Game Theoretic and Machine Learning Techniques for Balancing Games

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jeff Long

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Game balance is the problem of determining the fairness of actions or sets of actions in competitive, multiplayer games. This problem primarily arises in the context of designing board and video games. Traditionally, balance has been achieved through large amounts of play-testing and trial-and-error on the part of the designers. In this thesis, it is our intent to lay down the beginnings of a framework for a formal and analytical solution to this problem, combining techniques from game theory and machine learning. We first develop a set of game-theoretic definitions for different forms of balance, and then introduce the concept of a strategic abstraction. We show how machine classification techniques can be used to identify high-level player strategy in games, using the two principal methods of sequence alignment and Naive Bayes classification. Bioinformatics sequence alignment, when combined with a 3-nearest neighbor classification approach, can, with only 3 exemplars of each strategy, correctly identify the strategy used in 55% of cases using all data, and 77% of cases on data that experts indicated actually had a strategic class. Naive Bayes classification achieves similar results, with 65% accuracy on all data and 75% accuracy on data rated to have an actual class. We then show how these game theoretic and machine learning techniques can be combined to automatically build matrices that can be used to analyze game balance properties.

Acknowledgements

Many thanks to my supervisor, Michael C. Horsch, for his invaluable support, advice and for reading this so many times he must surely be bored of it by now.

Thanks also to all participants in the study, and the valuable advice of my thesis committee.

Thanks also to NSERC for funding the years of study that went into this work.

Contents

List of Tables

List of Figures

Chapter 1

Problem Description and Motivation

People who play any sort of competitive game on a regular basis have an intuitive sense of what 'game balance' means. Indeed, many a losing player will, sometimes with justification and sometimes not, blame his impending loss not on his own lack of skill, but on imbalances present in the game. Take, for example, a game of chess. The player who is assigned to play black may feel that he has been forced to play the weaker side in the game, and thus a loss may not be due to his skill, but rather because the odds were stacked against him. Available online statistics indicate that such a complaint could be well-founded, and that black is at a significant disadvantage over white in chess [Chessgames, 2005].

Similarly, players will often come to identify individual game elements with the monikers 'overpowered' or 'underpowered.' This is especially prevalent in online competitive video games. Certain elements of the game — such as a particular military unit or weapon, for example — may seem overly effective, thus marginalizing other elements that they overpower. In the same manner, elements that are too weak will simply go unused by experienced players. Problems of this sort have long plagued online multiplayer games; the *mutalisk* unit, from the popular strategy game Starcraft [Blizzard Entertainment, 1998], has been cited by designers as a prime example of a game element that was overpowered and difficult to balance [Cadwell, 2002].

However, while players are intuitively familiar with the concepts briefly sketched above, intuition is generally where the familiarity ends. While some classical games have been modeled from a complexity theory standpoint (chess, for instance, is EXPTIME-complete [Fraenkel and Lichtenstein, 1981]), to my knowledge there has been no formal definition of these particular problems in an academic setting, espe-

cially coming from the perspective of a design standpoint. Some expertise and advice on 'rules of thumb' exist in industry, of which Cadwell [Cadwell, 2002] is a prime example. Carpenter [Carpenter, 2003] uses risk analysis and spread-sheeting techniques to balance individual elements of multiplayer role-playing games, and Kennerly [Kennerly, 2003] suggests the use of data mining for improving game design. A few other tips and tricks are available [Adams, 1998, 2002, Rouse III, 2005], and usually consists mainly of the author's personal advice and experiences. Furthermore, none of these articles demonstrate rigorous proof or testing of the suggested techniques, nor do they suggest a formal and general-purpose framework for discussing game-balance and its most important properties.

For the purposes of this thesis, the term *game* is used to refer primarily to so-called parlor games - often manifested by board games such as chess, and card games such as Blackjack - and multiplayer computer games. However, many of the concepts and issues raised are applicable to an even wider variety of competitive domains, such as professional sports. We also restrict our attention to games of the two-player variety, though in general the principles discussed are intended to apply to games with numerous players.

To further examine and codify the problem of balance, which at this point has only been discussed as an informal concept, we divide it into two distinct categories: *external balance* and *internal balance*. We will formalize both of these concepts in Chapter 2. External balance refers to the balance between two opposing players in an asymmetric game who are assigned different roles, or action sets, at the game's outset. Many games exhibit this asymmetric property; the example of chess, in which white moves first, has already been mentioned. In the domain of online games, it is extremely common for players to choose a role or faction at the game's outset that dictates their available action set for the remainder of the game. The game is said to be *externally balanced* if neither player has an advantage over the other due simply to the role initially assigned or chosen.

