinook. Poker differs from those games in that it is stochastic and is played with imperfect information. These characteristics reduce the feasibility of brute force search.

In 2015, despite these challenges, one form of poker, heads-up limit hold 'em, became the first imperfect information game to be weakly solved, with caveats considering the stochastic nature of the game by Bowling, et al [7]. The authors proved that this form of poker is a winning game for the dealer, by utilising a variant of counterfactual regret minimization, which was a method used since 2007 to solve a series of increasingly complex simplifications of this form of poker. This approach is based in game theory, whereby the Nash equilibrium is approximated through repeated self-play.

Other forms of poker, such as no limit and poker with more than two players, remain open problems. Rubin and Watson identified three broad categories for approaching poker problems: game theoretic solutions; heuristic rule-based systems; and simulation/enumeration-based approaches, analogous to search in perfect information games [29]. In 2002 the program Poki was demonstrated to perform well using both of the former approaches in both lower- and higher limit games. The program CASPER used a CBR approach to betting [28]. It constructed a target case based on the current state of the game and then attempted to retrieve a matching case from a specific case base for the given stage of the hand. It also utilised domain knowledge such as a table ranking all 169 different opening hand configurations. CASPER's performance was found to be strongly dependent upon the nature of its opponent, for instance it was found to perform well against the aforementioned Poki, and less effectively against human opponents for real money.

2.3 Modern Computer Games

Laird emphasised the distinction between AI for traditional board games, such as Chess and Go, and AI for interactive computer games [19]. He argued that although most AI research of the preceding 50 years has been applied to such traditional games as Chess, Go, Checkers et cetera, and with great success, this research has been too narrow in its scope, by focussing on only a few human capabilities, and has done little to advance AI to human-levels in the general sense. Laird proposed that modern computer games provided a superior testbed for human-level AI, due to the richness of the environments in which the agents would be required to operate. Buro further emphasised Real Time Strategy (RTS) computer games as a particular field of interest for AI research [8]. Buro pointed out that a key distinction between these and chess and checkers were that individual actions in RTS games rarely have immediate global effects, meaning that

human abilities to reason abstractly and generalize enabled them to easily outperform searching computers.

2.4 Eurogames and Other Modern Board Games

Eurogames provide an interesting midpoint between the above domains, especially when considering computer games of the strategy genre. In a Eurogame there are frequently several possible avenues to victory by exploiting different aspects of the game mechanics, and individual actions often do not have immediately obvious impacts on the game outcome. This tends to bring them more in line with strategy computer games. By contrast, Eurogames are still highly abstract and do not require the engagement with a "virtual world" that Laird felt gave computer games some of their value. An interesting aspect of these games is that their stochastic nature and particularly their indirect player interaction tend to de-emphasises the need for optimal play. Although many Eurogames, particularly RFTG, require high degrees of skill to play consistently well, it is still quite possible for satisfying games to be played between players of unbalanced skill levels. This opens the possibility to emphasise varied human-like play, as opposed to fixating on performance gains.

Previous academic research into creating programs to play Eurogames is a limited but growing area.

2.4.1 Settlers of Catan, Dominion, Carcassone, Magic: The Gathering

The most popular Eurogame is Settlers of Catan, for which several AI programs exist. Szita, et al, successfully applied MCTS to the game and found that its performance greatly exceeded the common rule-based approach [36].

Fynbo and Nellemann developed an agent capable of playing Dominion with some success [13]. Dominion is comparable to RFTG in that it is a stochastic card game with an extremely large state space, and much of the game's strategic element is based in evaluating the relative quality of cards in the context of the current game state. Their approach involved dividing the game into three parts, and applying different AI techniques to each, but primarily focussing on using artificial neural networks. For two parts the agent was found to be very effective, whereas for the third it was found that a simple rule-based approach was more effective.

Heyden developed an agent to play the game Carcassone [17]. Carcassone is less similar to RFTG than Dominion, but is still broadly comparable to it in type and complexity. Heyden used a search-based approach to the game, comparing MCTS and a variant of minimax search. In that case, the minimax implementation with the Star2.5 pruning technique was shown to greatly exceed the performance of the MCTS implementation. Furthermore, the program utilising the minimax search was shown to be able to beat an advanced human player, although the breadth of the experimental results was limited to only a few games.

Another modern game which has gained some attention from AI researchers is Magic: The Gathering (MTG). MTG is not a Eurogame, but it bears some similarities to them, especially RFTG. A commercial game, Duels of the Planeswalkers uses a simple AI agent which can play the full game at beginner level, and other projects also exist which use various approaches such as minimax and simple rules. None of these AI's are regarded as providing challenging opponents for human players beyond the beginner level [37]. Cowling and Ward developed a system using Monte Carlo Tree Search for a simplified version of the game, and demonstrated that even a limited MCTS rollout was able to improve the agent's performance compared to a randomised agent [37]. They concluded that the addition of MCTS to a rule-based program significantly

improved performance. Unfortunately their conclusions could only be applied to their simplified version of the game, which was tailored to work well with MCTS.

2.4.2 Race For The Galaxy

Finally, most relevant to this dissertation, Jones produced an open source implementation of RFTG, and created an AI agent which utilises artificial neural networks to evaluate game states and predict opponent moves [18]. Unfortunately this was not accompanied by any academic report to provide any insight into the development. It demonstrates that an adequate AI agent for RFTG is feasible, and provides the basis for my own research.

Chapter 3

Race For The Galaxy

Race For The Galaxy [21] is a popular Eurogame in which players attempt to build the best empire by constructing a tableau of cards. The game is highly stochastic as the game progresses as cards are randomly drawn from the deck, lending a high degree of variety and unpredictability to gameplay, but player actions are resolved deterministically, resulting in a richly strategic and skilful game. RFTG has several expansions which increase the complexity of the game further, and can be played by up to 6 players. Currently, SCOUT is designed to be played only in a 2-player game without expansions, the basic game.

The rules are significantly more complex than classic board games such as Chess, and a basic overview of these is necessary to discuss it further. This section aims to be a sufficient description of the game rules to understand the remainder of this dissertation, and therefore simplifies or omits certain aspects of the game for clarity. A rigorous description of the rules is available at the publisher's website [14].

3.1 Basic Game Elements

The basic elements of the game consist of:

- **Cards**, of which each card is either a **World** or a **Development**.
- A **deck** of cards, from which all players draw.
- Each player's **hand**, consisting of cards which are drawn from the deck.
- Each players **tableau**, consisting of cards which are placed from their hand.
- Each players collection of **action cards**.
- A **discard pile**, consisting of all cards discarded by players.
- A **pool** of **victory point chips**, from which the players may acquire chips.

The game is structured into **rounds**, which are themselves structured into a 1-5 **phases**, and a beginning and an end. At the beginning of each round, each player secretly selects and then simultaneously reveals one of their 7 possible **action cards**. Each possible action card indicates a phase which will occur in the round, as well as a bonus which the selecting player will receive. During the remainder of the round, every player will take actions in every phase which has been

selected by every player, and each player will receive their individual bonus as per their selected action card. At the end of the round each player discards any cards in hand in excess of 10, and the next round begins in exactly the same fashion. This continues until one of the conditions for the game to end is met, at which point the player with the most **victory points** is declared the winner. This comprises the entirety of that game, apart from the start of the game, where each player is given a single **start world** as the first card in their tableau and their opening hand of 4 cards.

3.1.1 Phases and Action Cards

The possible phases of a round, in the order in which they take place if selected, are as follows:

Explore: Each player draws 2 cards, and then discards one and keeps the other in their hand.

Develop: Each player may choose to place a single Development card from their hand to their tableau.

Settle: Each player may choose to place a single World card from their hand to their tableau.

Consume: Each player must consume any goods on their tableau with any consume powers in their tableau.

Produce: Each player must place goods on any cards in their tableau which produce goods.

The possible action cards are as follows:

Explore +1/+1: Explore phase selected, and the selecting player may draw and keep an additional card.

Explore +5: Explore phase selected, and the selecting player may draw 5 additional cards, but still keep only one.

Develop: Develop phase selected, and the selecting player receives a discount of 1 to the cost of any Development they place.

Settle: Settle phase selected, and the selecting player may draw and keep a card at the end of the Settle phase if they place a World.

Consume-Trade: Consume phase selected, and the selecting player must consume one of their goods in exchange for a fixed number of cards.

Consume-x2: Consume phase selected, and the selecting player doubles any victory points which they receive due to their consume powers.

Produce: Produce phase selected, and the selecting player my place a good on one of their Windfall worlds, which are cards specified to not normally produce goods but to do so in this case.

3.1.2 Placement

The two phases with by far the most depth of play (excluding phase selection itself) are the Develop and Settle phases, during which placement of Development or World cards take place respectively. Placement involves selecting a card from hand and placing it in the tableau, while also discarding a set number of other cards as indicated by the placed card's cost. Certain cards, **Military Worlds**, do not require any cards as payment but require the player to have a military score greater than a certain value. Once a card is placed, any actions which are indicated on its face are then able to be performed by its owner during all relevant phases which subsequently occur. This is the manner in which a tableau is built, and this comprises the bulk of the game's action and strategy. A player may also choose to pass on placing a card in order to save the cards in their hand for later.

3.1.3 Goods

There are 4 types of goods, each of which are produced by different types of worlds and have different value when traded for cards. Once produced, goods exist on the tableau alongside the cards. These goods may be consumed via actions indicated on cards in a player's tableau. Deciding when to select a Consume- action card to use these goods is an important and difficult strategic consideration, but in general once that decision is made the decision of how to use each of them is trivial.

3.1.4 Game End

The game ends at the end of a round in which either:

(a) A player places their 12th card into their tableau.

(b) The pool of victory point chips is emptied.

At this point, each player sums their score of total victory points, which consists of the number of victory point chips plus the sum of points which each card in their tableau is worth. The player with the highest score wins.

3.2 Strategy

In chapter 7 we will discuss how SCOUT's style of play compares to that of a human player. This will require a discussion of strategic considerations that a human player would make. These are outlined here. These concepts are adapted from those commonly understood and discussed by the broader community of RFTG players, for instance at *boardgamegeek.com* [6].

3.2.1 Strategy Archetypes

There is a great deal of unknown information at the outset of a game of RFTG, and to a large extent success depends on being aware of the possibilities which lie ahead, which a player typically learns through experience over many games. Making decisions based solely on the information currently available, even if maximally optimal based on that information, is heavily reliant on luck and will only lead to intermediate levels of success. This is essentially how the current standard of RFTG AI plays: optimally within the bounds of the open information. In contrast, skilled human players typically aim to play towards a particular strategic plan based on their

start world, opening hand, and knowledge of their opponents, and adapt that plan as necessary based on what cards become available and what the opponents are doing. Success in a high skill match is dependant on being able to adapt these plans well to the current game situation. Emphasis is put on building "coherrent" tableaux with cards which will compliment eachother for exponentially increasing rewards later in the game. Furthermore, there are a variety of cards which are generally useful in many situations, and other cards which are very useful in specific situations but useless in others. Because of this, there are a vast number of ways to play and build a tableau but most fall within a few archetypal categories:

Develop strategies which rely upon the player using particular cards to frequently place expensive and rewarding Developments and complete their tableau quickly;

Military strategies which involve the player placing Worlds without paying cards for them, and acquiring new cards by trading goods;

Consume-Produce strategies which involve building an "engine" of a few production worlds and cards with good consume powers and repeatedly playing the Consume-x2 action to rapidly deplete the pool of victory point chips. In the basic game, Consume-Produce is typically the most effective strategy and the most difficult for new players to grasp.

3.2.2 Stages of the Game

Play typically also has an early, mid, and late game, although much like chess the exact bounds of these stages are fluid. Strategic considerations for each of these stages are distinct: In the early game a player would likely favour gaining more cards so as to give themselves more strategic options; In the mid game players will develop their plans, either trying to accelerate toward the late game or building their capacity to acquire large amounts of points; In the late game, where it becomes possible to predict the moment at which the game will end, players begin to focus on quickly acquiring points at all costs, beginning to place overpriced cards and discarding cards which would otherwise be beneficial but cannot be played in time. Being able to evaluate the stage of the game is another important element of skilled play.

3.3 Computer RFTG

Multiple computer implementations of RFTG have been developed for online play, although before SCOUT only a single AI system has been developed. Currently the most commonly used game engine is that hosted by the site boardgamearena.com [4], while our work was done with an offline open source game engine developed by Keldon Jones, which also has an integrated AI system [18]. In terms of its reasoning processes, SCOUT is designed to function largely independently of the game engine with which it is playing RFTG, and could be implemented to work with any game engine. Henceforth we will use "RFTG game engine" when referring to this part of the system in general, and "Keldon game engine" when referring to the specific engine used in our implementation.

3.3.1 Keldon AI

The AI system integrated in Keldon Jones' RFTG engine is the system which we used as an opponent and benchmark for SCOUT, and henceforth will be referred to as the Keldon AI. The Keldon AI is sophisticated by the standards of popular game AI and plays the game competently at an intermediate level. It is generally outplayed by a skilled human player but is able to win

with favourable draws, and it will regularly beat a novice human player. Unfortunately, the system is undocumented apart from a few source code annotations, and Jones is no longer active in the online community, so the following explanations are derived from our examination of the system behaviour and the code itself. It uses a combined approach of an expert system and search, along with neural networks for predicting opponents' phase selection and for evaluating positions. Essentially the process is as follows:

1. If the current decision is for phase selection, then the system will attempt to guess what phase the opponent will select, and then assume that phase is selected for the rest of the search.

2. For certain decision types it will limit its options based on rules derived from domain expertise.

3. The system will simulate games (with a randomised deck to avoid cheating by gaining secret information) and enumerate every possible choice that it can make, along with every other possible choice that will be led to later in the turn, and evaluate the positions which can be reached at the end of the turn.

4. The system chooses the option considered most likely to lead to a victory.

The evaluations are performed by using a neural network to determine how likely is the evaluated state to lead to a victory, and the network is trained by the AI playing a number of games against itself. Due to the game being stochastic and having large amounts of hidden information, this search is inherently limited, and the search will only look ahead to the end of the current turn.

Figure 3.1: *A game of RFTG in progress in the Keldon engine [18], with the Human player close to defeating the Keldon AI with a military strategy. Card images property of Rio Grande Games [15]*

1. *Player Tableau, of 9 cards*
2. *Opponent Tableau, of 6 cards*
3. *Player Hand, The first 3 cards are Worlds, the 4th is a Development*
4. *Player Action Cards*
5. *(left to right) Player hand size, number of victory chips and total victory points, military score*

Chapter 4

SCOUT

The objective of developing SCOUT was to create a system which could, given any RFTG game state, use Case-Based Reasoning to select a single good move for that game state, and as such play an entire game using only past games as guidance. In its current state, this objective has largely been realised, although one strategically insignificant decision is deferred to the Keldon AI system, and the CBR system for phase selection is partially augmented by a simple expert system. In the former case, this is largely due to the decision being insignificant in gameplay terms while also being difficult to approach with a CBR system. In contrast, the expert system is implemented in order to address some weaknesses in SCOUT's play, and ultimately we would aim to eliminate the need for it.This is discussed in detail in the Development section, below.

SCOUT is also designed to record and evaluate its performance over many games, and add and remove cases in its case base accordingly. It can be set to play continuous runs of thousands of games, and to retain its developing case bases over multiple runs. Furthermore it treats a fixed number of games as an "era", and upon the completion of an era SCOUT performs periodic evaluations of its entire case base and overall performance, in addition to the continuous evaluation of cases as they are used. It also archives its case base as of the end of each era. This is useful during development as it allows for analysis and to allow for rollbacks if suboptimal settings lead to SCOUT's maintenance system undermining its own performance.

4.1 Basic Operation

The basic operating procedure for SCOUT during a single game is as follows:

1. The RFTG engine presents a game state to SCOUT from the perspective of the player being controlled by SCOUT.
2. SCOUT determines the type of decision which needs to be made in the current state.
3. SCOUT compares the game state to cases of states from prior successful games, which include the decision made in response to that state.
4. SCOUT selects the most appropriate case from the case base, and adapts the decision made in that case to the current game.
5. SCOUT instructs the RFTG engine to implement that decision for the player, and the RFTG engine resolves the game state and presents SCOUT with the next decision to be made.

Later, if the decisions led to a successful game outcome, SCOUT adds all states from that game, along with the decisions made, into its case bases.

This is an atypical approach for a game AI system. As discussed in the literature review, state of the art game AI typically involves searching ahead and evaluating possible moves, or MCST rollouts. SCOUT does not perform any such search or rollout nor indeed does it make any evaluation the state in game terms such as the strength of its own position or the likelihood of any particular move leading to victory. Instead it operates under the assumption that any case's presence in a case base implies that it contains one of a sequence of moves which leads to victory in certain conditions, and if those conditions are sufficiently similar to the conditions of the current game then that move could be a good move for the current game. Furthermore, when SCOUT uses cases from games won by skilled human players, it aims to implicitly take advantage of the human's evaluation of the game and expectations of future moves. If a case in the case base is a good match for the current state, this may provide advantages over the Keldon AI's evaluation functions, given that the highly stochastic nature of the game makes searching ahead an unreliable and costly endeavour.

4.2 Implementation

4.2.1 Python Modules

SCOUT is implemented as a collection of Python2.7 modules which run on the PyPy interpreter to improve performance [27], as searching the large case bases is CPU and memory intensive. The case base is a collection of XML documents. Some modifications were made in the source code of the Keldon engine, which is written in C, to enable SCOUT to integrate with it. For simplicity and independence, SCOUT exchanges information with the game engine via a socket server-client interface.

SCOUT was not developed with an eye towards efficient performance times, since it is intended to play a turn based game against human opponents. In testing, however, many thousands of games need to be run and therefore case base searches are parallelised. Time and memory cost are proportional to the size of the case base, and time is also dependent on the type of decision being made, as methods for processing the case base varies. In the current version, making a single Placement decision with the initial case base takes approximately 1 second on a midrange home desktop. This is perfectly acceptable for playing against a human opponent but is a performance bottleneck when running large numbers of games against the Keldon AI when attempting to improve the case base.