Internal balance refers to the balance between different available actions or elements of an individual player. If a given action is too weak, then the optimal strategy

may be to never use it. Similarly, an action that is too strong may be overused, and could in fact become the game's only viable course of action. Unlike problems of external balance, internal balance problems can exist even in symmetric games where both players have identical action sets available. For instance, online real-time strategy games are symmetric so long as both players select the same initial faction, but the players may have military units available to them that should never be built under optimal play. This is an *internal balance* problem.

Cadwell [Cadwell, 2002] contends that all forms of imbalance boil down to an elimination of choices. We see that the definitions given here agree with that claim. In the case of internal balance, it is evident that if the effectiveness of available actions is too disparate, several player choices have been eliminated. In the case of external balance, if one role in the game is stronger than the other, then that role should always be selected first, if a player is permitted to choose her own role. Again, assuming both players are trying their utmost to win the game, it seems reasonable to say that a player choice exists only if the two sides of the game, although different, are equally likely to win the game.

On the point of external balance, it should be noted that in many games a handicap is sometimes desirable so as to even the playing field between two players of disparate skill. Therefore, it might sometimes be argued that factions of different strengths are desirable so as to provide such a handicap, so that one player can volunteer to play the 'harder' side. However, it is our opinion that a handicap should be an explicit element of a game, configurable by the players, rather than hidden away in the game's opposing roles. A handicap is of little use, after all, if players must first be experts at the game to even realize that the handicap exists.

Both forms of imbalance are highly undesirable in a competitive game. Cadwell [Cadwell, 2002] describes poor game balance as the factor that often stands in between a good design and a good game; Carpenter goes so far as to call it the "holy grail" of game design [Carpenter, 2003]. External imbalance can create discontent and frustration among players before the game has even begun, particularly for the player who is forced into the weaker role. Internal imbalance means that the game

contains extraneous elements, that exist only to frustrate and confuse inexperienced players. From a designer's point of view, it makes little sense to devote resources to creating a game element that expert players will quickly come to ignore.

The most formal of publicly available industry methods of which we are aware for dealing with these problems is the Risk Analysis method by Carpenter [Carpenter, 2003]. This is essentially a probabilistic sampling method which simulates the pitting of game elements against one another and generates a probabilistic model of the outcome using multiple iterations. This method requires an extensive user-defined description of the process being simulated, as well as a knowledge of 'average' player statistics. The method provides a pair-wise comparison of game elements, but for truly complex systems, the number of iterations required to obtain a meaningful output will be significant. Kennerly advocates the use of data mining techniques [Kennerly, 2003], involving simple comparisons of statistics and rates of change. Such techniques can inform a designer that a given aggregated rate (such as the rate at which a certain character class improves in an online role-playing game, for instance) needs to be changed, although how to change the rate or by how much is left to the discretion of the designer.

Beyond the above methods, it appears for the most part that game balance is achieved through extensive amounts of play-testing and trial-and-error on the part of designers. Companies such as Blizzard Entertainment run extensive beta tests that last for months and involve thousands of players, in part to obtain data on game balance issues Blizzard Entertainment [2005]. Balance patches for online strategy games continue to be released for months or even years after a game goes public. Board games and card games generally do not have this option, and so it is even more imperative they be properly balanced before a commercial release. An analytical method for extracting balance problems and principles from game data, or better yet identifying and correcting game balance issues without the need for generating large amounts of data, would be a significant contribution in both domains.

The contributions of this thesis are twofold. First, we reframe and formalize the problems of game balance using the language of economic game theory. We show

how game theoretic concepts and tools can be used to analyze games in extensional form and extract properties relating to game balance. Secondly, we propose building formal game models of manageable size through the use of strategic abstractions, i.e., a high-level strategy implemented by the player. We provide and evaluate several different machine learning techniques for automatically classifying these high-level strategies from real game data, and then show how the results can be used to build game theoretic matrices.

The remainder of this thesis is organized as follows. Chapter 2 gives a brief overview of the game theory used in this work and presents game theoretic definitions of balance, as well as a brief overview of the general application of game theory to games. Chapter 3 presents a general overview of the machine learning approach used in this thesis, as well as describing all of the individual technologies used for experiments. Chapter 4 describes in detail the experimental work of this thesis in automatically classifying strategies from game data, and presents an analysis of the results. Finally, Chapter 5 suggests directions of future work and concludes the thesis.