4.2.2 A SCOUT Case

A SCOUT case is stored as an XML document which specifies every known feature of the game state, along with the type of decision to be made in that state, the options available to the player, and the option selected by the player, ie the case's solution. Additionally it contains a game id shared by all cases derived from the same game. The specification of this document is strongly inspired by the specification which is used by the Keldon engine to export saved games, but has many modifications to convert it into a useful case. The case describes all information available to the player being controlled by SCOUT, but hidden information such as the cards in an opponent's hand is not described, although the size of their hand is described. The format of the cases is identical regardless of the type of decision which the case describes, but different features are indexed depending on decision type. This is further discussed in the Chapter 6. Listing 4.1 is an example of a case for a Placement decision as it appears to SCOUT. Game

states are also described with exactly the same specification, and thus SCOUT's operation is largely a matter of parsing and comparing documents of this form. This case and its meaning will be discussed further in chapter 6.

Listing 4.1: A SCOUT Case in XML form

```
<RftgStateCase>
  <Type>3</Type>
  <Choices>98.23.41.86</Choices>
  <Choice>23</Choice>
  <PlayerName>Player 0</PlayerName>
  <GameID>1457588116</GameID>
  <Status>
    <Round>6</Round>
    <Phases>
      <Phase id="3" current="yes">Develop</Phase>
      <Phase id="7">Consume</Phase>
    </Phases>
    <Deck>7687219</Deck>
    <Discard>17</Discard>
    <Pool>24</Pool>
  </Status>
  <Player id="1" ai="yes">
    <Name>Player 1</Name>
    <Actions>
      <Action id="7">Consume-Trade</Action>
    </Actions>
    <Chips>0</Chips>
    <Score>4</Score>
    <Military>0</Military>
    <Tableau count="3">
      <Card id="10" good="yes" num_goods="1">Alpha Centauri</Card>
      <Card id="57" num_goods="1">Galactic Engineers</Card>
      <Card id="15" good="yes" num_goods="1">Comet Zone</Card>
    </Tableau>
    <Hand count="3">
    </Hand>
  </Player>
  <Player id="0" ai="yes">
    <Name>Player 0</Name>
    <Actions>
      <Action id="3">Develop</Action>
    </Actions>
    <Chips>0</Chips>
    <Score>3</Score>
    <Military>0</Military>
    <Tableau count="4">
      <Card id="8">New Sparta</Card>
      <Card id="11">Rebel Fuel Cache</Card>
      <Card id="26" good="yes" num_goods="1">Spice World</Card>
      <Card id="54" num_goods="1">Pre-Sentient Race</Card>
    </Tableau>
    <Hand count="6">
      <Card id="92">Gambling World</Card>
      <Card id="33">Rebel Outpost</Card>
      <Card id="23">Space Marines</Card>
      <Card id="86">Genetics Lab</Card>
      <Card id="41">Mining Conglomerate</Card>
      <Card id="98">Galactic Imperium</Card>
    </Hand>
  </Player>
</RftgStateCase>
```

During runtime, all cases are stored in memory and committed to disk periodically or upon exit.

4.2.3 Modifications to the Keldon engine

Four major modifications were made to Keldon game engine source code. The details of these changes are included in Appendix A, and cover the client-side of the server-client interface, the ability to interchange different AI agents for controlling players, and additional testing functionality. This modified system is flexible and essentially gives the Keldon engine a modular AI system. This could save a lot of work in future attempts to develop new RFTG AI systems. The source code of the Keldon engine is large, undocumented, and cumbersome to work with, but a future system could be integrated with our modified version of the engine with ease.

Chapter 5

SCOUT Design

5.1 Overview

This chapter aims to give an overview of each of the functional elements of the latest version of SCOUT. The next chapter will detail the specific design choices and developments which lead to this structure.

SCOUT consists of a group of independent modules, each of which handles one aspect of its functionality, along with a multi-part case base. The aim of this approach is to be flexible and to facilitate easy experimentation with- and comparison of- different approaches to developing an AI system for RFTG or potentially other Eurogames. This was inspired by Molineaux and Aha's TIELT system for integrating AI systems with RTS games [22, 1].

There are 6 modules in the current iteration of SCOUT:

- The Head module
- The Interface module
- The Case Controller module
- The Placement Reasoning module
- The Phase Reasoning module
- The Payment Reasoning module

In brief, the **Head** module determines how to process incoming requests from the RFTG game engine and facilitates communication between separate modules; The **Interface** receives game information and decision requests from the game engine and translates them into the specification used by SCOUT; The **Case Controller** module organises and maintains a registry of the case bases; The **Placement**, **Phase**, and **Payment** modules each reason about the game state and make decisions when a relevant request is made by the game engine.

This model is very flexible, for example: In order to work with a different implementation of RFTG, only the Interface module would need to be modified. If we wished to try a completely new reasoning process for card placement, we could swap out the Placement module. Alternatively, if we wish to disable any part of SCOUT's reasoning system and defer back to the Keldon AI, this can be achieved with a simple switch in the Head module. Meanwhile all other parts

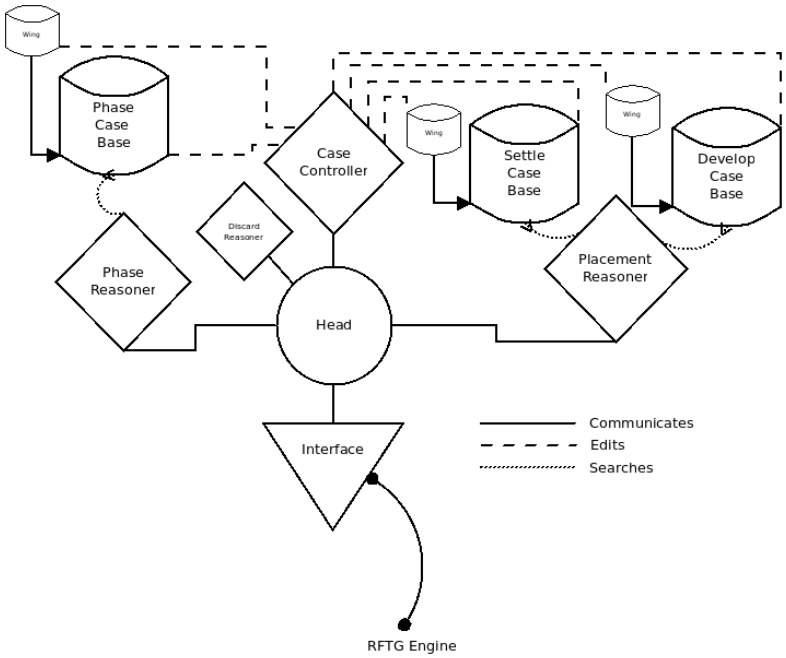

Figure 5.1: *Visibility between the modules that constitute SCOUT.*

of the system function unchanged. This is particularly useful during testing, as it allows us to measure the influence of another part of the system on SCOUT's overall performance in isolation. Modules make requests of one another via the Head but their internal processes are irrelevant to each other, particularly with regards to the reasoning modules. The Case Controller is of course specific to a CBR approach, but the Placement system, for example, could be reworked to classify a game state using a neural network while the Phase module continued to utilise the CBR system and each would still work in tandem.

5.2 Head

This is the central control module. It incorporates the entry point and main event loop of the program, and it initialises all other modules in the system, along with the RFTG game engine. The Head is the only module which communicates directly with other modules and therefore the only one which is must be compatible with other modules' implementations.

In an iteration of the main event loop, the Head requests the Interface to retrieve the current message from the RFTG engine and determines to which of two classes it belongs:

(a) If it is a request for a decision it then either switches control to the relevant reasoning module. Once the reasoning module returns a decision it sends the decision to the Interface and also sends the decision along with the game state to the Case Controller, potentially to be retained as a new case.

(b) If it is a notification of a completed game it triggers an endgame procedure which the Head handles itself. In the event of a victory for SCOUT it instructs the Case Controller to retain the decisions and states as new cases or to discard them in the event of a loss. If the

game is the last of an era, then it triggers further processing of the case base and analyses and records SCOUT's performance during that era.

In addition to this, the permanent log is written from the Head, which records information about completed eras, results of games, and insertions and deletions of cases in the case base. It is also responsible for printing various notifications to stdout in realtime for monitoring, including a digest of the processing game state and some data from the decision process, which enables manual analysis of specific decisions being made and identification of bugs and flaws in the reasoning processes.

At the end of a run of games, the Head initiates all cases and records to be committed to disk.

5.3 Interface

The RFTG game engine and SCOUT are distinct processes, and the Interface module's purpose is to simplify the communication between them. In the current implementation it takes the form of a socket server, since this inter-process communication method is simple and transparent, and since communication time is not a major factor in the overall performance of the system. As mentioned, we patched a socket client into the Keldon game engine to connect to and communicate with this server. During initialisation, the Interface is initialised before the RFTG process is begun, and then blocks the SCOUT process until the RFTG process is initialised and connects to the server. The two processes then remain connected for the duration of a run. Thereafter the Interface is called upon once at the start of the main loop to receive a game state and once near the end of the main loop to deliver a decision back to the RFTG engine.

Because we modified the Keldon game engine to send state information in the same specification as that with which SCOUT works, very little processing is done to the data in the Interface, except to accommodate for changes that have been made to SCOUT over the development process. If we were to integrate SCOUT with a different game engine, however, further processing would be done here to convert the state data into the SCOUT state/case specification.

5.4 Case Controller

This module is responsible for the maintenance and analysis of SCOUT's case bases. It keeps records of case retrieval and usage efficacy, inserts new cases into the case base, and deletes unwanted cases from the case base. Upon initialisation, the Case Controller brings all stored cases into memory and reads a registry containing information about each individual case and each class of case.

During a run, states and their corresponding decisions are passed to the Case Controller, which combines them into a new case and it retains all new cases in a temporary problem case log - and likewise all of the cases retrieved in making the decision are recorded in a temporary retrieval log - until the outcome of the game is determined. Once a game is completed, the Case Controller processes all of the cases in each of these logs dependent on the outcome of the game:

If SCOUT won the game: The Controller passes through the retrieved case log and updates its and its class's registry entries with an additional retrieval and an additional victory. The Controller passes through the problem case log and places them into their relevant "wing" case base in the correct position.(This is further detailed in the Case Bases subsection and the next chapter)

If SCOUT lost the game: The Controller passes through the retrieved case log and updates its and its class's registry entries with an additional retrieval. If a case is has been retrieved a number of times beyond the grace period and is performing below the acceptable threshold, it is deleted from the case base, and a replacement is brought in from the wing case base into the main case base, if available. The Controller discards all cases in the problem case log.

Furthermore, if the Head module also signals that the era has ended, the Case Controller performs a culling operation on the entire case bases. Every case's registry entry is analysed, and if it has not performed well enough for its age it is deleted from the case base. After culling, the Case Controller replenishes the case bases from the wing case bases. Finally it writes the case bases and registry to disk as milestones so that case bases as of the beginning of each era are available for future analysis or to roll back the case base to an earlier point.

5.5 Placement Reasoner

This module, along with the Phase Reasoner, comprise the core of SCOUT's ability to play Race For The Galaxy. It receives the current state from the Head module, employs a K-Nearest Neighbour algorithm to retrieve the cases most similar to the current game state and adapts their solutions to make a choice to either place an available card or pass. It returns this choice to the Head module.

The module can also imports and utilises various items of game knowledge and feature weightings, but this data is either hand crafted or generated by programs external to SCOUT. In the current implementation there is a combination of limited expert knowledge and data derived from analysing one of the initial case bases.

The module also has testing methods which are valuable for development but not used in SCOUT's primary operation, in particular methods for leave-one-out and leave-one-in testing [25, p.224] to test retrieval functions across a pre-existing case base. This test uses each case one by one as an unknown case to be classified the same retrieval functions which are used in practice, and records data about how many times cases from one class are correctly classified, how many times they are classified into each other class, and how many times cases from other classes are classified as belonging to the given class. This data is also divided into degrees of difficulty for each classification, ranging from 2 to 10 possible choices from the 31 or 64 classes for Develop and Settle respectively.

In actuality, the leave-one-out function is a slight variation on the typical version. Rather than leaving just the case being tested out of the case base, all cases which share the same game ID are left out of the case base. This is done to accommodate for the fact that case bases typically contain every decision made from a successful game, especially before SCOUT's learning system prunes the case base. Cases from within a game share many similarities which would make them good candidates to match with other cases from the same game, and hence if they were left in during testing they would skew the data away from a realistic reflection of practical usage, where obviously cases from an in-progress game would not be available or useful for retrieval.

5.6 Phase Reasoner

Conceptually the Phase reasoning module is virtually identical to the Placement module, although its algorithms are in practice more complicated. In particular the module can generate hypothetical states as Placement problems and request the Head to solve them as it would a

game state proper. This is used so that the Placement module can be used to determine which cards SCOUT would place, given the opportunity, to assist the Phase Reasoner to determine which features to compare in its K-NN algorithm. This is covered in detail in the next chapter.

Apart from this, the role and functionality of the Phase Reasoner is analogous to the Placement Reasoner.

5.7 Payment Reasoner

This module is the simplest of the three reasoners. It passes through the relevant case bases and determines a ranking of each card's likelihood to be used as payment for a given placement, based on how frequently it was spent versus how frequently it was saved in previous payments for that placement. Since this ranking rarely changes, it is only recalculated periodically at the start of each era, and the rankings are cached by the Payment reasoner, which returns the top n cards from a ranking to the Head upon request, where n is the cost of a placement.

5.8 Case Bases

5.8.1 Layout of The Case Bases

SCOUT maintains 6 distinct case bases which are interrelated but organised independently of one another. These are:

- The Phase Case Base
- The Settlement Case Base
- The Development Case Base
- The Phase Case Base Wing
- The Settlement Case Base Wing
- The Development Case Base Wing

Each case base is used for a particular decision, clearly indicated by its name: The Settlement- and Development Case Bases are used by the Placement Reasoner to make Settle and Develop decisions respectively, and the same case bases are used by the Payment Reasoner to make Payment decisions, while the Phase Case Base is used by the Phase Reasoner to make Phase decisions. Each of these three case bases has a corresponding case base wing, where newly generated cases are held in order of usefulness but are not used by the reasoners, instead waiting in the wing until a space becomes available in the case base proper. Because a case is only relevant to its particular decision type, a case can only ever exist in one of the three case bases, and therefore a separate wing must be maintained for each case base proper so as to avoid more common types of cases, in particular Phase cases, taking up all the spaces and preventing any of the rarer Development cases from being held. Wings are arranged so that the cases are ranked upon entry into the wing dependant on how similar to them were the cases which were retrieved as their matches. When the time comes to promote a case into the case base proper, the highest-ranked case is always selected first, which is the case which had the lowest similarity to the cases which were retrieved to create it. The purpose of this is to encourage SCOUT to fill gaps in its case base, working under the assumption that if a new case was low in similarity to existing

```
1  p5:1816083232_r8-p0-d5_9321.xml:0:1:25.58:1
2  p3:2160350082_r8-p0-d5_2810.xml:0:0:16.65:9
3  p5:1465107409_r16-p5-d5_3282.xml:2:2:13.17:0
4  p0:1460609542_r12-p0-d0_2518.xml:1:2:16.36:0
5  p0:1458890879_r1-p0-d0_8889.xml:6:25:21.11:0
6  p3:1461583760_r9-p3-d5_2064.xml:1:2:16.00:0
7  p0:1470798405_r9-p0-d0_184.xml:4:9:15.03:0
8  p0:1459136614_r14-p0-d0_7862.xml:4:5:16.43:0
9  p0:1461932258_r2-p0-d0_1910.xml:12:32:18.30:0
10 p0:1466163833_r10-p0-d0_1042.xml:4:11:15.76:0
11 x:p5_68:23:132:0.00:0
12 x:p5_61:197:674:0.00:0
13 x:p5_60:192:664:0.00:0
```

Figure 5.2: *An extract from a case registry from the 10th era of a series of games. Each of line 1-10 are individual case entries, each of line 11-13 are case class entries. The columns are separated by colons, and are defined as follow:*

1. *Case type*
2. *Unique entry ID*
3. *Retrievals in won games*
4. *Total retrievals*
5. *Average unnormalised similarity score*
6. *Age indicated by era introduced*

cases then it must belong in a sparse region of the problem space. Wings also have a maximum size, and when this is reached, only new cases which outrank a current case in the wing will be inserted. This is further illustrated later in figure 6.4.

5.8.2 Case Registry

Information about all cases is maintained in a registry, which allows SCOUT to keep track of the utility of each of its cases, and the number of cases per decision type and per solution. Cases of the same decision type with the same solution are considered to belong to the same class. The information in this registry is:

For each individual case:

- How many times each individual case has been retrieved,
- How many times their retrievals were part of successful games,
- The average similarity score the case has when it is retrieved,
- The age of the case.

For each class of case:

- The total number of times cases of that class have been retrieved,
- How many times cases of that class were retrieved as part of successful games,
- How many cases of that class exist in the case bases.

This registry is used to determine the performance of a given case relative to the other cases in its class, and after a certain grace period - measured in both number of retrievals and amount

of time spent in the case base - cases become eligible to be deleted from the case base if they are under performing. This is further detailed in the next chapter. Deletion of under performing. cases is very important not only because of their potential to be lead SCOUT to poor decisions, but also as an attempt to avoid the swamping problem [34]. Thus cases that are never retrieved or retrieved very rarely (without exceptional success), are also deleted in order to reduce processing of a potentially very large case base.

5.8.3 Use of The Case Bases

Initially, each case base consists of an initial case base, created externally to SCOUT, and the wings are empty. During a run all case bases are held in memory, and the Case Controller reads them from and writes them to disk at the start and end of a run respectively. A case base has a pre-determined maximum size and, if this size is larger than the initial case base size, new cases will be brought in from the wings as soon as they are created in the early stages of evolving the case base until this maximum size is reached. Furthermore, over the an extended period, the case bases are purged of cases whose retrieval leads to an inordinate number of losses, and replaced with new cases from the wings.

The case bases themselves have little organisation internally but each reasoning module has specific structures for splitting the case bases to enable accessing relevant cases efficiently. In implementation, these are multiple Python dictionaries of pointers to cases, for which the keys are a feature or features of interest. For instance, the Phase Reasoning module may rapidly interchange between exploring its case base ordered by features such as tableau size, round, or case class, dependent on the requirements of the algorithm.

Chapter 6

SCOUT Development

This chapter will go into detail about the specific aspects which lead to the system described in the previous chapter. It will make general reference to SCOUT's performance, which will be discussed in further detail in the next chapter. The development of SCOUT can be divided into three main versions, each of which builds upon the previous.