Chapter 2

Game Theory

In this chapter, we first present the game theoretic notation used in this thesis and then go on to formally define the concepts of external and internal balance and show how they can be identified once a game has been reduced to a normal form matrix.

As discussed in Chapter 1, game balance for competitive, multi-player games is essentially ensuring that the following two broad properties hold. The first property we call *external balance*. For a 2-person game to be externally balanced, we require that both players have an equal chance at winning the game regardless of the game's potential starting conditions. This means we do not want the starting conditions of the game to give an inherent advantage to one player. A symmetric game is trivially externally balanced, because the starting conditions are identical for both players. The second property we call *internal balance*. For a 2-person game to be internally balanced, we require that for each available action or game element available to a player, it is useful in at least one circumstance. In other words, we do not want to confuse players with options that should never sensibly be taken.

For the subsequent work of this thesis, it is necessary to formalize these concepts. To do so, we turn to the language of game theory, as commonly used in the economic sciences. In spite of its name, game theory has rarely been applied to actual games, especially in the context of the game balance problem studied here. We will use game theory to describe the formalisms that will be our objects of study and analysis.

A game, G, consists of an n-dimensional array of payoff elements, where n is the number of players in the game. This matrix representation is said to be the *normal form* of the game. The rows and columns of the matrix represent actions or *strategies*

that are available to the players. Each payoff element is an n-tuple, which specifies the payoffs to each of the n players. A payoff is the reward given to a player for the outcome that a particular array element represents. Usually, these payoffs may be any element of the set of real numbers; sometimes they can even be qualitative in nature. It is assumed that each player's preference is to maximize this payoff.

In this thesis, we consider only 2-player games, in which case the array is a 2-dimensional matrix M, in which the rows represent possible actions of player 1 (termed the row player) and the columns represent the actions of player 2 (termed the column player). The element $M(i, j)$ is a pair of payoff values, indicating the payoff for each player when player 1 selects action i and player 2 selects action j.

We also restrict our attention to *zero-sum games*; that is to say, games that require that the sum of the payoffs in each particular matrix cell is zero. This model is an accurate reflection of competitive games, since in such games one player winning requires that the other has lost. In fact, two-player, zero-sum games are often termed "strictly competitive" in game theoretic language. In this thesis, we represent the payoff value of a win as 1, the value of a loss as -1, and a draw (in games where such is permitted) as 0. Payoff tables throughout this thesis show only the payoff for the row player; the column player's payoff is of course the negation of these payoff elements.

The solution concept used in this thesis is that of mixed strategy Nash equilibrium. A set of strategies is said to be in Nash equilibrium if no player can improve her expected payoff by changing strategies, provided that the other players also do not change strategies. Each row in the payoff matrix is said to represent a pure strategy for the row player (and similarly with columns for the column player). A mixed strategy for a player is a probability distribution over the set of his pure strategies, that defines the probabilistic frequency that each pure-strategy will be selected by the player. While many games will not contain a pure-strategy Nash equilibrium, at least one mixed strategy Nash equilibrium is guaranteed to exist. This mixed strategy will often be referred to as a player's optimal strategy. The standard method of finding the optimal mixed strategy for a matrix game is the Simplex algorithm, a

good description of which can be found in Owen [Owen, 1995] or Papadimitriou and Steiglitz [Papadimitriou and Steiglitz, 1998].

The value of a game, v, is the expected payoff that player 1 will receive upon playing her optimal strategy; player 2 will receive its negation. In the case of two-player, zero-sum games, this value will be unique. Calculating the value of a game is straightforward once the optimal strategies of the game are known; usually it can be obtained simply as a byproduct of calculating these strategies via the Simplex method.

It should be noted that while probably the most common, Nash equilibrium is not the only solution concept in game theory. There are others that exist as well; Azhar, McLennan and Reif [Azhar et al., 1992] and Papadimitriou [Papadimitriou, 2005] discuss some of them, as well as some of the shortcomings of Nash equilibrium. Perhaps the most significant short-coming is that in the general case, multiple Nash equilibria may exist, and there is no known efficient means to find all equilibria points. However, two convenient properties of Nash equilibria are obtained by restricting the proposed model to two-person, zero-sum games. The first is that while it is possible for multiple equilibrium mixed strategies to exist, they are all guaranteed to result in the same expected payoff. Secondly, all equilibrium strategies are interchangeable; that is to say, if (σ_1, σ_2) is an equilibrium pair, that is to say σ_1 is an optimal mixed strategy for player 1 and σ_2 is an optimal mixed strategy for player 2, and (τ_1, τ_2) is also an equilibrium pair, then so are (σ_1, τ_2) and (τ_1, σ_2) [Owen, 1995].