The basis of SCOUT's reasoning faculties was an initial case base generated by a human player playing 1000 games against the Keldon AI. Every game which the human player won (748 games) was stored, and cases were extracted from it in different ways depending on the version of SCOUT. By using these cases, we hoped that SCOUT would be able to take advantage of the human player's superior skill by mimicking their play style. We also experimented with cases generated by running the Keldon AI against itself, in order to generate more cases than a human player could do in a reasonable amount of time. The impact of different initial case bases is discussed in the next chapter.

6.1 SCOUT1

SCOUT1 was the initial prototype system. It was capable of making placement decisions using k-NN algorithm on a case base with simplified cases with essentially only 2 indexed features. Despite it's simplicity, it was capable of performing its task with some success, and when used in tandem with the Keldon AI or a human player making the other decisions it was consistently superior to a system making random placement decisions. This encouraged us to proceed with the project and was also illuminating about the problem domain.

In essence, the reasoning approach for SCOUT1 was to attempt to recreate previous winning tableaux by exploring a case base of completed tableaux, retrieving those most similar to the tableau in the current problem state, and then choosing to place a card which would make the current state's tableau even more similar to the retrieved tableau. The motivating principle behind this was that cards which are together in a tableau in successful games potentially have good synergy with one another and attempting to recreate a successful tableau is analogous to repeating a successful placement strategy. The system therefore attempts to capture the reasoning process of a human player trying to build a coherent tableau from experience of prior games.

This type of case model is what Richter and Weber would describe as an extended view of a CBR system, whereby problem states are compared with potential solutions directly [25, p.41]. Later versions of SCOUT used a more standard case model, where a case is represented as a

pairing of a problem and a solution, and the system compares problem states to other problems in the case base, as opposed to comparing potential solutions.

A major factor in beginning with this approach was that completed game tableaux were able to be exported from a completed game within the Keldon engine by default, and thus we were able to prototype the system before beginning the challenging task of reverse-engineering the Keldon game engine to produce more sophisticated cases.

6.1.1 SCOUT1 Cases

In the SCOUT1 system, a case consisted of the victorious player's tableau from the end of a completed game, described in terms of:

1: The set of cards present in the tableau, with each card represented by a unique nominal ID.

2: The position of each card in the tableau, represented by an integer from 1-14

Listing 6.1: A SCOUT1 Case in XML form

```
<RftgStateCase>
    <Tableau count="9">
      <Card id="8" pos="1">Alpha Centauri</Card>
      <Card id="70" pos="2">Runaway Robots</Card>
      <Card id="100" pos="3">Mining World</Card>
      <Card id="72" pos="4">Terraforming Robots</Card>
      <Card id="84" pos="5">Radioactive World</Card>
      <Card id="61" pos="6">Galactic Trendsetters</Card>
      <Card id="27" pos="7">Lost Species Ark World</Card>
      <Card id="44" pos="8">Investment Credits</Card>
      <Card id="6" pos="9">Old Earth</Card>
    </Tableau>
</RftgStateCase>
```

Listing 6.1 is an example of a case as represented in XML form. Each Card element has an id attribute representing the unique card id and a pos attribute representing that card's position in the tableau. The card names in text were present to allow us as developers to understand the cases. SCOUT would interpret the set of all ids as the first feature, and the set of (id,pos) tuples as the second feature.

6.1.2 First SCOUT1 Retrieval Function

Initially, this was implemented only in terms of the first feature. The system would create a problem by enumerating the cards in the current tableau. Furthermore it would create a list of all the cards which could possibly be placed in this instance, which represented all possible choices for the solution to the current problem (hereafter *solution cards*). It would then filter the case base by ignoring any cases whose tableau did not include at least one of the choices currently being considered for placement. It would then pass through the case base and compare each case to the problem situation, taking as the distance the cardinality of the intersection of the sets of cards in the tableaux over the maximum possible distances which was 14.

$$dist(p, c) = |(T_p \setminus T_c) \cup (T_c \setminus T_p)| \tag{6.1}$$

Where T_p and T_c are the sets representing the tableau in the problem situation and the case respectively. This is a typical approach to comparing set-valued features [16].

The system would then take the K cases with the smallest distance and adapt them into solutions. This was achieved by extracting the card from the retrieved tableau which was a solution card for the current game state and adding it to a list of possible solutions. In cases where two or more of the choices existed in the same tableau, each would be added to the list. Typically this would result in a list of slightly more than K solutions, most of which were of a single class with one or two other solutions occurring less frequently. The system would then randomly select one solution from the list, resulting in each possible solution having a probability to be selected of:

$$P(s) = S/k \tag{6.2}$$

Where S is the number of solutions s in the k-sized list of possible solutions. The aim of this was to favour solutions of placing cards which occurred more frequently in successful games with similar tableaux, while also giving the potential to occasionally select less frequent cards (nevertheless with some supporting evidence of success) to diversify SCOUT1's play style. As discussed below in the next chapter, this approach had its share of flaws and in particular was not very discriminating, and the best K retrievals would frequently all have the same distance but at least 2 different solutions adapted from them, requiring random tie-breaking with little bias amongst the top 2 or more choices. Thus the system was overly probabilistic, which was undesirable. Nevertheless the system demonstrated an ability to eliminate very inappropriate options and favoured generally superior cards. The overall effect was a system which randomly chose from a selection of the better possible moves, but frequently displayed an erratic play style.

6.1.3 Second SCOUT1 Retrieval Function

One of the major flaws was that the system was that it had no way of differentiating between cards which were placed into the tableau early in the game and cards placed in the later game, as it merely checked for a card's presence in the tableau. Thus cheaper cards which were played early in the game in retrieved were being selected as solutions to late game placement problem over more expensive cards, which were superior choices but used more rarely. In particular it heavily favoured "start worlds", the 5 cards of which and at least one was always present as the first card in a tableau, despite their being generally weaker cars rarely worth placing. Thus they contradicted the assumption that a card's presence in a similar and successful tableau suggested a potential good placement. This was addressed by indexing the second feature, the position of the potential solution card in the tableau. Now, the distance would be a weighted combination of the cardinality of the intersection as before, and the difference between the position of the solution card in the retrieved case's tableau and the size of the current tableau plus 1, which would be the card's position in the current tableau if it was chosen.

$$dist(p, c, s) = w_1|\big((T_p \setminus T_c) \cup (T_c \setminus T_p)\big)| + w_2|(|T_p| + 1 - p_s)| \tag{6.3}$$

Where T_p and T_c are the sets representing the tableau in the problem situation and the case respectively, w_1 and w_2 are feature weightings, and p_s is the position of the potential *solution card* in the case tableau.

This development improved SCOUT1's ability to discriminate between cases, thus reducing the reliance on random differentiation, and also reduced its inappropriate selection of cheaper cards and start worlds in the late game. It still retained numerous critical flaws which could

not be addressed without using a model whereby a problem situation is compared directly to problems in the case base, as per what Richter and Weber describe as the "standard view" of CBR [25, p.41]. One such flaw was that the use of endgame tableaux as cases meant that leave-one-out testing was impossible, as a case was not analogous to a problem, and hence the only testing could be done by examining the overall performance of the system and manually comparing it to decisions which we would have made in the same situation. This was time consuming and limited the amount of testing which could be done, and also made it difficult to analyse the impact of weighting adjustments. For the next version of SCOUT, lessons learned from here were used in developing the similarity metric for the player_tableau feature.

6.2 SCOUT2

SCOUT2 was vastly more sophisticated than SCOUT1: It used a case base of cases whose features captured every known aspect of the game state and which represented a problem-solution pair as opposed to just a solution. Furthermore it was able to reason about not just Placement decisions but also Phase/Action decisions and Discard decisions. Its reasoning systems were the same as those implemented in SCOUT3, but SCOUT2 lacked the capacity to revise and retain cases or maintain its case base. It was able to play an entire game independently of the Keldon AI or a human player, with the single minor exception of how to distribute goods during the Consume phase, which was ignored as we judged it to be a difficult problem with little impact on overall performance.

6.2.1 Placement

The reasoning approach for SCOUT2's placement system was essentially to attempt to find the game states from previous successful games most similar to the current game state and adapt the decision made in that case to the current situation. This was a more standard CBR approach than the system for SCOUT1. The rationale behind this was that if a case was similar enough to the current state in the relevant features then SCOUT2 could take advantage of all of the reasoning and planning which the player used to make a decision in the original case.

Retrieval

Again, a k-NN algorithm was used to retrieve cases. Each case in the case base now represented a specific game state, defined in the same terms as the problem situation, and with a single solution. Deciding what card to place in the current game state essentially became a classification problem, where each of the 95 cards were represented by a class, and cases with a particular card as a solution belonged to the class representing that card. By classifying the problem case, the system determined the best card to place.

As is typical, the algorithm passed through the entire case base and evaluated the similarity of a case to the problem case by summing the weighted similarities of each indexed case feature. From each of the k best matching cases an appropriate solution for the problem was adapted and added to a multiset of solutions, and finally, as with SCOUT1, a single element of the multiset was randomly selected as the solution. Unlike, SCOUT1, the elements in the multiset were largely homogeneous because SCOUT2's retrieval algorithms were more effective in classifying the cases consistently. Therefore the random element was much less pronounced and frequently completely deterministic, as all retrieved cases yielded the same solution. When the stochastic element did come into effect this was generally heavily biased toward a single good possibility, with an improbable secondary possibility providing some desirable variation.

Indexing

SCOUT2 processes cases from the xml schema shown in figure 4.1 into an an internal representation of the case with 23 features, representing a game state, paired with the decision which was made in response to that game state, which constitutes the case's solution. The features are as follow, and are unindexed where not specified:

game_id A non-unique nominal id shared by all cases which were generated within the same game.

case_name A unique nominal id for each case.

game_round An integer representing the game round in which the game state occurs. Indexed with high importance for both decision types.

player_chips An integer representing the number of victory point chips the player possesses.

player_hand A set of nominal ids representing the cards in the player's hand. Indexed with low importance for both decision types.[1]

player_hand_size An integer representing the number of cards in the player's hand. Indexed with high importance for Action/Phase Selection decisions.

player_military An integer representing the player's military score. Indexed with high importance for Placement decisions.

player_goods A set of nominal ids representing the player's goods. Indexed with high importance for Action/Phase Selection decisions.

player_score An integer representing the player's total score. Indexed with low importance for both decision types.

player_tableau A set of nominal ids representing the cards in the player's tableau. Indexed with very high importance for both decision types.

player_tableau_size An integer representing the number of cards in the player's tableau. Indexed with high importance for both decision types.

opponent_chips An integer representing the number of victory point chips the opponent possesses.

opponent_hand_size An integer representing the number of cards in the opponent's hand. Indexed with high importance for Action/Phase Selection decisions.

opponent_goods A set of nominal ids representing the opponent's goods.

opponent_military An integer representing the opponent's military score

opponent_score An integer representing the opponent's total score.

opponent_tableau A set of nominal ids representing the cards in the opponent's tableau. Indexed with moderate importance for Action/Phase Selection decisions.

[1] The player's hand is extremely important in game terms, but as a feature its use in terms of case similarity is limited, for reasons discussed later in this chapter. The relevant cards present in the player's hand are instead presented to SCOUT as potential solution cards for the problem game state.

opponent_tableau_size An integer representing the number of cards in the opponent's tableau. Indexed with very low importance for both decision types.

score_difference An integer representing the player's score minus the opponent's total score. Indexed with moderate importance for Action/Phase Selection decisions.

deck An integer representing the number of cards currently in the deck.

discard An integer representing the number of cards currently in the discard pile.

pool An integer representing the number of victory point chips currently available. Indexed with high importance for Action/Phase Selection decisions.

Two of these features, case_name and game_id, have no inherent meaning within the game and are only used to identify cases, but the remaining features are all potentially indexed features. We judged this number to be too high, especially since player_tableau and opponent_tableau in particular are highly complex features in themselves, and we aimed to identify which of these features were relevant to Placement decisions. Reducing the number of features as much as possible was important because k-NN algorithms have a tendency to be highly sensitive to irrelevant, interacting and noisy features [24] Such identifications could be made with domain expertise, however since we wished to find a method by which this could be done naively, and also which had the potential to expose unexpected patterns, we used statistical analysis on the case base to identify the most relevant features.[2] For each numeric feature, the distribution of values across the entire case base was compared to its distribution of values among cases from each class separately, and if these distributions were found to vary significantly across several classes and the general distribution then these features were determined to be of relevance in terms of case similarity. For example, the a certain Development "Contact Specialist" (Figure 6.1) is almost always played when the player has 0 or -1 military score, while across the entire case base cards are played with various military scores.

Figure 6.1: *Contact Specialist negatively affects the player's military score, and therefore military score is a relevant feature to consider when placing it. Card image property of Rio Grande Games [15]*

Similar patterns with many cards indicates that the player's military score is a highly relevant feature in terms of the player's reasoning process. On the other hand the size of the discard pile, for example, was found to have a no relevance because its distribution across the entire case base was found to be largely identical to its distributions on a class by class basis. Several features

[2]For this purpose we used a large case base of 10000 games generated by the Keldon AI playing against itself.

were found to be relevant for Phase selection but not Placement, and vice versa, while some features were always relevant and others were found to be never relevant.

In general the results strongly correlated with what we judged to be the most relevant features via our domain knowledge. This approach was not appropriate for the set-type features, but since they were few and expressed the majority of the game information they were adjudged to be important and indexed.

Feature Weighting

Feature weighting is an important aspect of k-NN algorithms, as the algorithm is highly sensitive to them, but notoriously difficult to optimise manually [40]. We attempted approaches inspired by Aha's CBL4, which updates feature weights based on successful or unsuccessful classifications [3], as well as an attempt at using genetic algorithms using the DEAP toolbox [12], inspired by Skalak [33]. Our evaluation function for the evolutionary algorithm consisted of running a leave-one-out test on a subset of the cases. This was very expensive in terms of time for an evaluation function and this rendered our trials ineffective. Ultimately, we opted to use domain knowledge and experimentation to define feature weightings by hand. This is potentially a major weakness in SCOUT2's retrieval algorithm and future work could focus on improving this.

Feature Similarity Metrics

Numeric Features In determining feature similarity, the numeric features' metrics were all simple comparisons which essentially symmetrically measured the difference between the feature's value in the retrieval case and the problem.

$$dist(p, c)_f = (|f_p - f_c|)^{g_f} \tag{6.4}$$

Where f_p and f_c are the values of the given feature in the problem and case respectively, and g_f is a predetermined value to adjust the exponentiality of the function. Essentially, a higher g_f value gives a stronger bias towards similar values. It is a supplementary weighting measure.

Symbolic Features The symbolic features were vastly more complicated. Consider the feature player_tableau, which was expressed as a set of symbolic card ids representing those cards currently present in the player's tableau. We noted that merely finding the cardinality of the set intersections to determine local similarity, as per SCOUT1, treated all cards as equally relevant to a decision, whereas when a human player would reason about their tableau they would consider certain cards to be more relevant than others, depending on the current situation. We wished to capture this by treating each card in the feature as a subfeature with its own specific weight in determining the feature similarity, and furthermore we wanted these weightings to vary depending on the options currently being considered.

As an example, consider three cards: Alien Tech Institute, Alien Robotic Factory, New Vinland (Figure 6.2). A skilled player would recognise that Alien Tech Institute and Alien Robotic Factory are cards that work extremely well together in a tableau, as the former makes the latter easier to play, and the latter makes the former worth more points. However they are expensive and only used in specific strategies. In contrast, New Vinland are cheap and generally useful cards that are regularly used in many different strategies.

By statistically analysing the case base we can identify that having an Alien Tech Institute in play greatly increases the likelihood of playing Alien Robotic Factory, whereas New Vinland's presence does not affect the likelihood of playing it. Therefore when an Alien Robotic Factory is

Figure 6.2: *The first two cards compliment each other well, whereas the third has little relation to them. Card images property of Rio Grande Games [15].*

one of the potential solution cards for a problem case, we would like to give Alien Tech Institute more weight in determining the similarity of the tableaux than New Vinland.

Cards whose presence negatively affects the likelihood of another card being played can be treated in exactly the same way, such a card is given higher weighting in determining similarity, making cases with that card present in the tableau more likely to be retrieved, which will in turn have a high probability of having a solution different from the card which would be a bad choice.

This is an attempt to merge basic association rules with the similarity function. Furthermore this approach is superior to purely using association rules because it allows the system to reason about fringe cases where the best card to play would contradict the general trend indicated by the rules. Despite the high correlation between having Alien Tech Institute in play and placing Alien Robotic Factory, there are cases where this is not the best good choice. These cases are will have the same local similarity in terms of the tableau, insofar as all other cards are equal, and if other features are more similar then the system will retrieve the fringe cases and can reason that it is a good situation to play a different card.

This is an adjusted example of class-specific weighting [40]. For each case class a separate list of weightings was defined indicating a specific card's relevance in terms of reasoning about whether to classify a case in that class. These weights were determined by analysing the case base and finding how frequently certain cards were present in a tableau when the given card was played, relative to how frequently they occurred in tableaux in the general case across the case base. If the former figure indicated a higher frequency, then that card was considered more relevant for that class, relative to the difference.

We experimented with 2 approaches for utilising these weightings, finally settling on the second approach:

- The first approach was to determine the weighting based on the class of the case being currently retrieved from the case base, for example when the problem was being compared to a case of class 12 then the subfeature weightings would be those defined for class 12. Normalising the similarity scores with different weightings proved difficult, however, and as a result cases of certain classes became disproportionately likely to be retrieved.

- The second approach, which appeared to be much more successful was to generate the weightings by merging those of the potential classes of the problem, for example if the potential solution card s for the current state belonged to the set $S = (12, 21, 98)$, then the problem potentially belonged to one of those three classes. Then for a given problem p, for each subfeature the weighting w_{t_p} would be defined as the maximum weighting of that subfeature amongst the weights defined for those three classes w_{t_s}.

$$w_{t_p} = max(w_{t_s}), s \in S \tag{6.5}$$

  This avoided the problem of the normalising local similarity scores because the weights were always the same for any given problem. As a result, minority classes were not put at a disadvantage [10].

Ultimately then, the local similarity metric of the player_tableau feature could be expressed as such:

$$dist(p, c)_f = 1 - \frac{\sum_{t \in (T_p \cap T_c)} w_{t_p}}{\sum_{t \in (T_p)} w_{t_p}} \tag{6.6}$$

That is, 1 minus the sum of all derived weights w_{t_p} for card/subfeatures t present in the intersection between the tableau in the problem T_p and the tableau in the case T_c, normalised by dividing by the maximum possible similarity between tableaux, where $T_p = T_c$.

The player_hand and opponent_tableau features were handled in exactly the same way.

Adaptation

We experimented with two different approaches to adapting the retrieved cases into solutions, which we will call the 1-Stage and 2-Stage adaptations. The 2-Stage adaptation was more computationally expensive but was quickly seen to make more correct decisions in leave-one-out tests and was the one retained.

1-Stage Adaptation The 1-Stage adaptation used the simple approach of directly reusing the solution of the retrieved case as the decision to make in the current game state. To avoid retrieving cases which did not have directly reusable solutions each case was initially checked to see if its solution was also a potential solution card from the current problem state, and if not, the case was skipped.

2-Stage Adaptation This approach favoured examining every possible case and using a more complex function to adapt the solution of the retrieved case to the current game state. This was more expensive during retrieval time due to the fact that cases could not be discarded before comparing similarity on the basis of having an incompatible solution. Of course the advantage was that the number of potential retrievals was vastly increased.

This required every card to have a score describing its similarity to every other card. Since Development solutions would never be needed to adapt into Worlds or vice versa, these scores were stored in a 31x31 matrix and a 64x64 matrix for Developments and Worlds respectively, and normalised such that a score of 1 indicated the card itself or an identical card and a score of 0 indicated a card dissimilar in all features. The solution to a retrieved case would be adapted into a possible solution by replacing the card which was placed in the retrieved case with the solution card which had the highest similarity score to it (Figure 6.3).

We experimented with two approaches to generate these similarity scores:

```
100
100,Mining World,1.0
15,Comet Zone,1.0
87,Bio-Hazard Mining World,0.846153846154
78,Imperium Armaments World,0.784615384615
13,Gem World,0.723076923077
95,Prosperous World,0.692307692308
91,Blaster Gem Mines,0.692307692308
96,New Earth,0.661538461538
27,Lost Species Ark World,0.661538461538
32,Distant World,0.630769230769
```

Figure 6.3: *Extract from card similarities data, showing 10 most similar cards to card 100 (Mining World). Column 1 is card id, column 2 is card name, column 3 is similarity to card 100. Note that Mining World and Comet Zone are functionally identical cards and this enables SCOUT to treat them as such when comparing and retrieving cases.*

First Approach The first approach was a naive one which ran a leave-one-out test with the retrieval part of the algorithm and returned the total number of times that cases of a particular class were classified correctly and how many times it was misclassified into each other class. Classes which were more frequently misclassified as particular other classes were expected to represent solution cards which were more similar to the latter in strategic terms, and therefore good options for adaptation. This approach was favoured as it was more generalisable and utilised no domain knowledge which we wanted to avoid, but it was flawed in that there were too few cases to give sufficient sample sizes for all the cards, leading to less commonly played cards having unreasonably high similarity scores.

For comparisons between the majority of cards it was noticed that there was indeed a strong correlation between cases of one class frequently being misclassified into a particular other class and the cards represented by those classes being strategically similar. As discussed in the next chapter, there were even several instances in which these figures were not immediately intuitive but upon closer analysis provided interesting strategic insights.

Second Approach To evade the problem of noisy data due to limited sample size, we tried a less naive approach, which instead analysed the cards' features in gameplay terms using data from the Keldon game engine, and assigned determined local and global similarity between cards much in the same way as a Nearest Neighbour algorithm. This approach generated the data in figure 6.3.

This approach proved quite effective, especially for Worlds which were more readily comparable to one another based on their features. In the case of certain Developments, cards which had similar strategic uses but contrasting features were given unreasonably low similarity scores. These were often the same cards which were compared surprisingly well with the first approach. Ultimately, we combined these approaches. The first approach using human cases was used as a baseline and scores which were significantly out of line with those generated using the second approach were manually adjusted to better reflect expert opinion on the cards' similarity.

6.2.2 Phase/Action Selection

Phase/Action selection reasoning follows essentially the same procedure as Placement, but there are a few differences which will be discussed.

Hypothetical Problems

One particular challenge we encountered with Phase selection was differentiating between good situations for selecting Develop and Settle phases. In leave-one out testing cases with these solutions would be confused with eachother to a high degree, although SCOUT did a good job of differentiating the two of them from other classes. The reason for this was likely that the player_hand feature had a high degree of relevance to differentiating between these classes, a feature which proved particular difficult to process in terms of a Nearest Neighbour algorithm. Two different hands of 10 cards could be similar for 9/10 cards, and yet if the 10th card is the best card to play in a particular situation then the other cards' similarity is almost meaningless.

A partial solution to this was to include a function within the Phase selection algorithm to generate hypothetical problem, creating Settle and Develop Placement problems, as they would occur in the scenario that SCOUT classified the current problem as one of those. SCOUT would then evaluate those problems in the normal manner as Placement decisions and retrieve the k best matches for each in the normal manner, which would then be combined into the k best matches overall, and from that set of matches, a single solution chosen. The selected case's decision type (Settle or Develop) would then be used as a solution to the Phase selection problem.

Feature Generalisation

An observation made during development of the Phase selection algorithm was that the number of cases available was not large enough to provide adequate coverage of the problem space. This meant that problems that occurred in sparsely covered regions of the problem space were being matched with largely unrelated cases. In the placement algorithm, the adaptation process did a sufficient job of compensating for this, but here the selection of solutions indicated by the k best matches were too varied to be of use. To address this we opted to reduce the size of the problem space by reducing the range of values through generalising feature values, using domain knowledge. In particular, cards were grouped by attributes and grouped cards were treated as equivalent for certain similarity functions. Whereas in an ideal situation each card was treated as unique and their qualities were understood only implicitly by their presence in cases, SCOUT now had information about the meaning of card ids in terms of a card's ability to produce and consume goods, and how much cards cost to place. In game terms, this meant that SCOUT2 could treat two different Production Worlds as more similar than a Production World and a Windfall World and allowed it to identify similar cases in sparser areas of the case base, but it came at the cost of limiting its ability to differentiate finely between cases in densely covered areas of the problem space. It also meant that SCOUT2 was given some hardcoded strategic information rather than learning exclusively from the case base, as was our aim.

Additionally, we defined auxiliary features which were derived from the tableau and hand features, but with narrowed ranges, such as PlayerTableauSize and PlayerNumberOfGoods, which simply defined the cardinality of the sets which they were derived from, but which provided useful measures for comparing features.

Another instance of this was give SCOUT an indication of whether the game was in the early-, mid-, or late-stages of the game based on particular features. The Phase module categorised its cases in terms of the size of the player_tableau and Pool features, which were related to the two aspects of the game which could cause the game to end. The retrieval algorithm was then limited to explore only cases with values for those features within small and fixed ranges of the problem case's values, and again these ranges were derived from experimentation and domain knowledge. This was based on the assumption that cases with highly similar values for these features would occur within a similar stage of the game, and therefore the reasoning process would have taken similar factors into account. Again, a more thorough case coverage may have

allowed those factors to be discovered implicitly.

6.3 SCOUT3

SCOUT3 is the latest version of SCOUT, which combines the reasoning faculties of SCOUT2 with the ability to maintain and improve its own case base. Therefore it implements the Revise and Retain stages af the CBR cycle, rounding out SCOUT as a complete CBR system and allowing it to improve its own performance, ideally, teaching itself how to play RFTG more effectively.

In addition to the initial case base generated by a human player, SCOUT3 would generate its own cases by playing the game, and continuously evaluate the quality of its cases and the density of its case base. This is an attempt to adapt Leake's methods of performance-guided case base maintenance to our particular needs [20].

6.3.1 Case Life Cycle

The life cycle of a case was as follows, and as illustrated further in fig 6.1:

1. The case is generated by combining a problem case with the solution selected for that problem case. The new case is temporarily stored until the end of the current game.

2. Upon the end of the game, the outcome of the game is evaluated, and if the game is successful, then all cases generated during that game are evaluated for usefulness.

3. The case is evaluated in terms of how similar it was on average to the k nearest matches, and those with lower average similarity are considered more useful, as they belong in a sparser region of the case base, and our objective in populating the case base is to achieve good coverage.

4. The case is inserted into a secondary case base, a "wing" case base, which sorted in terms of usefulness and is specific to its decision type. The size of each wing case base is limited to a certain value W_d, and any case less useful than the W_dth case is immediately discarded.

5. The case waits in the wing until it is either discarded by being outranked in terms of usefulness by too many new cases, or it becomes the most useful case in the wing, and is then brought into the main case base.

6. Upon being entered into the main case base, the case is registered in the case registry, which records its type, number of times it is used and its adapted solution is used as SCOUT's choice, number of times it is used in successful games, and the number of games it for which it is present in the case base (age).

7. The case is retained in the main case base, and evaluated during each relevant pass through the case base, and its usage is updated in the registry.

8. Once the case has been in the case base long enough, it becomes eligible to be evaluated for quality after each time it is used. This begins to occur once the case's age is greater than a fixed value G, which defines a grace period in which the case is not evaluated in terms of quality. The evaluation is made in terms of ...

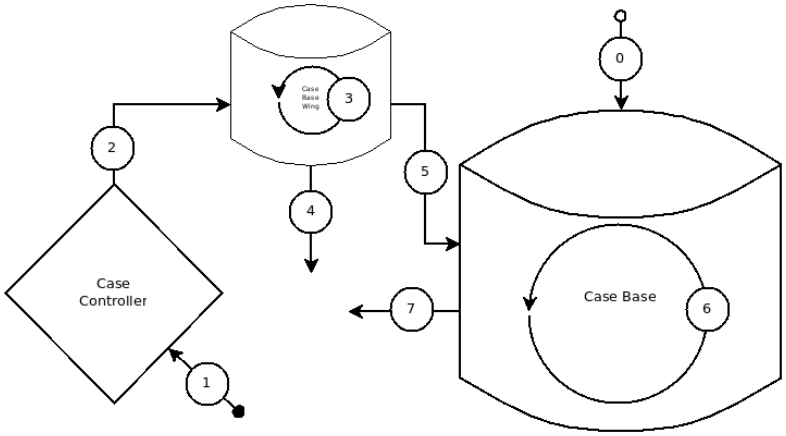

Figure 6.4: *SCOUT3 Case Life Cycle*

(0. *Initial case is loaded from initial case data)*
1. *Case is built and evaluated by Controller*
2. *Case is inserted into Wing*
3. *Case waits in wing*
4. *Case discarded without being used*
5. *Case inserted into main case base*
6. *Case used and evaluated in case base*
7. *Case discarded if under performing*

9. The case continues to be evaluated each time it is used, and if it is evaluated to be of low quality then it is deleted from the case base.

*Cases from the initial case base begin at stage 6.

This is an attempt to address issues related to the Temporal Credit Assignment Problem [35]. Over the course of a game a player may make decisions of varying quality, and although the assumption is that a successful game will have more good decisions than bad, if the final outcome of the match is the only metric for judging the quality of a case then it is inevitable that some strategically poor decisions will be rewarded via insertion into the case base. Therefore the continued evaluation of cases as they are used is necessary to determine their actual quality, with the assumption that a case whose use leads to more successful games contains a good decision for its game state.

6.3.2 Evaluating Cases

Another limitation is that, although a case may consist of the "correct" decision for its own game state, there is the risk that it is retrieved and adapted to inappropriate game states, leading to unsuccessful games. Because of this, and the stochastic nature of the game, there needs to be lenient in evaluating the case and allowance for unsuccessful retrievals. The number of successful retrievals which a case has needs to be measured against other cases of its type and class, and furthermore there needs to be enough retrievals made to gain a certain amount of statistical validity.

The reason for limiting the size of the case base is that it takes many games to evaluate the quality of a single case in the case base, over which time many more cases will be generated through successful games. Continuing to add every case into the case base results in the volume of

low-quality and unevaluated cases overwhelming the high quality cases which have been retained and proven over a longer period of time. This in turn creates the need for the ranked waiting system, in order to retain the most useful cases generated over an extended period of time as opposed to simply taking whatever new case is available at the time space becomes available in the case base.

In early trials, the case's minimum successful retrieval rate threshold, below which a case would be deleted, was equal to what was found to be the average successful retrieval rate for all cases of the same class. The effect of this tended to be that average cases would quickly be eliminated if their retrieval success briefly dipped below the average, and this resulted in cases being destroyed too readily. This was compensated for by setting the delete threshold to be slightly below the average success rate for a given case class.

Chapter 7

Performance and Discussion

7.1 SCOUT's Performance in Games Against Keldon AI

As of SCOUT3, SCOUT's overall performance does not reach the Keldon AI's level, let alone that of a human player, but it does demonstrate an ability to play reasonably and competitively. This chapter will cover the results of many runs of games against the Keldon AI, along with other benchmarks. This will be followed by discussion about SCOUT's strengths and weaknesses as indicated by these results.

7.1.1 SCOUT1

Since it could only reason about one type of decision, SCOUT1's performance was measured by integrating it with the Keldon AI to make the remaining decisions of the game. For comparison we also tested other agents controlling the placement decision: the Keldon AI, a Random agent, and a Human. The Random agent merely selects one of the possible options with equal probability. The purpose of this is to give an indication of the absolute minimum level of performance possible. The human player is the same whose cases trained SCOUT. Contrasting the Random agent, this is intended to give a rough indication of ideal performance.

Each test was comprised of 5000 games against a player controlled by a pure Keldon AI agent, except for the Human, for which the test was only 50 games. The number of games was selected by running the Keldon AI against itself until it reached a stable victory rate. The victory rate includes tied matches, hence the Keldon AI's victory rate against itself being greater than 50%.

Placement controlling agent	Victory rate
SCOUT1	32.1%
Keldon AI	51.0%
Random	21.1%
Human	54.5%

Figure 7.1: *Victory rate of four different agents controlling placement decisions, in tandem with Keldon AI making all other decisions, against a second Keldon AI agent.*

As indicated by Random's relatively strong performance, and Human's relatively weak performance (see later results), the integrated Keldon AI has a major impact on the final results. This

is to be expected as placement choices are of secondary importance to action/phase selection. Nonetheless despite its simplicity SCOUT1 clearly outperforms a random agent.

7.1.2 SCOUT2

Overall Performance vs Keldon AI

SCOUT2's performance was tested similarly to SCOUT1, except with SCOUT2 controlling all the decisions for which it had the capability, effectively all of them. Again, it is compared against the performance of Keldon AI, Random, and Human agents, performing the same tasks as SCOUT2 does. Again 5000 matches were played between each agent, except for the Human (1000 games).

Full controlling agent	Victory rate
SCOUT2	30.2%
Keldon AI	51.0%
Random	0.04%
Human	74.8%

Figure 7.2: *Victory rate of four different agents controlling all strategically significant decisions, against a Keldon AI agent.*

The drop from SCOUT1's win rate is due to SCOUT2 additionally controlling the Action/Phase selection decisions. The sharp increase in the Human's win rate and total inability for the Random agent to win a game (barring exceptional circumstances), demonstrate that this is a better indication of the reasoning quality of the controlling agent. This is the most important result, and it clearly indicates that SCOUT is not as strong as the Keldon AI in overall performance, but is capable of playing and winning in a way that a non-reasoning agent cannot.

Score Distribution

Figure 7.3 shows the distribution of score ratios across 5000 games between SCOUT2 and Keldon AI. Scores are best measured relative to the opponents score, as it is not useful to measure scores in absolute terms across multiple matches. Shorter matches typically have a lower winning score, but they are not necessarily indicative of inferior performance to a higher score from a different match, indeed the opposite is often the case. Within a single match, however, close scores typically give some indication that the performance of the competing players was also close.

Representing the score of a match as $S_i = S_{s_i}/S_{k_i}$ where S_{s_i} is SCOUT's score in a match and S_{k_i} is Keldon AI's score, gives log-normal distribution of scores.

These scores show that although SCOUT loses the majority of matches against Keldon AI, it usually achieves a competitive and respectable score. The score ratios have a median of 0.85, indicating that although reaching a 50% win rate against Keldon AI would mean an 66% increase in win rate, it would take only an 18% increase in SCOUT's scoring would bring it to that level.

Phase Selection

Comparing the frequency of selected Actions/Phases of SCOUT to the Keldon AI and the human player whose cases trained it, it can be seen that in SCOUT's choices follow the same general trend of both other players (Figure 7.4). This is evidence of reasonable play compared to a random agent which would have an equal frequency distribution across all actions.