2.1 External Balance

To define *external balance* in game theoretic terms, we relate it to the concept of the value of a two-person, zero-sum game.

Definition 2.1 *A two-player competitive game is externally balanced if neither player, given perfect skill, has a higher probability of winning the game than the other.*

We will now show how this definition relates to the game-theoretic value of the game through the following theorem.

Theorem 2.1 *A two-player competitive game G is externally balanced if and only if the corresponding zero-sum matrix game has a value of 0.*

Proof 2.1 *The following equivalences can be used to obtain both directions of the proof.*

Let G be an $n \times m$ zero-sum matrix game where n and m are the number of pure strategies available to players 1 and 2 respectively, and where element g_{ij} is 1 if player 1 wins the game when her action is i and player 2's action is j, and -1 if player 1 loses. A payoff of 0 indicates a draw. Let x and y be the optimal mixed strategies for players 1 and 2 respectively, where x_i and y_i represent the players' frequency of play for pure strategy i. G has a value of 0 if and only if the following sum holds:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_i y_j g_{ij} = 0 \tag{2.1}$$

As previously defined, $g_{ij} \in \{-1, 0, 1\}$, with 1 representing a win for player 1, -1 representing a loss, or 0 representing a draw. We can partition this sum according to the values of g_{ij}.

$$\sum_{i,j:g_{ij}=1} x_i y_j g_{ij} + \sum_{i,j:g_{ij}=-1} x_i y_j g_{ij} + \sum_{i,j:g_{ij}=0} x_i y_j g_{ij} = 0 \tag{2.2}$$

Clearly, the terms where $g_{ij} = 0$ can all be dropped, as they do not affect the equality. Furthermore, since the g_{ij} are constant in both the other sums (1 for the first sum, and -1 for the second), they can be factored from the summation, yielding the following:

$$\sum_{i,j:g_{ij}=1} x_i y_j - \sum_{i,j:g_{ij}=-1} x_i y_j = 0 \tag{2.3}$$

$$\sum_{i,j:g_{ij}=1} x_i y_j = \sum_{i,j:g_{ij}=-1} x_i y_j \tag{2.4}$$

Now, let p_1 be the probability that player 1 wins the game, and p_2 be the probability that player 2 wins the game, given perfect play by both sides. Since by the definition of

G, player 1 wins the game when $g_{ij} = 1$, and the $x_i y_i$ define a probability distribution over these possible outcomes, we see that:

$$p_1 = \sum_{i,j:g_{ij}=1} x_i y_j \tag{2.5}$$

$$p_2 = \sum_{i,j:g_{ij}=-1} x_i y_j \tag{2.6}$$

By equation 2.4, this is true if and only if:

$$p_1 = p_2 \tag{2.7}$$

2.2 Internal Balance

Next we consider the issue of *internal balance* in game theoretic terms.

Definition 2.2 *A two-player competitive game is said to be internally balanced if for every available action i, there exists some optimal strategy x for which x_i has a non-zero frequency of play.*

One obvious method of determining this is simply to find a player's optimal mixed strategy, x. If $x_i > 0$ for all i, then the definition of internal balance is immediately satisfied. Since optimal strategies are interchangeable in a two-player, zero-sum game, we need not concern ourselves with finding all possible optimal strategies; finding a single one for which this property holds is sufficient. However, while finding a single strategy could prove that the property does hold, we would have to find all optimal strategies to prove that it does not, and in general there is no known method for doing this [Owen, 1995].

Therefore, it would be convenient if an even simpler criteria could be found for determining internal imbalance. One such game-theoretic concept that seems intuitively plausible is that of domination. We say that the ith row dominates the kth row if, in a matrix game G, $g_{ij} \geq g_{kj}$ for all j and $g_{ij} > g_{kj}$ for at least one j. The situation is similar for columns. A dominated row or column can be eliminated from the game without affecting optimal strategies for either player.

Table 2.1: A 4x4 matrix game that illustrates internal imbalance, where the first row and column represent play frequencies for each player's optimal mixed strategy. Note that no row or column is dominated by any other.

//	0%	33%	33%	33%
0%	-1	1	0	-1
0%	0	-1	-1	1
100%	1	0	0	0
0%	-1	0	-1	1

However, while it is evident that a game is not internally balanced if it contains a dominated row or column, the converse does not hold. The proof is through the following simple counter-example in Table 2.1.