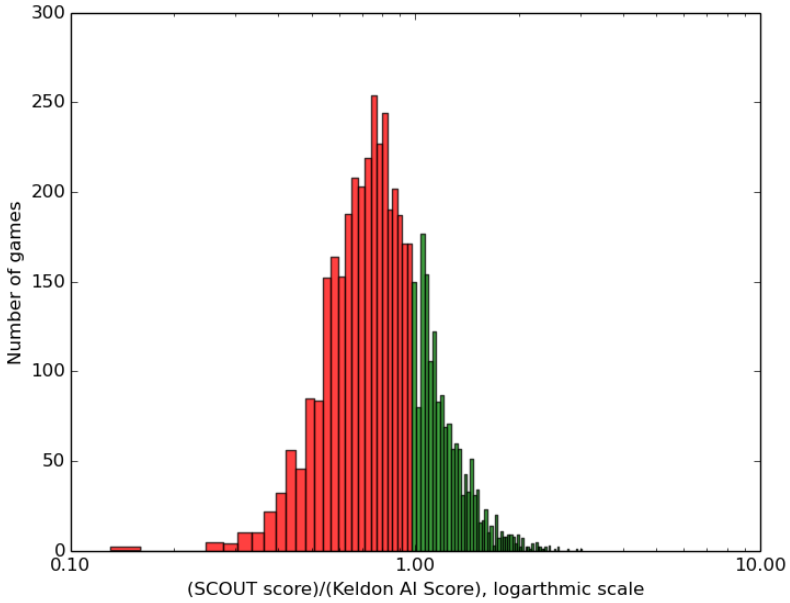

Figure 7.3: *Distribution of scores from 5000 games between SCOUT2 and Keldon AI, SCOUT won or drew the game when* $x >= 1.0$ *(31.7%). Note that the value of won or drawn games here is slightly higher as it includes those drawn matches which went against SCOUT in the tiebreaker.*

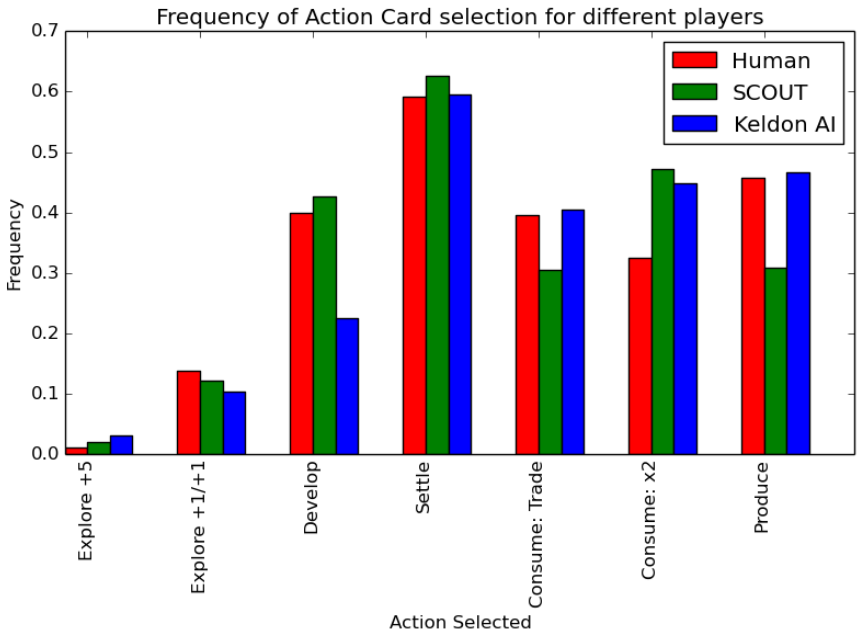

Figure 7.4: *Frequency of Action Card selection for different agents*

A noticeable feature is that despite our observance that SCOUT follows a Consume-Produce strategy as frequently as possible, it in fact does not select Produce as frequently as either other player. This is likely explained by SCOUT's inability to perfectly manage its producer cards against its consumer cards. We regularly observed it producing many more goods than it could consume at once, and hence over 3 turns it would call Produce, then Consume, then Consume again, whereas a skilled human player, and to a lesser extent Keldon AI, tend to have more balanced numbers and thus call Produce and Consume on alternate turns.

A more promising observation is that across the first four action types, SCOUT's frequencies are more similar to the human player than to Keldon AI, indicating some success in mimicking the human's play style, at least in terms of selection frequency.

7.1.3 SCOUT3

Performance Change Over Time

Figure 7.5 shows the number of wins per era (of 200 games) over three separate runs of 4000 games. The case bases are maintained and added to over this these runs, and ideally a good maintenance system should result in an improvement in performance over time.

There is no clear trend across these results, although the best fit lines of all three runs indicate a slight performance improvement over time. The results are limited by the number of runs, and due to their subtlety may indicate an actual improvement or natural variation. More test runs over more games may provide evidence for a stronger claim. Running so many games is costly as SCOUT3 is not designed to work quickly, especially as the case bases increase in size. Each of the test runs took about 5 days.

If the line of best fit does indicate a real improvement, then only subtle improvements are to be expected, as the case base size increase is gradual, and due to the inferiority of SCOUT to the human player, the average quality of the added cases should not be equal to those of the initial case base. The net increase in case size over time is dependent on the size of k in the relevant retrieval algorithm, the maximum case base size, and the length of the two grace periods in terms of age and number of retrievals for which cases are immune to deletion. With the settings set during these runs the size of the Settle case base, for example, increases by on average approximately 80 cases per era. Furthermore there is a once-per run mass deletion at the start of the Gth (length of the grace period) era, at which point approximately 700 of the original cases are deleted for having never been used in a successful game. As a result, the Settle case base increases from 2947 cases initially to 3367, 3765, and 3603 in the 20th era of the first, second, and third run, which is not a significant increase in terms of problem space coverage.

Case Quality

The case base maintenance system presumes a high degree of reliability in terms of SCOUT's retrieval function, in that cases which are frequently used in losing matches are generally poor cases, or at least cases which do not work well within SCOUT's system. In practice however, it is possible that poor performance is also partly due to SCOUT sometimes retrieving cases inappropriate to the current situation, regardless of whether the decision in the case was a good one in the context in which it was created. In order to evaluate this, we scanned the case registries at the end of the 20th era of each run, to attempt to identify cases with commonly high success rates.

This was achieved by identifying those cases retained from the initial case base for all 20 eras of the run. This resulted in 781, 891, and 773 cases respectively. The cardinality of the

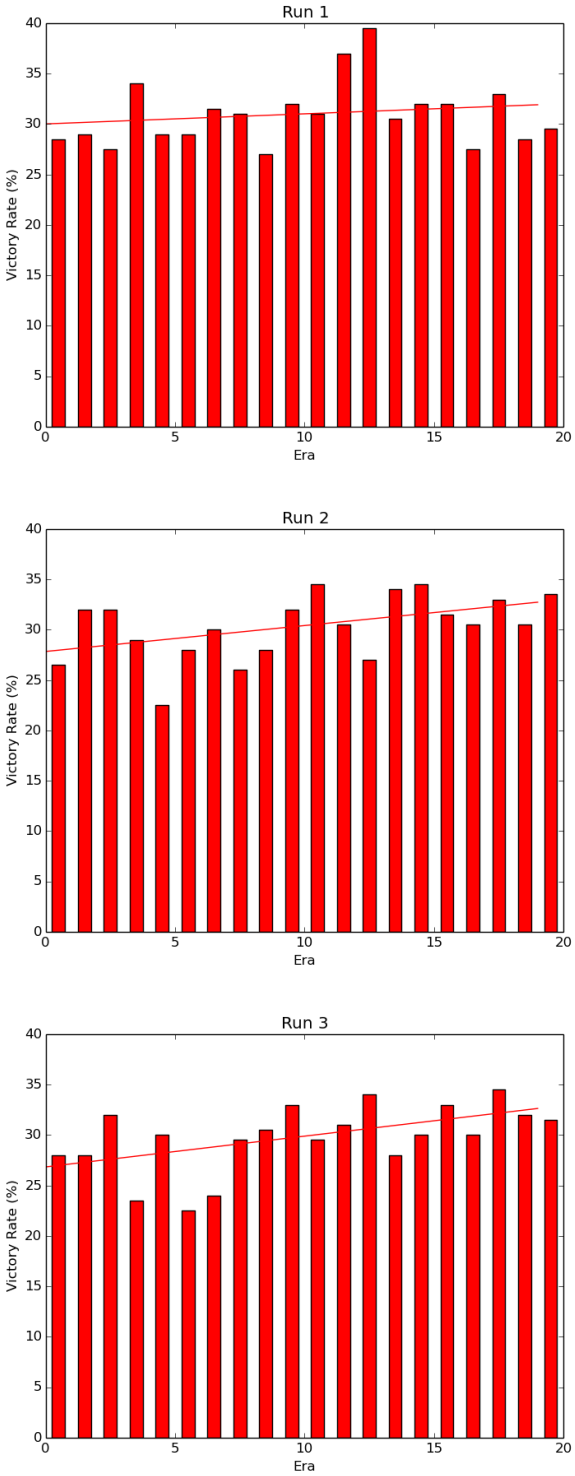

Figure 7.5: *Wins per 200 games across 4000 games, in 3 runs.*

intersection of these sets of cases was then found to be 310, indicating that this many cases were considered to have been of high quality across all three runs.

Two conclusions can be drawn from this. Firstly the total number of initial cases retained are lower than they intuitively should be, is this implies that only 28% of the skilled human player's Settle decisions were of reasonable quality, which is improbable given the human's win rate. This is likely to be due to a combination of reasons: SCOUT underutilising good cases; SCOUT's deletion policy being too strict, and a lot of cases being deleted through lack of retrieval, because SCOUT's play style is less versatile than the human's. Because SCOUT tends to play similar strategies in most matches, it may be heavily exploring a limited region of the problem space, and having no need to explore other regions because it cannot play itself into game states of those types, thus ignoring cases which cover those regions.

Secondly and more positively, the size of the intersecting set of retained cases is significantly higher than that expected by a of random selection of cases would result in, which would be approximately $\frac{781}{2947} \times \frac{891}{2947} \times \frac{773}{2947} \times 2947 = 62$. This indicates that there is some correlation between actual case quality and retention by the maintenance system. The actual number of high quality cases is probably much larger than this, however, but improvements to SCOUT's retrieval functions would likely be the first step to addressing this, as opposed to the maintenance system itself.

Directly Observed Games

Finally, during development we directly observed and analysed many hundreds of games played between SCOUT and Keldon AI. This section details results and observations of a random selection of 10 games observed during the 4th era of a run. Game 1 was selected to be observed at random, and the rest are those that occurred in sequence thereafter. It represents better than average performance by SCOUT, which won 50% of these games. These observations are limited by their subjective nature, but they are necessary in terms of gaining more insight into SCOUT's performance than raw scores and victory rates can provide. The final game states of each of these game as they are exported by the Keldon engine are available in Appendix B.

Game 1, SCOUT: 26 - Keldon AI: 23 - Won with Consume-Produce strategy against Consume-Produce Strategy. Made a questionable early decision to pass on a good placement, likely triggered by having another good card in hand which SCOUT played two turns later. Both cards could have been placed with correct hand management, however.

Game 2, SCOUT: 36 - Keldon AI: 33 - Won with Consume-Produce strategy against a tableau with several large worlds and the key card *Galactic Survey: SETI*. SCOUT played near-perfectly.

Game 3, SCOUT: 17 - Keldon AI: 25 - Lost with an incoherent tableau against Consume-Produce strategy. SCOUT's draws were sufficient that it could have played a good Develop strategy. Keldon AI played well.

Game 4, SCOUT: 30 - Keldon AI: 25 - Won with Consume-Produce strategy against an incoherent tableau. Made some poor placement decisions but won due to unfortunate draws for Keldon AI.

Game 5, SCOUT: 20 - Keldon AI: 35 - Lost with an attempted Consume-Produce strategy against a Consume-Produce strategy. SCOUT made fatal errors, repeatedly wasting goods and actions, probably due to our feature generalisation indicating that the tableau was similar to other tableaux with subtle but important differences.

Game 6, SCOUT: 25 - Keldon AI: 26 - Lost with Consume-Produce strategy against Military strategy. Both agents played reasonably.

Game 7, SCOUT: 46 - Keldon AI: 43 - Won with Military strategy against a Consume-Produce strategy. SCOUT played near-perfectly and made difficult decisions correctly, and Keldon AI played well. This was the best of the 10 games and the one that most resembled a match between skilled humans.

Game 8, SCOUT: 26 - Keldon AI: 24 - Won with a mixed strategy against a Military Strategy. SCOUT made a few poor placement decisions which led it away from an effective Consume-Produce strategy, but won because Keldon AI also played poorly.

Game 9, SCOUT: 20 - Keldon AI: 35 - Lost with a Consume-Produce strategy against Consume-Produce Strategy. SCOUT was unfortunate not to draw any of the good consumer cards. Keldon AI had good draws and played well.

Game 10, SCOUT: 26 - Keldon AI: 31 - Lost with a mixed strategy against Consume-Produce Strategy. SCOUT made a modestly effective tableau with poor draws.

It can be seen that SCOUT pursues a clear Consume-Produce strategy whenever possible, and also often when it is not possible. Since this is generally the best strategy, this is good for SCOUT's overall success rate, but occasionally damages the quality of its reasoning. Game 7 shows that when the situation is right it will pursue a military strategy very effectively.

7.2 SCOUT's Play Style

7.2.1 Strengths

Playing to a plan In direct observations of played games, SCOUT does show a tendency to play toward a fixed strategy better than the Keldon AI. When it draws sufficient cards favourable to an orthodox strategy, it usually wins the game, but it is very poor in adapting less optimal draws into an effective strategy. In particular, it has a tendency towards playing the Consume-Produce strategy, and when it has an unclear choice it will generally opt to place key cards for that strategy, and frequently selects the Consume actions. It is likely that this is due to the fact that the human player whose games form the initial case base also heavily favours the Consume-Produce strategy, and thus a majority of the cases in the case base represent decisions made with that strategy in mind. The human player can, however, evaluate game states more effectively and thus wins significantly more games.

In addition, while SCOUT tends to make choices typical of a Consume-Produce type strategy, it does not exclusively select these choices whenever they are available, and it is also able to follow other strategies when more appropriate. In particular it shows an aptitude for pursuing a military strategy when a Consume-Produce strategy is inappropriate. For example, when its tableau is characteristic of a military strategy tableau it will tend to trade its goods for cards instead of

points, as a skilled player would. That these behaviours are entirely derived from the case base is pleasing, because they are exactly the sort of behaviours that would be hard-coded in with a rule-based system, and yet they are learned dynamically from its cases. It also shows that it is able to draw more from its case bases than a simple tendency to always make the most common decisions, and is responsive to the current game state.

It seems that SCOUT plays best when it is given fewer options in the early game, when the lack of well-defined features limits its ability to find appropriate matches in the case base. Particularly when there are 2-3 cards in the tableau, there is very little to indicate case similarity, since most other features are identical in in the early stages of all games. As the game progresses, if SCOUT has had favourable draws and limited options it will find that it has a tableau very similar to one or a few in the case base and this will lead to its reasoning system becoming more effective in the later part of the game. In contrast, too many options in the early game, where the features of the game are more nebulous, can lead to an incoherent set of decisions which in turn lead to a game which has few similarities to those in the case base, undermining the usefulness of retrieved cases. Furthermore, in the early-mid game, SCOUT's strong bias towards playing a Consume-Produce type strategy can lead it away from an effective start in another type of strategy and towards a mediocre mixed strategy, if the opportunity arises.

7.2.2 Weaknesses

Evaluating the Endgame Nonetheless, SCOUT also has several major weaknesses which limit its performance. SCOUT is particularly weak in the endgame, where its inability to evaluate its options in game terms limits its ability to adapt its decisions to the fact that the game is about to end. Ideally, SCOUT would have enough cases to be able to naturally retrieve endgame cases when the current game is drawing to an end, but in practice this does not frequently occur, and it will for example take an action such as Produce in the final round of the game, which would not yield any points until the next round. This is a very poor move which only a novice player would ever make. Figure 7.3 demonstrates that in a large portion of games SCOUT loses by only 1-4 points. This is the the sort of score that can be gained by optimising play in the last turn or two with the knowledge that the game is about to end.

Opportunity Cost Another particular flaw in SCOUT's play is its inability to take into account the opportunity cost of its decisions. Suppose SCOUT is presented with two decisions, A and B, where both decisions are highly beneficial, but A is more beneficial than B. Suppose that if A is selected, then B will become unavailable; but if B is selected, then A will still be available in the next round. Unless cases representing the exact scenario are present in the case base, then SCOUT is very likely to select decision A immediately, whereas a skilled player would observe that selecting decision B first would likely be a superior decision. Without a significantly more thorough case base, it is difficult to envisage a pure CBR system resolving this issue.

Case Coverage With a more thorough coverage of the problem space, this pure CBR approach could be feasible to make a system capable of superior performance to the Keldon AI, but the problem space is vast, with 5.37×10^{14} possible variations on the player_tableau feature alone, 7.51×10^{13} possible player hands, and many more features of smaller ranges. Clearly having a complete case base is out of the question, but across many dimensions, for example between a few points in the Pool feature, the difference between many possible cases is negligible. The exact degree of coverage necessary is difficult to predict. Certainly it would be larger than even the largest case bases which we experimented with, and this would mean that retrieval times would also be an issue. Little effort has been put thus far into optimising SCOUT's retrieval

algorithm in terms of processing requirements, and the xml schema for the cases in particular are very inefficient.

Due to the inevitably limited coverage of the case base, the performance of the system relies heavily upon isolating the most pertinent features and generalising values to shrink the problem space, but this is a difficult to task to balance correctly, and also biases the system towards our own strategic considerations as the developer, which we would like to avoid. Nonetheless, there has been some success with this approach, as a case base derived from only 748 games is able to provide SCOUT with sufficient information to play the game at a reasonable level. Despite this, we do not believe that the size of the initial case base that we have available is sufficient for SCOUT's performance to reach its potential. At the outset of this project we had hoped to be able to access game data from the vast number of games which have been played online with the Keldon game engine and others, but this proved infeasible and we instead created our own cases. Keldon Jones' server previously held data of games played on his server, of which there have been 240,979 as of October 2016 [18], but only a digest of this data is available now which is insufficient for our purposes. *boardgamearena.com* has data of 3,622,822 games as of October 2016 [4]. Acquiring and parsing this data was too great a task for this project but could be a useful resource for future work. An alternative approach could be to recruit members of the online community to crowdsource the cases specifically for this purpose, as many players already store data of their own games.

Feature Weightings The hand-defined feature weightings which we used may be a weakness in SCOUT's performance, and a more focussed, methodical approach to finding optimal feature weights may be beneficial. Furthermore, despite the aim for SCOUT's reasoning to be based entirely on its case base, our hand-defined feature weights biased it towards focussing on certain features which we deemed most important, at the expense of other features which some skilled players could consider to be equally important.

Chapter 8

Future Work and Conclusions

The purpose of this dissertation was to explore the use of the CBR methodology to play a modern board game, a domain which has received comparatively little attention in game AI research, despite offering many interesting challenges. SCOUT has demonstrated that a system using CBR and only limited domain knowledge can create a feasible agent for playing RFTG. As of yet, however, it does not play at the same level as the current standard of RFTG AI, the Keldon AI, which uses more conventional search and evaluation methods. The Keldon AI is a sophisticated system which has been developed and improved over several years, and reaching its level of performance is a high benchmark. Therefore while it is disappointing that SCOUT's performance is not up to this standard, we feel that we have had some success in creating a system which can make reasonable decisions for a complete and complex game.

In attempting to create a system from scratch, which includes the capacity to reason about various types of decision and to evaluate, maintain, and improve its own case bases, we have undertaken a large project with a broad scope. This may have come at the expense of focussed optimisation of key elements. As a result, we do not believe SCOUT currently reaches the full potential of an agent using this methodology. This leaves the potential for future work in refining these aspects of the system, which could include systematically deriving feature weighting, or the development of more sophisticated retrieval algorithms. From a broader perspective, a hybrid approach which combines SCOUT's ability to recognise successful combinations of cards and make decisions in terms of a coherent strategy, combined with a system which can evaluate possible moves in the terms of game itself, such as that of the Keldon AI, may result in a system which is superior to both, and also closer to a human player's reasoning process. Combining CBR with other methodologies is a popular approach to such systems [39, 5]. SCOUT's architecture has the potential to be used as a basis for different AI agents for RFTG, as could our fork of Keldon Jones' RFTG engine, with the improved modularity of its control system.

While this dissertation focussed on training SCOUT with human player's cases in an attempt to benefit from their reasoning, it may also be interesting to experiment with automatic case elicitation as per Powell's CHEBR system [23], beginning with a small or empty initial case base. We have demonstrated, however, that a random agent is completely incapable of winning a game of RFTG, so a different approach would need to be taken in the early stages of generating the case base. In particular, an evaluation function which took more into account than the final result would be necessary. The overall performance of SCOUT3's learning functionality proved to be limited, but there is potential to adjust its parameters and tweak its deletion policies.

Most importantly, future work that aims to improve upon SCOUT's performance would require access to a much larger case base of games by skilled human players. This could open up the possibility for using data mining techniques to gain insight into feature weights, and of course give greater coverage in the initial case bases. A case base which includes negative cases to indicate potentially poor decisions to SCOUT, may also improve performance [2].

Acknowledgements

I would like to acknowledge and thank:

Associate-Professor Ian Watson for his advice and guidance during this project,

Matt and Cathy Kearse and members of boardgamegeek.com for discussion about RFTG and for playing the game with me,

Brandee Thorburn for naming Scout,

And finally Tom Lehmann for designing RFTG itself and Keldon Jones for his computer implementation and its AI system.

Appendix A

Keldon Engine Modifications

1. A socket client was added to the engine, which would gather game information and send it to the SCOUT socket server, and then wait and receive the response from SCOUT.

2. A third type of player was added. Previously a player would be indicated to be controlled by a human or by the engine's built-in AI system, and we added the further option for a player to be controlled by an external AI. At many points in code, control is switched based on the type of player. This meant that games could now be played by any pairing of the 3 different player types - controlled by human, Keldon AI, or SCOUT AI player.

3. A new function for running an n-length series of games in which one player is controlled by SCOUT and one is controlled by the Keldon AI.

4. New functionality was added for exporting game state information to a new XML document each time for each decision made during a game, in order to generate initial case bases. This is achieved by modifying the load game functions: Rather than exporting cases while a game was being played, the saved game files of successful games were retained and then reloaded through the modified functions to extract cases. This meant that the case specifications could be altered and cases could be regenerated in different ways without having to play new games.

Appendix B

End States of Directly Observed Games

The following listings display the end game states of the games observed in subsection 7.1.3. They are included for reference but are difficult to parse. An individual listing however contains sufficient data to be loaded as a savegame file into the Keldon engine to observe the game in its entirety. In every game SCOUT is represented by the player with name Player 0.

Listing B.1: Game 1

```
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>15</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>3</Deck>
    <Discard>79</Discard>
    <Pool>-6</Pool>
  </Status>
  <Player id="0" ai="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>13</Chips>
    <Score>23</Score>
    <Tableau count="5">
      <Card id="6">Old Earth</Card>
      <Card id="29">Artist Colony</Card>
      <Card id="30">Alien Robotic Factory</Card>
      <Card id="84">Radioactive World</Card>
      <Card id="26" good="yes" num_goods="1">Spice World</Card>
    </Tableau>
    <Hand count="8">
    </Hand>
  </Player>
  <Player id="1" ai="yes" winner="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>17</Chips>
    <Score>26</Score>
    <Tableau count="6">
      <Card id="8" good="yes" num_goods="1">Alpha Centauri</Card>
      <Card id="96" good="yes" num_goods="1">New Earth</Card>
      <Card id="12" num_goods="1">Public Works</Card>
      <Card id="28" num_goods="1">New Vinland</Card>
      <Card id="100" num_goods="1">Mining World</Card>
      <Card id="32" num_goods="1">Distant World</Card>
    </Tableau>
    <Hand count="10">
      <Card id="36">New Survivalists</Card>
      <Card id="53">Galactic Resort</Card>
      <Card id="58">Black Market Trading World</Card>
      <Card id="27">Lost Species Ark World</Card>
      <Card id="61">Galactic Trendsetters</Card>
      <Card id="16">Expedition Force</Card>
      <Card id="72">Terraforming Robots</Card>
      <Card id="42">Research Labs</Card>
      <Card id="45">Pan-Galactic League</Card>
      <Card id="90">Galactic Renaissance</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
4098510202
2 0
0 0 0
none
375 2 0 2 13 93 0 0 0 2 7 -1 0 5 -1 1 86 0 12 0 1 2 0 0 0 2 5 -1 0 5 109 2 109 86 0 6 0 4 52 92
    112 86 0 0 0 2 7 -1 0 12 0 1 109 0 0 0 2 3 -1 0 2 0 1 95 0 5 7 4 87 7 85 12 0 0 0 2 5 -1 0 5
    30 1 30 0 6 0 2 87 85 0 0 0 2 9 -1 0 0 0 2 7 -1 0 12 0 1 109 0 13 0 1 109 1 0 15 0 1 2 2
    109 0 13 0 1 30 1 0 15 0 1 30 2 30 0 0 0 2 9 -1 0 2 0 1 22 0 2 0 1 26 0 0 0 2 8 -1 0 5 113 3
    113 34 105 0 6 0 2 98 15 0 13 0 1 109 1 0 15 0 1 30 2 109 0 13 0 1 7 1 1 15 0 1 109 2 7 1
    15 0 1 2 2 30 0 0 0 2 5 -1 0 5 34 2 34 105 0 6 0 4 14 80 45 97 0 0 0 2 9 -1 0 2 0 1 40 0 0 0
    2 8 -1 0 13 0 1 109 1 0 15 0 1 113 2 109 0 13 0 1 7 1 1 15 0 1 30 2 7 1 15 0 1 34 2 30 0 2
    0 1 71 0 0 0 2 8 -1 0 13 0 1 109 1 0 15 0 1 109 2 109 0 13 0 1 7 1 1 15 0 1 2 2 7 1 2 0 1 27
    0 0 0 2 9 -1 0 2 0 1 94 0 2 0 2 83 99 0 0 0 2 8 -1 0 13 0 1 109 1 0 15 0 1 113 2 109 0 13 0
    1 7 1 1 15 0 1 34 2 7 1 15 0 1 30 2 30 0 2 0 3 1 24 105 0
241 2 0 2 77 39 0 0 0 2 5 -1 0 5 31 1 31 0 6 0 1 84 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 7 -1 0 12 0 1
    31 0 0 0 2 2 -1 0 2 0 1 25 0 5 -1 6 43 6 54 56 79 48 0 0 0 2 5 -1 0 5 32 1 32 0 6 0 6 43 6
    54 56 79 48 0 0 0 2 9 -1 0 0 0 2 7 -1 0 12 0 1 32 0 15 0 1 31 2 0 1 0 0 2 1 -1 0 2 0 6 37 16
    55 23 42 76 0 0 0 2 5 -1 0 5 96 5 8 96 70 100 28 0 6 0 2 51 70 0 15 0 2 96 31 2 0 1 0 0 2 5
    -1 0 5 28 3 8 100 28 0 6 0 2 19 91 0 0 0 2 2 -1 0 2 0 1 74 0 0 0 2 8 -1 0 15 0 2 32 31 2 0
    1 0 0 2 8 -1 0 15 0 1 28 2 0 1 0 0 2 1 -1 0 2 0 6 75 73 21 63 3 90 0 0 0 2 8 -1 0 15 0 2 32
    31 2 0 1
]]>
  </Save>
</RftgExport>
```

Listing B.2: Game 2

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>17</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>18</Deck>
    <Discard>66</Discard>
    <Pool>-9</Pool>
  </Status>
  <Player id="1" ai="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>11</Chips>
    <Score>33</Score>
    <Tableau count="8">
      <Card id="9">New Sparta</Card>
      <Card id="15">Comet Zone</Card>
      <Card id="17">Mining Robots</Card>
      <Card id="25">Avian Uplift Race</Card>
      <Card id="49">Galactic Survey: SETI</Card>
      <Card id="96">New Earth</Card>
      <Card id="30">Alien Robotic Factory</Card>
      <Card id="41">Mining Conglomerate</Card>
    </Tableau>
    <Hand count="10">
    </Hand>
  </Player>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>22</Chips>
    <Score>36</Score>
    <Tableau count="7">
      <Card id="6">Old Earth</Card>
      <Card id="10">Earth's Lost Colony</Card>
      <Card id="24">Contact Specialist</Card>
      <Card id="63">Star Nomad Lair</Card>
      <Card id="90">Galactic Renaissance</Card>
      <Card id="13">Gem World</Card>
      <Card id="44">Investment Credits</Card>
    </Tableau>
    <Hand count="5">
      <Card id="36">New Survivalists</Card>
      <Card id="85">Aquatic Uplift Race</Card>
      <Card id="61">Galactic Trendsetters</Card>
      <Card id="19">Export Duties</Card>
      <Card id="74">New Galactic Order</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
2710913335
2 0
0 0 0
none
312 2 0 2 99 41 0 0 0 2 5 -1 0 5 4 2 66 4 0 6 0 2 44 66 0 0 0 2 9 -1 0 0 0 2 7 -1 0 12 0 1 4 0 0
    0 2 3 -1 0 5 25 2 9 25 0 5 71 2 65 71 0 6 0 0 1 25 0 0 2 7 -1 0 12 0 1 71 0 0 0 2 9 -1 0 0 0
    2 8 -1 0 13 0 1 4 1 0 15 0 1 4 2 4 0 15 0 1 71 2 0 1 0 0 2 9 -1 0 5 103 3 21 103 9 0 6 0 6
    21 30 60 76 9 65 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1 71 0 13 0 1 4 1 0 15 0 1 4 2 4 0 0 0 2 9 -1
    0 0 0 2 7 -1 0 12 0 1 71 0 13 0 1 4 1 0 15 0 1 4 2 4 0 0 0 2 9 -1 0 0 0 2 8 -1 0 13 0 1 4 1
    0 15 0 1 4 2 4 0 15 0 1 71 2 0 1 0 0 2 9 -1 0 5 8 5 69 36 90 8 105 0 6 0 2 36 105 0 0 0 2 8
    -1 0 13 0 1 4 1 0 15 0 1 8 2 4 0 15 0 2 71 4 2 0 1 0 0 2 9 -1 0 5 51 2 17 51 0 6 0 1 90 0 0
    0 2 8 -1 0 13 0 1 4 1 0 15 0 1 8 2 4 0 15 0 2 71 4 2 0 1
280 2 0 2 46 54 0 0 0 2 5 -1 0 5 11 3 11 106 70 0 6 0 3 110 106 70 0 0 0 2 9 -1 0 0 0 2 7 -1 0 12
    0 1 11 0 0 0 2 5 -1 0 5 14 1 14 0 6 0 2 96 2 0 5 27 1 27 0 0 0 2 7 -1 0 12 0 1 27 0 0 0 2 9
    -1 0 0 0 2 7 -1 0 12 0 1 11 0 0 0 2 3 -1 0 5 57 4 56 57 80 95 0 6 0 5 56 5 40 80 74 0 0 0 2
    5 -1 0 5 109 2 63 109 0 6 0 4 111 63 29 95 0 15 0 1 11 2 109 0 0 0 2 9 -1 0 0 0 2 7 -1 0 12
    0 1 27 0 15 0 1 11 2 109 0 0 0 2 9 -1 0 0 0 2 8 -1 0 15 0 1 11 2 109 0 0 0 2 5 -1 0 5 32 4
    19 1 32 28 0 6 0 6 42 20 12 23 1 91 0 0 0 2 7 -1 0 12 0 1 27 0 15 0 1 11 2 109 0 0 0 2 3 -1
    0 5 45 4 45 55 43 112 0 6 0 2 33 86 0 0 0 2 8 -1 0 13 0 1 45 1 1 15 0 2 109 11 2 45 1 15 0 1
    32 2 109 0 2 0 1 92 0
]]>
  </Save>
</RftgExport>
```

Listing B.3: Game 3

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>12</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>29</Deck>
    <Discard>53</Discard>
    <Pool>0</Pool>
  </Status>
  <Player id="1" ai="yes" winner="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>17</Chips>
    <Score>25</Score>
    <Tableau count="6">
      <Card id="7">Epsilon Eridani</Card>
      <Card id="70">Runaway Robots</Card>
      <Card id="13" good="yes" num_goods="1">Gem World</Card>
      <Card id="40" num_goods="1">Consumer Markets</Card>
      <Card id="28" good="yes" num_goods="1">New Vinland</Card>
      <Card id="43" num_goods="1">Deficit Spending</Card>
    </Tableau>
    <Hand count="8">
    </Hand>
  </Player>
  <Player id="0" ai="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>7</Chips>
    <Score>17</Score>
    <Tableau count="6">
      <Card id="6">Old Earth</Card>
      <Card id="23">Space Marines</Card>
      <Card id="67" good="yes" num_goods="1">Reptilian Uplift Race</Card>
      <Card id="77" good="yes" num_goods="1">Secluded World</Card>
      <Card id="100" good="yes" num_goods="1">Mining World</Card>
      <Card id="80" num_goods="1">Replicant Robots</Card>
    </Tableau>
    <Hand count="7">
      <Card id="29">Artist Colony</Card>
      <Card id="89">Destroyed World</Card>
      <Card id="93">Expanding Colony</Card>
      <Card id="10">Earth's Lost Colony</Card>
      <Card id="27">Lost Species Ark World</Card>
      <Card id="33">Rebel Outpost</Card>
      <Card id="72">Terraforming Robots</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
3618697623
2 0
0 0 0
none
195 2 0 2 34 74 0 0 0 2 3 -1 0 5 24 1 24 0 6 0 1 3 0 0 0 2 2 -1 0 2 0 1 96 0 0 0 2 5 -1 0 5 75 3
    88 72 75 0 15 0 1 75 2 0 1 0 0 2 5 -1 0 5 88 3 2 88 72 0 6 0 1 2 0 0 0 2 7 -1 0 12 0 1 88 0
    0 0 2 9 -1 0 5 -1 1 82 0 0 0 2 8 -1 0 13 0 1 0 1 1 15 0 2 88 75 2 0 1 0 0 2 7 -1 0 12 0 1 88
     0 0 0 2 5 -1 0 5 113 3 31 113 72 0 6 0 3 57 87 40 0 0 0 2 7 -1 0 12 0 1 88 0 0 0 2 3 -1 0 5
     91 3 91 9 82 0 6 0 3 93 9 72 0 13 0 1 0 1 1 15 0 2 113 88 2 0 1 0 0 2 9 -1 0
293 2 0 2 103 17 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 2 -1 0 2 0 1 67 0 0 0 2 8 -1 0 5 78 3 78 63 8 0 15
     0 1 78 2 1 1 0 0 2 9 -1 0 5 8 2 63 8 0 6 0 2 15 44 0 0 0 2 7 -1 0 12 0 1 78 0 15 0 1 8 2 1
    1 0 0 2 3 -1 0 5 43 4 6 55 54 43 0 6 0 4 94 6 55 16 0 0 0 2 9 -1 0 13 0 1 1 1 1 15 0 1 8 2 1
     1 0 0 2 9 -1 0 13 0 1 43 1 0 15 0 1 8 2 43 0 15 0 1 78 2 1 1 2 0 4 86 63 20 39 0 0 0 2 5 -1
     0 5 30 3 30 60 28 0 6 0 2 52 48 0 0 0 2 9 -1 0 13 0 1 43 1 0 15 0 1 8 2 43 0 13 0 1 1 1 1
    15 0 1 78 2 1 1 2 0 3 19 71 37 0 0 0 2 8 -1 0 5 50 5 50 85 53 46 54 0 6 0 2 85 54 0 13 0 1
    43 1 0 15 0 2 30 8 2 43 0 13 0 1 1 1 1 15 0 1 78 2 1 1 13 0 1 50 1 0 14 0 1 46 0 0 0 2 7 -1
    0 13 0 1 50 1 0 14 0 2 28 27 0
]]>
  </Save>
</RftgExport>
```