It can be found through standard means that the optimal strategy profile x for player 1 consists of the vector (0, 0, 1.0, 0), whereas the profile y for player 2 consists of (0, 0.33, 0.33, 0.33). Both players have one or more options available that should never be played under their optimal strategies. However, an examination of the matrix reveals that no row or column is dominated by any other. This means that a lack of domination is not sufficient to demonstrate internal balance in a game.

Therefore, for this thesis, we must resort to calculating a strategy profile for each player, and ensuring that no action has a zero-frequency of play. Should we find an optimal strategy such that this property holds, we call the game internally balanced. Should we find a dominated action, we say the game is internally imbalanced. Should we fail to find either a strategy where no action has a zero-frequency of play or a dominated action, we cannot say whether or not the game is internally balanced. Therefore, the problem of determining internal balance is a semi-decidable problem.

2.3 Game Theory in Games

In spite of its name, as a modelling tool game theory has more often been used to describe economic, social and political situations than actual games. However, the most basic and famous game-playing algorithm, minimax search, is based upon

game theoretic principles, and with the computational enhancement of alpha-beta pruning [Knuth and Moore, 1975], has long been used for playing well-known, *perfect information* games such as chess [Shannon, 1950]. A perfect information game is one in which all the events that have thus far occurred in the game are visible to all players. Berlekamp, Conway and Guy [Berlekamp et al., 1982] apply a game theoretic, mathematical analysis to a wide variety of puzzles and classical games, but as with chess, the focus of the work remains solving for optimal play, not game design. Ernst [Ernst, 1995] provides a deep, combinatorial game theoretic analysis of the ancient Hawaiian game of konane, but contends that such an approach is unlikely to be practical for games that do not fit the alternating-move, perfect information model. To our knowledge, there is no specific work in applying game theory to provide a general framework for describing design properties of modern games.

Chapter 3

Machine Learning for Constructing Game Matrices from Data

Chapter 2 makes a rather cavalier assumption: namely, that the real-life game in question has already been converted into its normal form matrix, and that payoff values for this matrix are readily available. In reality, it is difficult to imagine a real world game as a simple matrix of payoffs, even for designers well-versed in game theory. It is also difficult to imagine the effects that changing some payoff in the formal matrix representation, which a designer might like to do to correct detected imbalances, might have on the actual game. The concern of this chapter is to examine methods for automatically building a normal form game directly from game data.

A fundamental problem of the normal form is size. No interesting game is simple enough that every possible exact line of play can be enumerated in the game matrix, because if it were small enough, we would be able to find the game's optimal strategy and the game would be solved and thus no longer interesting, in the same way that tic-tac-toe is a solved game. Even if such a matrix could be built, the computational costs for computing the required balance properties would be prohibitive, as the Simplex algorithm is exponential in the worst case [Owen, 1995]. And furthermore, any results that did arise from such a huge matrix would be of little use to game designers, since it is difficult to interpret the meaning of individual payoff values when there are many millions of them in the matrix.

Therefore, in order to make use of the definitions in Chapter 2, designers need two things. First, they will require tools, techniques and guidelines for mapping a real-world game into a normal matrix form, a process which is possibly quite unintuitive

for designers unfamiliar with formal game theory. Secondly, the size of the resulting matrix must be limited so as to be manageable, both for reasons of complexity and meaningfulness of results, while still capturing the essential balance dynamics of the game.

One approach to the problem of balance that might be imagined is to construct some form of database listing all game elements and their numeric properties, over which the designer presumably has control. Designers could make changes to various quantities in the table and ask the system for some kind of prediction on the outcome, similar to the value of the game that we present in Chapter 2. However, we are aware of no known computational method for predicting game results from such a table. Such a table might still be unmanageably large and predicting the outcome of a single change seems a very difficult task. Furthermore, a table that compares game elements only on an individual basis fails to capture potential synergy between elements, as well as intangible variables such as ease of use, or how easy it is for a human player to make use of a particular element. Modelling these factors purely from table seems impossible without a fully-featured automated player which is competitive with high-level human play, which of course does not exist for almost all modern competitive games.

In this thesis, we instead propose using machine learning techniques to build what we call 'strategic abstractions.' We use the term strategic abstraction to refer to the higher-level intent behind individual lines of play, which results in general strategies that are very often recognizable by human experts. For instance, among professional chess players, there are a reasonably small number of 'standard' openings. If we treat these openings as 'strategies' in the game theoretic sense, they could serve as the rows and columns of the matrix necessary for the analysis of Chapter 2, although we would of course prefer an abstraction that captures the entire game, not just the opening. Similarly, players of modern online games often develop and use high-level strategies that can be used as abstractions of the game's state space.