Listing B.4: Game 4

```
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>13</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>28</Deck>
    <Discard>52</Discard>
    <Pool>-9</Pool>
  </Status>
  <Player id="1" ai="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>14</Chips>
    <Score>25</Score>
    <Tableau count="11">
      <Card id="9">New Sparta</Card>
      <Card id="20">Former Penal Colony</Card>
      <Card id="37">Outlaw World</Card>
      <Card id="75">Asteroid Belt</Card>
      <Card id="51">Refugee World</Card>
      <Card id="64">The Last of the Uplift Gnarssh</Card>
      <Card id="28">New Vinland</Card>
      <Card id="12">Public Works</Card>
      <Card id="57">Galactic Engineers</Card>
      <Card id="60">Tourist World</Card>
      <Card id="18">Rebel Miners</Card>
    </Tableau>
    <Hand count="4">
    </Hand>
  </Player>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>19</Chips>
    <Score>30</Score>
    <Tableau count="10">
      <Card id="8" good="yes" num_goods="1">Alpha Centauri</Card>
      <Card id="10" num_goods="1">Earth's Lost Colony</Card>
      <Card id="7" num_goods="1">Epsilon Eridani</Card>
      <Card id="77" num_goods="1">Secluded World</Card>
      <Card id="89" num_goods="1">Destroyed World</Card>
      <Card id="72" num_goods="1">Terraforming Robots</Card>
      <Card id="100" num_goods="1">Mining World</Card>
      <Card id="22" num_goods="1">New Military Tactics</Card>
      <Card id="15" good="yes" num_goods="1">Comet Zone</Card>
      <Card id="29" num_goods="1">Artist Colony</Card>
    </Tableau>
    <Hand count="7">
      <Card id="54">Pre-Sentient Race</Card>
      <Card id="84">Radioactive World</Card>
      <Card id="34">Rebel Warrior Race</Card>
      <Card id="12">Public Works</Card>
      <Card id="39">Diversified Economy</Card>
      <Card id="49">Galactic Survey: SETI</Card>
      <Card id="76">Merchant Guild</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
3684293614
2 0
0 0 0
none
363 2 0 2 26 10 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1 2 0 0 0 2 5 -1 0 5 4 1 4 0 6 0 2 51 45 0 0 0 2 5
    -1 0 5 1 1 1 0 6 0 2 43 56 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1 2 0 13 0 1 1
     1 1 15 0 1 4 2 1 1 0 0 2 5 -1 0 5 88 2 102 88 0 6 0 1 92 0 0 0 2 9 -1 0 5 102 1 102 0 6 0 0
     0 0 0 2 8 -1 0 5 82 2 18 82 0 6 0 3 37 18 85 0 13 0 1 1 1 1 15 0 1 4 2 1 1 13 0 1 4 1 0 15
    0 1 88 2 4 0 13 0 1 88 1 0 15 0 1 2 2 88 0 15 0 1 102 2 82 1 0 0 2 9 -1 0 5 113 1 113 0 6 0
    2 20 27 0 19 0 1 2 0 0 0 2 8 -1 0 13 0 1 1 1 1 15 0 1 4 2 1 1 13 0 1 4 1 0 15 0 1 113 2 4 0
    13 0 1 88 1 0 15 0 1 88 2 88 0 15 0 1 2 2 82 1 0 0 2 3 -1 0 5 22 4 80 14 87 22 0 5 11 2 11
    31 0 6 0 2 80 14 0 0 0 2 9 -1 0 5 31 1 31 0 6 0 1 40 0 19 0 1 2 0 0 0 2 8 -1 0 13 0 1 1 1 1
    15 0 1 4 2 1 1 13 0 1 4 1 0 15 0 1 88 2 4 0 13 0 1 88 1 0 15 0 1 31 2 88 0 15 0 1 113 2 82 1
286 2 0 2 108 48 0 0 0 2 5 -1 0 5 19 3 86 19 39 0 0 0 2 5 -1 0 5 39 2 86 39 0 0 0 2 8 -1 0 5 -1 1
     86 0 15 0 1 19 2 39 0 0 0 2 5 -1 0 5 86 1 86 0 6 0 2 101 46 0 0 0 2 5 -1 0 5 59 1 59 0 6 0
    0 0 15 0 1 59 2 39 0 0 0 2 5 -1 0 5 72 2 72 30 0 0 0 2 5 -1 0 5 30 2 100 30 0 6 0 2 100 112
    0 0 0 2 3 -1 0 5 7 1 7 0 13 0 1 39 1 0 15 0 1 86 2 39 0 13 0 1 30 1 0 15 0 1 30 2 30 0 15 0
    1 72 2 7 1 0 0 2 5 -1 0 5 65 1 65 0 6 0 2 74 47 0 19 0 1 86 0 0 0 2 7 -1 0 12 0 1 86 0 13 0
    1 39 1 0 15 0 1 30 2 39 0 0 0 2 5 -1 0 5 -1 0 0 5 68 6 16 71 68 5 90 109 0 6 0 4 71 110 90
    109 0 0 0 2 5 -1 0 5 16 2 16 5 0 19 0 1 86 0 0 0 2 8 -1 0 13 0 1 39 1 0 15 0 1 16 2 39 0 13
    0 1 68 1 0 15 0 2 30 86 2 68 0
]]>
  </Save>
</RftgExport>
```

Listing B.5: Game 5

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>14</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>22</Deck>
    <Discard>60</Discard>
    <Pool>-7</Pool>
  </Status>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>21</Chips>
    <Score>35</Score>
    <Tableau count="8">
      <Card id="7">Epsilon Eridani</Card>
      <Card id="63">Star Nomad Lair</Card>
      <Card id="13">Gem World</Card>
      <Card id="37">Outlaw World</Card>
      <Card id="100">Mining World</Card>
      <Card id="15" good="yes" num_goods="1">Comet Zone</Card>
      <Card id="96" good="yes" num_goods="1">New Earth</Card>
      <Card id="27" good="yes" num_goods="1">Lost Species Ark World</Card>
    </Tableau>
    <Hand count="10">
    </Hand>
  </Player>
  <Player id="1" ai="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>10</Chips>
    <Score>20</Score>
    <Tableau count="7">
      <Card id="8">Alpha Centauri</Card>
      <Card id="11">Rebel Fuel Cache</Card>
      <Card id="87">Bio-Hazard Mining World</Card>
      <Card id="40">Consumer Markets</Card>
      <Card id="81">Pilgrimage World</Card>
      <Card id="93">Expanding Colony</Card>
      <Card id="77">Secluded World</Card>
    </Tableau>
    <Hand count="4">
      <Card id="64">The Last of the Uplift Gnarssh</Card>
      <Card id="18">Rebel Miners</Card>
      <Card id="35">Rebel Underground</Card>
      <Card id="16">Expedition Force</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
1817748500
2 0
0 0 0
none
248 2 0 2 67 43 0 0 0 2 7 -1 0 5 5 1 5 0 12 0 1 2 0 0 0 2 7 -1 0 5 100 1 100 0 6 0 2 91 15 0 12 0
      1 5 0 0 0 2 3 -1 0 5 44 3 50 44 24 0 6 0 4 50 89 73 24 0 5 93 1 93 0 6 0 0 0 0 0 0 2 9 -1 0
     19 0 1 2 0 0 0 2 8 -1 0 15 0 2 100 2 2 93 0 0 0 2 9 -1 0 5 -1 0 0 19 0 1 2 0 0 0 2 8 -1 0 15
      0 2 100 2 2 93 0 0 0 2 9 -1 0 5 -1 0 0 19 0 1 2 0 0 0 2 8 -1 0 15 0 2 100 2 2 93 0 0 0 2 2
     -1 0 2 0 1 62 0 5 106 1 106 0 6 0 1 41 0 0 0 2 2 -1 0 2 0 1 69 0 0 0 2 2 -1 0 2 0 1 96 0 0 0
      2 5 -1 0 5 88 1 88 0 6 0 1 90 0 0 0 2 8 -1 0 13 0 1 44 1 0 15 0 1 88 2 44 0 13 0 1 106 1 0
     15 0 1 100 2 106 0
315 2 0 2 26 99 0 0 0 2 5 -1 0 5 71 3 39 8 71 0 15 0 1 71 2 1 1 0 0 2 5 -1 0 5 8 3 33 39 8 0 6 0
     2 33 85 0 0 0 2 5 -1 0 5 -1 0 0 5 39 1 39 0 0 0 2 9 -1 0 0 0 2 8 -1 0 13 0 1 1 1 1 15 0 1 8
     2 1 1 15 0 1 71 2 39 0 0 0 2 5 -1 0 5 113 3 113 78 63 0 6 0 3 46 63 97 0 0 0 2 8 -1 0 13 0 1
      1 1 1 15 0 1 113 2 1 1 15 0 1 8 2 39 0 0 0 2 5 -1 0 5 11 4 60 29 11 78 0 6 0 3 76 14 78 0 0
      0 2 8 -1 0 13 0 1 1 1 1 15 0 1 8 2 1 1 15 0 1 113 2 39 0 0 0 2 5 -1 0 2 0 2 110 42 0 5 109
     6 109 108 0 4 60 29 0 6 0 5 108 0 23 55 84 0 0 0 2 8 -1 0 2 0 2 54 83 0 13 0 1 1 1 1 15 0 1
     11 2 1 1 0 0 2 8 -1 0 2 0 2 31 21 0 0 0 2 9 -1 0 5 29 4 64 4 60 29 0 6 0 5 87 22 77 4 27 0 0
      0 2 8 -1 0 13 0 1 1 1 1 15 0 1 113 2 1 1 13 0 1 39 1 0 15 0 1 8 2 39 0 15 0 1 71 2 109 0
]]>
  </Save>
</RftgExport>
```

Listing B.6: Game 6

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>11</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>19</Deck>
    <Discard>61</Discard>
    <Pool>-1</Pool>
  </Status>
  <Player id="1" ai="yes" winner="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>13</Chips>
    <Score>26</Score>
    <Tableau count="7">
      <Card id="10" good="yes" num_goods="1">Earth's Lost Colony</Card>
      <Card id="23" num_goods="1">Space Marines</Card>
      <Card id="65" num_goods="1">Alien Robot Sentry</Card>
      <Card id="13" good="yes" num_goods="1">Gem World</Card>
      <Card id="67" num_goods="1">Reptilian Uplift Race</Card>
      <Card id="53" num_goods="1">Galactic Resort</Card>
      <Card id="35" num_goods="1">Rebel Underground</Card>
    </Tableau>
    <Hand count="10">
    </Hand>
  </Player>
  <Player id="0" ai="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>12</Chips>
    <Score>25</Score>
    <Tableau count="7">
      <Card id="7">Epsilon Eridani</Card>
      <Card id="72">Terraforming Robots</Card>
      <Card id="77" good="yes" num_goods="1">Secluded World</Card>
      <Card id="55" good="yes" num_goods="1">Deserted Alien Outpost</Card>
      <Card id="28" good="yes" num_goods="1">New Vinland</Card>
      <Card id="60" num_goods="1">Tourist World</Card>
      <Card id="27" good="yes" num_goods="1">Lost Species Ark World</Card>
    </Tableau>
    <Hand count="4">
      <Card id="84">Radioactive World</Card>
      <Card id="31">Plague World</Card>
      <Card id="66">Pirate World</Card>
      <Card id="17">Mining Robots</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
1384843456
2 0
0 0 0
none
256 2 0 2 43 50 0 0 0 0 2 2 -1 0 2 0 1 17 0 5 82 3 82 46 83 0 6 0 3 94 65 83 0 0 0 0 2 5 -1 0 2 0 1
    111 0 5 88 1 88 0 6 0 1 35 0 0 0 0 2 2 -1 0 2 0 1 40 0 0 0 0 2 9 -1 0 0 0 0 2 7 -1 0 5 63 2 63 105
     0 6 0 4 76 85 105 46 0 12 0 1 63 0 13 0 1 1 1 1 1 15 0 1 88 2 1 1 0 0 0 2 5 -1 0 2 0 1 25 0 5
    30 7 2 106 60 89 30 68 66 0 6 0 2 106 66 0 0 0 0 2 9 -1 0 5 -1 0 0 0 0 0 2 5 -1 0 5 68 5 70 2 60
     89 68 0 6 0 4 70 77 2 89 0 13 0 1 68 1 0 15 0 2 30 88 2 68 0 13 0 1 30 1 0 15 0 1 63 2 30 0
     0 0 2 9 -1 0 0 0 0 2 8 -1 0 5 29 3 59 29 60 0 6 0 5 92 99 59 60 27 0 13 0 1 1 1 1 1 15 0 1 88 2
     1 1 13 0 1 68 1 0 15 0 2 30 63 2 68 0 0 0 0 2 9 -1 0
265 2 0 2 57 87 0 0 0 0 2 3 -1 0 2 0 1 67 0 5 23 2 14 23 0 6 0 1 14 0 0 0 0 2 1 -1 0 2 0 6 97 58 113
    107 24 31 0 5 73 2 73 3 0 0 0 0 2 7 -1 0 2 0 1 55 0 12 0 1 73 0 0 0 0 2 9 -1 0 0 0 0 2 5 -1 0 5 8
    6 78 8 62 32 101 3 0 6 0 2 51 101 0 15 0 1 4 2 4 0 0 0 0 2 2 -1 0 2 0 1 102 0 5 75 6 61 75 78
    62 32 3 0 0 0 0 2 3 -1 0 5 21 2 47 21 0 0 0 0 2 8 -1 0 5 61 6 61 78 62 32 37 3 0 6 0 3 103 62 3
    0 13 0 1 61 1 0 15 0 1 8 2 61 0 15 0 1 4 2 4 0 0 0 0 2 8 -1 0 13 0 1 61 1 0 15 0 1 61 2 61 0
    15 0 1 75 2 4 0 0 0 0 2 5 -1 0 5 37 3 78 32 37 0 6 0 0 1 21 13 0 1 61 1 0 15 0 1 8 2 61 0 15 0
     1 4 2 4 0 0 0 0 2 8 -1 0 13 0 1 61 1 0 15 0 1 73 2 61 0 2 0 1 22 0
]]>
  </Save>
</RftgExport>
```

Listing B.7: Game 7

```
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>19</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>2</Deck>
    <Discard>82</Discard>
    <Pool>-5</Pool>
  </Status>
  <Player id="1" ai="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>28</Chips>
    <Score>43</Score>
    <Tableau count="8">
      <Card id="10">Earth's Lost Colony</Card>
      <Card id="15">Comet Zone</Card>
      <Card id="87">Bio-Hazard Mining World</Card>
      <Card id="17">Mining Robots</Card>
      <Card id="56" good="yes" num_goods="1">Deserted Alien Colony</Card>
      <Card id="59" num_goods="1">Merchant World</Card>
      <Card id="60" num_goods="1">Tourist World</Card>
      <Card id="13" num_goods="1">Gem World</Card>
    </Tableau>
    <Hand count="1">
    </Hand>
  </Player>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>1</Chips>
    <Score>46</Score>
    <Tableau count="12">
      <Card id="8" good="yes" num_goods="1">Alpha Centauri</Card>
      <Card id="91" num_goods="1">Blaster Gem Mines</Card>
      <Card id="73" num_goods="1">Drop Ships</Card>
      <Card id="63" num_goods="1">Star Nomad Lair</Card>
      <Card id="18" good="yes" num_goods="1">Rebel Miners</Card>
      <Card id="49" num_goods="1">Galactic Survey: SETI</Card>
      <Card id="98" num_goods="1">Galactic Imperium</Card>
      <Card id="21" num_goods="1">Malevolent Lifeforms</Card>
      <Card id="82" num_goods="1">Rebel Homeworld</Card>
      <Card id="33" num_goods="1">Rebel Outpost</Card>
      <Card id="100" good="yes" num_goods="1">Mining World</Card>
      <Card id="96" num_goods="1">New Earth</Card>
    </Tableau>
    <Hand count="5">
      <Card id="51">Refugee World</Card>
      <Card id="29">Artist Colony</Card>
      <Card id="70">Runaway Robots</Card>
      <Card id="34">Rebel Warrior Race</Card>
      <Card id="80">Replicant Robots</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
3276706053
2 0
0 0 0
none
295 2 0 2 87 18 0 0 0 2 7 -1 0 5 -1 1 104 0 12 0 1 2 0 0 0 2 5 -1 0 5 104 2 60 104 0 6 0 2 26 3 0
 0 0 2 7 -1 0 12 0 1 104 0 0 0 2 3 -1 0 5 84 3 47 21 84 0 6 0 3 47 102 21 0 0 0 2 5 -1 0 5
71 4 20 16 60 71 0 0 0 2 7 -1 0 12 0 1 71 0 0 0 2 5 -1 0 5 16 5 65 90 20 16 60 0 0 0 2 3 -1
0 5 57 3 58 6 57 0 6 0 5 58 65 6 90 60 0 0 0 2 7 -1 0 12 0 1 16 0 0 0 2 9 -1 0 19 0 1 2 0 0
0 2 7 -1 0 12 0 1 16 0 0 0 2 3 -1 0 5 111 4 111 51 92 25 0 6 0 5 70 30 51 92 25 0 0 0 2 5 -1
 0 5 20 1 20 0 0 0 2 5 -1 0 5 94 1 94 0 0 0 2 7 -1 0 12 0 1 20 0 0 0 2 5 -1 0 5 35 2 1 35 0
0 0 2 7 -1 0 12 0 1 20 0 0 0 2 5 -1 0 5 113 5 59 31 74 113 1 0 6 0 2 74 1 0 0 0 2 5 -1 0 5
109 4 78 109 59 31 0 6 0 4 44 12 46 99 0 15 0 1 20 2 109 0
415 2 0 2 42 101 0 0 0 2 5 -1 0 5 11 2 105 11 0 6 0 3 105 7 9 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 8 -1
 0 15 0 1 4 2 4 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1 11 0 0 0 2 9 -1 0 5 100 1 100 0 6 0 3 95 43 41
 0 0 0 2 8 -1 0 15 0 1 4 2 4 0 0 0 2 8 -1 0 5 -1 0 0 15 0 1 11 2 4 0 0 0 2 9 -1 0 5 -1 0 0 0
0 2 7 -1 0 12 0 1 11 0 15 0 1 4 2 4 0 0 0 2 9 -1 0 0 0 2 8 -1 0 15 0 1 4 2 4 0 0 0 2 7 -1 0
5 15 2 83 15 0 6 0 2 83 5 0 12 0 1 11 0 15 0 1 100 2 4 0 0 0 2 9 -1 0 5 64 4 106 62 33 64 0
6 0 5 106 62 10 40 33 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1 64 0 15 0 1 4 2 4 0 0 0 2 9 -1 0 15 0
1 11 2 4 0 0 0 2 7 -1 0 5 67 3 67 32 66 0 6 0 4 72 110 39 80 0 12 0 1 64 0 13 0 1 4 1 0 15 0
 1 11 2 4 0 13 0 1 67 1 1 14 0 2 14 98 0 0 0 2 7 -1 0 12 0 1 100 0 13 0 1 4 1 0 15 0 1 4 2 4
0 13 0 1 67 1 1 14 0 2 0 66 0 2 0 1 24 0 0 0 2 9 -1 0 5 68 3 8 68 32 0 6 0 4 55 27 53 38 0
0 0 2 8 -1 0 5 8 1 8 0 6 0 2 81 32 0 13 0 1 4 1 0 15 0 1 11 2 4 0 13 0 1 68 1 0 15 0 2 100 4
 2 68 0 13 0 1 67 1 1 14 0 2 22 112 0
]]>
  </Save>
</RftgExport>
```

Listing B.8: Game 8

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>13</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>19</Deck>
    <Discard>61</Discard>
    <Pool>6</Pool>
  </Status>
  <Player id="0" ai="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>9</Chips>
    <Score>24</Score>
    <Tableau count="9">
      <Card id="7">Epsilon Eridani</Card>
      <Card id="95">Prosperous World</Card>
      <Card id="63">Star Nomad Lair</Card>
      <Card id="37">Outlaw World</Card>
      <Card id="78">Imperium Armaments World</Card>
      <Card id="25">Avian Uplift Race</Card>
      <Card id="96">New Earth</Card>
      <Card id="9">New Sparta</Card>
      <Card id="69" good="yes" num_goods="1">Alien Robot Scout Ship</Card>
    </Tableau>
    <Hand count="6">
    </Hand>
  </Player>
  <Player id="1" ai="yes" winner="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>9</Chips>
    <Score>26</Score>
    <Tableau count="12">
      <Card id="10">Earth's Lost Colony</Card>
      <Card id="84">Radioactive World</Card>
      <Card id="12">Public Works</Card>
      <Card id="27" good="yes" num_goods="1">Lost Species Ark World</Card>
      <Card id="77" num_goods="1">Secluded World</Card>
      <Card id="72" num_goods="1">Terraforming Robots</Card>
      <Card id="51" num_goods="1">Refugee World</Card>
      <Card id="57" num_goods="1">Galactic Engineers</Card>
      <Card id="28" num_goods="1">New Vinland</Card>
      <Card id="41" num_goods="1">Mining Conglomerate</Card>
      <Card id="93" num_goods="1">Expanding Colony</Card>
      <Card id="81" num_goods="1">Pilgrimage World</Card>
    </Tableau>
    <Hand count="5">
      <Card id="29">Artist Colony</Card>
      <Card id="55">Deserted Alien Outpost</Card>
      <Card id="33">Rebel Outpost</Card>
      <Card id="56">Deserted Alien Colony</Card>
      <Card id="43">Deficit Spending</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
1921927157
2 0
0 0 0
none
340 2 0 2 68 67 0 0 0 2 5 -1 0 2 0 1 15 0 5 96 2 96 59 0 6 0 2 85 43 0 0 0 2 7 -1 0 12 0 1 96 0 0
      0 2 7 -1 0 12 0 1 4 0 0 0 2 3 -1 0 5 7 5 44 95 7 58 13 0 5 29 2 29 59 0 6 0 5 44 95 97 58
      13 0 0 0 2 9 -1 0 0 0 2 7 -1 0 5 88 2 88 59 0 6 0 1 94 0 12 0 1 29 0 13 0 1 4 1 0 15 0 1 4 2
      4 0 13 0 1 7 1 1 15 0 1 96 2 7 1 0 0 2 3 -1 0 5 81 2 22 81 0 6 0 2 75 36 0 5 59 1 59 0 6 0
      0 0 0 0 2 5 -1 0 5 65 1 65 0 6 0 2 61 22 0 0 0 2 1 -1 0 2 0 6 34 24 12 79 100 87 0 13 0 1 4
      1 0 15 0 1 59 2 4 0 0 0 2 9 -1 0 5 30 1 30 0 6 0 2 101 54 0 19 0 1 96 0 0 0 2 8 -1 0 13 0 1
      4 1 0 15 0 1 4 2 4 0 13 0 1 7 1 1 15 0 1 88 2 7 1 13 0 1 81 1 1 15 0 1 96 2 81 1 13 0 1 30 1
      0 15 0 1 30 2 30 0 15 0 1 59 2 88 0 0 0 2 3 -1 0 5 45 3 49 45 48 0 6 0 2 72 48 0 5 106 2
      106 63 0 6 0 1 19 0 0 0 2 5 -1 0 5 93 1 93 0 6 0 0 0
263 2 0 2 33 37 0 0 0 2 2 -1 0 2 0 1 105 0 5 108 3 109 70 108 0 6 0 3 23 110 70 0 0 0 2 9 -1 0 0
      0 2 8 -1 0 13 0 1 1 1 1 15 0 1 108 2 1 1 0 0 2 5 -1 0 5 -1 0 0 5 71 1 71 0 0 0 2 7 -1 0 12 0
      1 71 0 0 0 2 5 -1 0 5 39 2 39 109 0 13 0 1 1 1 15 0 1 108 2 1 1 0 0 2 5 -1 0 5 -1 2 55 82
      0 5 89 3 89 69 109 0 6 0 4 69 55 82 74 0 0 0 2 5 -1 0 5 27 1 27 0 0 0 2 7 -1 0 2 0 2 14 111
      0 12 0 1 27 0 0 0 2 5 -1 0 5 109 3 60 28 109 0 6 0 5 98 53 112 46 41 0 0 0 2 8 -1 0 13 0 1
      1 1 1 15 0 1 108 2 1 1 13 0 1 39 1 0 15 0 1 89 2 39 0 13 0 1 109 1 0 15 0 1 109 2 109 0 0 0
      2 5 -1 0 5 -1 1 25 0 5 3 4 2 3 60 28 0 0 0 2 5 -1 0 5 77 5 20 2 77 60 28 0
]]>
  </Save>
</RftgExport>
```

Listing B.9: Game 9

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>11</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>28</Deck>
    <Discard>60</Discard>
    <Pool>-4</Pool>
  </Status>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>19</Chips>
    <Score>35</Score>
    <Tableau count="8">
      <Card id="6">Old Earth</Card>
      <Card id="54">Pre-Sentient Race</Card>
      <Card id="96">New Earth</Card>
      <Card id="58">Black Market Trading World</Card>
      <Card id="100">Mining World</Card>
      <Card id="95">Prosperous World</Card>
      <Card id="27">Lost Species Ark World</Card>
      <Card id="86">Genetics Lab</Card>
    </Tableau>
    <Hand count="5">
    </Hand>
  </Player>
  <Player id="1" ai="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>9</Chips>
    <Score>20</Score>
    <Tableau count="8">
      <Card id="10" good="yes" num_goods="1">Earth's Lost Colony</Card>
      <Card id="70" num_goods="1">Runaway Robots</Card>
      <Card id="41" num_goods="1">Mining Conglomerate</Card>
      <Card id="13" num_goods="1">Gem World</Card>
      <Card id="15" num_goods="1">Comet Zone</Card>
      <Card id="29" good="yes" num_goods="1">Artist Colony</Card>
      <Card id="51" good="yes" num_goods="1">Refugee World</Card>
      <Card id="39" num_goods="1">Diversified Economy</Card>
    </Tableau>
    <Hand count="2">
      <Card id="36">New Survivalists</Card>
      <Card id="93">Expanding Colony</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
2205377102
2 0
0 0 0
none
244 2 0 2 63 92 0 0 0 2 3 -1 0 5 21 1 21 0 5 78 3 97 39 78 0 6 0 0 1 21 0 0 2 7 -1 0 12 0 1 78 0
     0 0 2 3 -1 0 5 46 2 50 46 0 6 0 2 50 39 0 5 -1 0 0 0 0 2 9 -1 0 0 0 2 7 -1 0 5 8 1 8 0 6 0 2
      85 97 0 12 0 1 78 0 15 0 1 4 2 4 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 8 -1 0 13 0 1 4 1 0 15 0 1 8
      2 4 0 15 0 1 78 2 46 1 0 0 2 5 -1 0 5 11 2 93 11 0 6 0 3 93 43 58 0 0 0 2 5 -1 0 5 31 2 31
     59 0 6 0 1 112 0 13 0 1 4 1 0 15 0 1 8 2 4 0 15 0 1 11 2 46 1 0 0 2 5 -1 0 5 59 1 59 0 6 0 0
      0 0 0 2 3 -1 0 5 41 3 41 23 99 0 6 0 3 32 23 99 0 13 0 1 4 1 0 15 0 1 8 2 4 0 15 0 1 11 2
     46 1
316 2 0 2 16 19 0 0 0 2 5 -1 0 5 -1 0 0 5 62 1 62 0 6 0 2 53 76 0 0 0 2 7 -1 0 12 0 1 62 0 0 0 2
     5 -1 0 5 9 3 55 9 47 0 6 0 2 55 47 0 5 109 2 66 109 0 6 0 0 1 9 0 0 2 9 -1 0 0 0 2 5 -1 0 5
     66 1 66 0 6 0 3 40 72 36 0 13 0 1 66 1 0 12 0 1 62 0 13 0 1 109 1 0 15 0 1 109 2 109 0 0 0 2
      5 -1 0 5 113 2 64 113 0 6 0 3 64 54 91 0 0 0 2 8 -1 0 13 0 1 109 1 0 15 0 1 113 2 109 0 13
     0 1 66 1 0 12 0 1 109 0 0 0 2 9 -1 0 5 108 2 70 108 0 6 0 3 70 6 14 0 0 0 2 8 -1 0 5 -1 0 0
     13 0 1 0 1 1 15 0 2 108 113 2 0 1 13 0 1 109 1 0 15 0 1 62 2 109 0 13 0 1 108 1 0 15 0 1 109
      2 108 0 0 0 2 9 -1 0 5 29 1 29 0 6 0 5 48 110 95 52 18 0 0 0 2 8 -1 0 5 98 1 98 0 6 0 2 89
     74 0 13 0 1 0 1 1 15 0 2 113 62 2 0 1 13 0 1 109 1 0 15 0 1 108 2 109 0 13 0 1 66 1 0 12 0 1
      29 0 15 0 1 109 2 108 0
]]>
  </Save>
</RftgExport>
```

Listing B.10: Game 10

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="xml" type="text/xsl"?>
<RftgExport>
  <Version>0.9.4</Version>
  <Server>local</Server>
  <PlayerName>Player 0</PlayerName>
  <Setup>
    <Players>2</Players>
    <Expansion id="0">Base game only</Expansion>
  </Setup>
  <Status>
    <Message></Message>
    <Round>13</Round>
    <GameOver />
    <Phases>
    </Phases>
    <Deck>37</Deck>
    <Discard>55</Discard>
    <Pool>-4</Pool>
  </Status>
  <Player id="0" ai="yes" winner="yes">
    <Name>Player 1</Name>
    <Actions>
    </Actions>
    <Chips>16</Chips>
    <Score>31</Score>
    <Tableau count="10">
      <Card id="6">Old Earth</Card>
      <Card id="89">Destroyed World</Card>
      <Card id="53">Galactic Resort</Card>
      <Card id="23">Space Marines</Card>
      <Card id="25">Avian Uplift Race</Card>
      <Card id="36">New Survivalists</Card>
      <Card id="72">Terraforming Robots</Card>
      <Card id="77">Secluded World</Card>
      <Card id="70">Runaway Robots</Card>
      <Card id="55">Deserted Alien Outpost</Card>
    </Tableau>
    <Hand count="2">
    </Hand>
  </Player>
  <Player id="1" ai="yes">
    <Name>Player 0</Name>
    <Actions>
    </Actions>
    <Chips>12</Chips>
    <Score>26</Score>
    <Tableau count="8">
      <Card id="7">Epsilon Eridani</Card>
      <Card id="10">Earth's Lost Colony</Card>
      <Card id="8">Alpha Centauri</Card>
      <Card id="49">Galactic Survey: SETI</Card>
      <Card id="71">Interstellar Bank</Card>
      <Card id="15">Comet Zone</Card>
      <Card id="54" good="yes" num_goods="1">Pre-Sentient Race</Card>
      <Card id="84" num_goods="1">Radioactive World</Card>
    </Tableau>
    <Hand count="1">
      <Card id="80">Replicant Robots</Card>
    </Hand>
  </Player>
  <Save>
<![CDATA[
0.9.4
2792765300
2 0
0 0 0
none
301 2 0 2 46 43 0 0 0 2 2 -1 0 2 0 1 34 0 5 4 3 100 4 2 0 6 0 2 100 49 0 0 0 2 5 -1 0 5 2 1 2 0 6
      0 2 22 110 0 13 0 1 1 1 1 15 0 1 2 2 1 1 0 0 2 9 -1 0 5 -1 0 0 0 0 2 7 -1 0 5 -1 0 0 12 0 1
      2 0 13 0 1 1 1 1 15 0 1 4 2 1 1 0 0 2 3 -1 0 5 57 3 45 14 57 0 6 0 5 45 14 60 16 76 0 5 -1
     0 0 0 0 2 9 -1 0 5 -1 0 0 0 0 2 7 -1 0 12 0 1 2 0 13 0 1 1 1 1 15 0 1 4 2 1 1 0 0 2 9 -1 0 0
      0 2 7 -1 0 12 0 1 2 0 13 0 1 1 1 1 15 0 1 4 2 1 1 0 0 2 5 -1 0 5 80 5 80 17 85 79 98 0 6 0
     2 79 98 0 5 11 2 67 11 0 6 0 2 67 17 0 0 0 2 9 -1 0 5 62 1 62 0 6 0 2 32 85 0 0 0 2 8 -1 0
     13 0 1 1 1 1 15 0 1 11 2 1 1 15 0 1 2 2 4 0 0 0 2 8 -1 0 5 96 2 39 96 0 6 0 1 39 0 13 0 1 1
     1 1 15 0 1 96 2 1 1 15 0 1 4 2 4 0
311 2 0 2 68 48 0 0 0 2 5 -1 0 2 0 1 47 0 5 102 2 86 102 0 6 0 1 3 0 0 0 2 7 -1 0 5 -1 2 61 86 0
      12 0 1 102 0 0 0 2 5 -1 0 5 61 2 61 86 0 6 0 3 111 91 83 0 0 0 2 3 -1 0 5 24 1 24 0 6 0 1 93
       0 13 0 1 61 1 0 15 0 1 61 2 61 0 0 0 2 5 -1 0 5 -1 0 0 5 27 4 88 86 38 27 0 0 0 2 5 -1 0 5
      38 3 88 86 38 0 0 0 2 8 -1 0 13 0 1 61 1 0 15 0 1 38 2 61 0 15 0 1 27 2 0 1 0 0 2 9 -1 0 19
      0 1 61 0 0 0 2 8 -1 0 13 0 1 61 1 0 15 0 1 38 2 61 0 13 0 1 0 1 1 15 0 1 61 2 0 1 0 0 2 3 -1
       0 5 82 3 95 99 82 0 6 0 2 99 86 0 5 88 1 88 0 6 0 1 95 0 0 0 2 5 -1 0 5 78 1 78 0 0 0 2 8
      -1 0 13 0 1 82 1 1 15 0 1 78 2 82 1 13 0 1 61 1 0 15 0 1 38 2 61 0 13 0 1 0 1 1 15 0 1 88 2
      0 1 0 0 2 5 -1 0 5 63 2 33 63 0 6 0 4 36 37 33 15 0 13 0 1 0 1 1 15 0 1 63 2 0 1
]]>
  </Save>
</RftgExport>
```

Bibliography

[1] Learning to win: Case-based plan selection in a real-time strategy game, author=Aha, David W and Molineaux, Matthew and Ponsen, Marc. In *International Conference on Case-Based Reasoning*, pages 5–20. Springer, 2005.

[2] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59, 1994.

[3] David W Aha. Case-based learning algorithms. In *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*, volume 1, pages 147–158, 1991.

[4] Board Game Arena. boardgamearena.com. en.boardgamearena.com [online], retrieved 22 October 2016, 2016.

[5] Bryan Auslander, Stephen Lee-Urban, Chad Hogg, and Héctor Munoz-Avila. Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. In *European Conference on Case-Based Reasoning*, pages 59–73. Springer, 2008.

[6] boardgamegeek.com. RFTG General Strategy Guide. boardgamegeek.com/thread/295249/rftg-general-strategy-guide [online], Retrieved 22 October 2016.

[7] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149, 2015.

[8] Michael Buro. Call for AI research in RTS games. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, pages 139–142, 2004.

[9] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1):57–83, 2002.

[10] Claire Cardie and Nicholas Howe. Improving minority class prediction using case-specific feature weights. In *ICML*, pages 57–65, 1997.

[11] David B Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, 2001.

[12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175, 2012.

[13] Rasmus Bille Fynbo and Christian Sinding Nellemann. Developing an agent for Dominion using modern AI-approaches. Master's thesis, IT- University of Copenhagen, 2010.

[14] Rio Grande Games. Race for the Galaxy. riograndegames.com/uploads/Game/Game_240_gameRules.pdf [online], Retrieved 22 October 2016, 2007.

[15] Rio Grande Games. Race for the Galaxy, 2007.

[16] Mabel González, Yanet Rodríguez, and Carlos Morell. k-NN behavior with set-valued attributes. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. 2010. Bruges, Belgium.*

[17] Cathleen Heyden. Implementing a computer player for carcassonne. Master's thesis, Maastricht University, 2009.

[18] Keldon Jones. Race for the Galaxy AI. www.keldon.net/rftg [online], Retrieved 22 October 2016.

[19] John Laird and Michael VanLent. Human-level AI's killer application: Interactive computer games. *AI magazine*, 22(2):15, 2001.

[20] David B Leake and David C Wilson. Remembering Why to Remember: Performance-guided Case-Base Maintenance. In *European Workshop on Advances in Case-Based Reasoning*, pages 161–172. Springer, 2000.

[21] Tom Lehmann. Race for the galaxy. 2007.

[22] Matthew Molineaux and David W Aha. TIELT: A testbed for gaming environments. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, page 1690. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[23] Jay H Powell, Brandon M Hauff, and John D Hastings. Utilizing case-based reasoning and automatic case elicitation to develop a self-taught knowledgeable agent. In *Challenges in Game Artificial Intelligence: Papers from the AAAI Workshop*, 2004.

[24] Oscar Reyes, Carlos Morell, and Sebastián Ventura. Evolutionary feature weighting to improve the performance of multi-label lazy algorithms. *Integrated Computer-Aided Engineering*, 21(4):339–354, 2014.

[25] Michael M Richter and Rosina O Weber. *Case-Based Reasoning.* Springer, 2013.

[26] Christopher K Riesbeck and Roger C Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, 1989.

[27] Armin Rigo and Samuele Pedroni. PyPy's approach to virtual machine construction. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 944–953. ACM, 2006.

[28] Jonathan Rubin. CASPER: design and development of a case-based poker player. Master's thesis, University of Auckland, 2007.

[29] Jonathan Rubin and Ian D Watson. Investigating the Effectiveness of Applying Case-Based Reasoning to the Game of Texas Hold'em. In *FLAIRS Conference*, pages 417–422, 2007.

[30] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[31] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[32] David Sinclair. Using example-based reasoning for selective move generation in two player adversarial games. In *European Workshop on Advances in Case-Based Reasoning*, pages 126–135. Springer, 1998.

[33] David B Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the eleventh international conference on machine learning*, pages 293–301, 1994.

[34] Barry Smyth and Mark T Keane. Remembering to Forget. In *Proceedings of the 14th international joint conference on Artificial intelligence*, pages 377–382. Citeseer, 1995.

[35] Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

[36] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-Carlo Tree Search in Settlers of Vatan. In *Advances in Computer Games*, pages 21–32. Springer, 2009.

[37] Colin D Ward and Peter I Cowling. Monte Carlo search applied to card selection in Magic: The Gathering. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 9–16. IEEE, 2009.

[38] Ian Watson. Case-based reasoning is a methodology not a technology. *Knowledge-based systems*, 12(5):303–308, 1999.

[39] Stefan Wender and Ian Watson. Combining Case-Based Reasoning and Reinforcement Learning for Unit Navigation in Real-Time Strategy Game AI. In *International Conference on Case-Based Reasoning*, pages 511–525. Springer, 2014.

[40] Dietrich Wettschereck, David W Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273–314, 1997.