These high-level strategies seem well-suited to our purposes of building a game matrix. First, they seem to satisfy in general the game theoretic assumption that the

row and column of the normal form game are selected secretly and simultaneously, and therefore map well into the formalism proposed in Chapter 2. Secondly, in some cases the number of such strategies is quite manageable, leading to a small matrix easily amenable to analysis. Furthermore, we conjecture that balance results expressed in terms of human-identifiable strategies would offer much more insight to designers than balance results about long and complex individual sequences of moves.

There are some potential drawbacks to the use of these strategic abstractions. The first is that an individual game example may exhibit no general high-level strategy at all. Sometimes the implementation of the strategy may be less clear or have variations, in which case human experts might disagree on the general strategy used. Secondly, the strategy is not something explicitly stored in a game's description, and thus automated methods must be developed for inferring this strategy from other game data. We examine these issues further as part of our experiment in Chapter 4.

The word "abstraction" is very often used in Computer Science and Artificial Intelligence. A complete review is impossible, so we mention a few topical and important uses and later show how they differ from our intended use. Knoblock [Knoblock, 1994] discusses abstractions in the planning domain, and develops a context-independent abstraction generator for planning problems. His abstractions are obtained by dropping literals from the conditions, essentially allowing the planner to build its plan around the more 'important' or difficult attributes of the problem domain. Thus, each state in the abstract search space corresponds to one or more states in the original space, but with certain details omitted. In generating his abstractions, Knoblock enforces the ordered monotonicity property, guaranteeing that operations identified at a higher level of abstraction will be left unchanged as the plan is further refined in detail.

Holte et al. [Holte et al., 1996] use a similar approach in the context of the well-known A* search algorithm. They create an abstraction of the original search space S, thereby allowing distances in the abstract space to serve as the heuristic in the original space. The abstraction is simply created by amalgamating states within a

certain edge radius of each other into a single abstract state. This is done repeatedly and hierarchically so that the search space may be examined at different levels of granularity.

These abstractions by Holte and Knoblock are similar to the strategic abstractions we discuss here in that they are *homomorphic transformations*: one state in the abstract space corresponds to several states in the more detailed, original space. The crucial difference is in the criterion for the grouping of states. Both Knoblock and Holte have as a goal to speed up the process of search. However, planning and search are non-adversarial environments. For the strategic abstractions that we define here, the link between states in the original space is what seems to be their general strategic equivalence against an adversary. The abstractions exist because they facilitate discussion and understanding of the game's strategic properties, as often seen on gaming discussion websites WCStrategy [2006]. Ultimately, though, this classification is one of opinion, and different experts might very well disagree in some cases as to whether a given game followed one strategy or another. Thus, an appropriate grouping into abstract states is not as straight-forward as simply grouping together states that are within n edges of each other in the original space. Furthermore, the search space of the games we examine is far too large to be fully surveyed, making an automatic amalgamation of search states impossible. We therefore instead rely on domain-specific knowledge to create meaningful strategic abstractions.

Therefore, in this thesis we propose the use of machine learning techniques to detect high-level player strategies from raw game data. The ultimate goal is to provide a four stage process of balancing games. First, strategic abstractions of the game are developed and machine learning models are trained to recognize strategies in data. Secondly, these strategies are used to construct a matrix in the normal form as described in Chapter 2. Thirdly, balance properties of this matrix are automatically detected and abstracted, again using the definitions of Chapter 2. Finally, problems are reported back to designers in terms of the over or under effectiveness of high-level strategies.

Although machine learning has the obvious disadvantage of requiring at least

some amount of data, usually at least some of which must be labelled, it otherwise seems well-suited for this task. First of all, a machine learning approach allows for the game matrices of our method to be built almost automatically, without the need for a formal model of the game constructed by the designer. Secondly, it provides an easy method for obtaining matrix payoff elements, which can easily be obtained from data once it has been partitioned into strategies. Thirdly, it captures what players of the game are actually doing, rather than what game designers think they should be doing.

3.1 Machine Learning in Games

There has been a vast array of work in applying machine-learning techniques to the playing of games. Among the most famous is Tesauro's work in training a system to play backgammon [Tesauro and Sejnowski, 1989] [Tesauro, 1995]. Learning of various forms has also been applied to checkers [Samuel, 1959], chess [Thrun, 1995] and othello [Buro, 1999]. Much of this work has been phenomenally successful and has resulted in computer players that rival or defeat human world champions. In these games, learning is often used to develop intermediate state evaluations for use in more traditional minimax search trees. All of these games, however, have benefitted from years of study and in many cases centuries of playtesting; there is no real question about the 'design properties' of these games. The concern is only to find a winning strategy for the game, not to extract properties of the game itself for design purposes.

An exception to this is the work by Kalles and Kanellopoulos [Kalles and Kanellopoulos, 2001], who specifically mention the problem of game design in applying a reinforcement learning algorithm to a simple abstract strategy game. The machine is given only the rules of the game, and is then subjected to a reinforcement learning algorithm in which it repeatedly plays against itself. The goals of this process are to quickly develop a competent computer player for the game, and to expose possible design flaws in the game itself. They posit that their approach can substan-

tially speed up the game design process by allowing designers to rapidly test rule variations. Kalles and Kanellopoulos informally mention many of the concepts we formalize here, particularly the 'fairness' (external balance) of the game, but also the possible dominance of a 'defender' strategy (internal imbalance), although they are not concerned with creating a general framework for identifying and classifying game properties that emerge from their approach. Another caveat is that the usefulness of their approach for design purposes is contingent on the learned behavior being competitive with high-level human play. For newly invented games such as those considered by Kalles and Kanellopoulos, this is difficult to verify without large amounts of human playtesting which was beyond the scope of their study.

It should also be noted that the classically studied games in the machine learning field, although obviously deep and complex in practice, are in general trivially simple from a game theoretic viewpoint. Chess, checkers and othello are all zero-chance, perfect information, alternating move games that fit perfectly into the minimax game tree framework. Many modern games, particularly video games, are not like this and are complicated by real-time play, imperfect information and a branching factor that dwarfs that of checkers. There has been some work by Kovarsky and Buro in applying classical search to simultaneous move abstract combat games [Kovarsky and Buro, 2005], but for the most part, competent game-playing algorithms for more 'difficult' games do not yet exist; instead, computer players are usually given hard-coded scripts to follow which are easily defeated by an experienced human player.

3.2 Experimental Methods and Terminology

So far, we have provided a game theoretic framework for analyzing games, and motivated the need to build strategic abstractions using machine learning tools in order to construct useful matrices. In this section, we will first give an overview of the testbed that will be used for the experimental work in Chapter 4, and then provide a brief background and description of the different computational methods used in the experimental work.

All of the methods we consider are *supervised learning* methods. A supervised learning method is one in which a classifier is built using training examples for which the variable of interest (usually called the class) is known. An unknown example is then assigned a label based on what the classifier has learned from the known labelled examples. We will discuss four such approaches in this section. The first is *Needleman-Wunsch sequence alignment*, a method commonly used in bioinformatics. The second is *k-nearest neighbor* classification. The third is *Naive Bayes* classification and the fourth is *boosting*, an approach that can be used to enhance the accuracy of existing learning algorithms.

3.2.1 Experimental Testbed

The major test-bed for this thesis will be the popular real-time strategy game Warcraft III, by Blizzard Entertainment [Blizzard Entertainment, 2005]. Warcraft III is a competitive, online military strategy game. During the game, players collect resources which they first use to build infrastructure such as production buildings, and then to build military units, which can then be used to attack the opponent. Player actions range from the selection of what buildings to construct and where, choosing the type and number of troops to produce and ordering existing troops to move, attack or use special abilities. In Warcraft III, each faction has nine or ten different types of military units available. In addition, players produce workers to perform basic resource-gathering tasks at their base, and Heroes, which are special units that become more powerful as they fight and gain experience. Military units cost resources to produce, such as gold and lumber. In addition, each unit has a food cost, which is a representation of the 'size' of the unit. A player may only have up to a given limit worth of food cost in play at any one time. Warcraft III is also an imperfect information game, as each player only observes the map within a certain range of her buildings and units.

An important facet of Warcraft III is that at the beginning of a match, players elect to play one of four distinct races, or factions. Each faction has unique units and abilities available, resulting in different strategies depending on the faction match-

up. Factions are chosen simultaneously by both players at the beginning of a match. Opinion of the 'best' faction — or in other words, of balance — is a frequent debate, and Blizzard routinely releases patches to fine-tune game parameters so as to address balance issues.

Warcraft III makes for a suitable test-bed for this project due to the plethora of game data in the form of online replays, available on numerous replay websites [WCReplays, 2006] [WCStrategy, 2006] [BattleReports, 2006]. It is also a game in which general high-level strategies seem to exist and quickly proliferate, due again to the easy availability of replays by expert players. The goal of the experimental section of this thesis is to compare the effectiveness of a variety of machine learning techniques at extracting these strategies from game data and applying game-balance definitions to the resulting matrices.

Our representation of each Warcraft III game consists of a *string of characters*, in which each character corresponds to a military unit constructed by the player. These strings are extracted directly from raw Warcraft III replays. This data format is particularly amenable to analysis by sequence alignment and k-nearest neighbors methods as discussed below, although we use this same representation for all the machine learning algorithms evaluated in Chapter 4. Details of this representation are also discussed in Chapter 4.

3.2.2 Sequence Alignment

In this section, we review the basic terminology of bioinformatics sequence alignment, the first of two methods used in the experimental section of this thesis.

Bioinformatics offers a plethora of techniques for comparing the similarity of long strings of characters (traditionally, representing nucleic and amino acids). Recent methods can compute this similarity for entire genomes consisting of thousands if not millions of characters [Delcher et al., 1999] [Brudno and Morgenstern, 2002]. The most fundamental of these methods is the Needleman-Wunsch algorithm [Needleman and Wunsch, 1970], along with the closely related Smith-Waterman algorithm [Smith and Waterman, 1981] and their many heuristic variants, such as BLAST and PSI-

BLAST [Altschul et al., 1997]. The basic idea of Needleman-Wunsch is to take two strings of characters and align them, usually through the insertion of gaps, especially in strings of disparate lengths, so as to maximize the score of the alignment. In its most simple form, we can use a scoring function of 0 for a match and -1 for a gap or a mismatch. Thus, an optimal alignment of the strings ATAT and AGGAT would consist of:

```
AT_AT
AGGAT
```

and the score would be -2. Using this scoring function, Needleman-Wunsch alignment in fact results in the Levenshtein [Levenshtein, 1966] distance, or edit distance, between two strings. However, sometimes a more complex scoring function is used [States et al., 1991].

Needleman-Wunsch alignment is a greedy, dynamic programming algorithm. Consider the alignment of two strings, X and Y, of lengths n and m respectively. We will find the global alignment of these strings by first storing the alignment of substrings of X and Y in an $n * m$ alignment matrix. Take the problem of aligning x_i, the ith character of X, with y_j, the jth character of Y. x_0 to x_{i-1} are assumed to have already been optimally aligned with y_0 to y_{j-1}, and stored in the alignment matrix. Therefore, at this stage, there are only three options. The first is to insert a gap in X at the current position, pay the cost of aligning a gap with y_j and increment j. The second is to insert a gap in Y at the current position, pay the cost of aligning a gap with x_i and increment i. The third is to pay the cost of aligning x_i and y_j and increment both i and j. Needleman-Wunsch alignment chooses the option with the lowest cost and store the result; if there is a tie, then either choice would result in an optimal alignment. When the last character for both strings is reached, the cost of the optimal alignment will be stored in the bottom right-hand corner of the matrix we have been using to keep track of the alignment scores.

Needleman-Wunsch sequence alignment is guaranteed to result in the optimal alignment score. However, the algorithm is quadratic in the length of the two strings, and since biological sequences are frequently in the millions of characters, most often

heuristic versions such as BLAST and PSI-BLAST are used [Altschul et al., 1997]. In addition, many modern alignment programs can deal with more sophisticated features to better model real-world biological processes, such as transposition and variable gap costs. For example, translation is the process through which two segments of DNA will be swapped in their ordering. Because of this process, we say that the strings AGGGCTTT and CTTTAGGG are much more similar than their simple edit distance would imply. Furthermore, in biological sequences, new characters tend to occur in chunks, and thus for most biological purposes, the presence of a gap is more significant than its length. Modern alignment programs can recognize these features and allow for them while calculating the alignment cost.

3.2.3 Nearest Neighbors Methods

The nearest neighbor approach is another machine learning method, dating back to Fix and Hodges [Fix and Hodges, 1957]. The fundamental idea of a nearest neighbor approach is, given a labelled dataset and an unlabelled example x, assign to x the label of the closest example in the dataset according to some distance metric. A common extension is the *k-nearest neighbor* approach (often abbreviated kNN), in which we select the k nearest neighbors to an unlabelled example x, where k is an input to the algorithm, and simply have a vote among these k examples to assign a class to x. This method is perhaps among the simplest of supervised learning methods.

The most difficult part of kNN classification is choosing an appropriate distance metric. Euclidean distance between numeric feature vectors is often inappropriate, as all features may not be of equal importance or measured on the same scale. Stanfill and Waltz [Stanfill and Waltz, 1986] examine these issues of adapting a distance metric to the data. Since we are representing our data as strings of characters, however, we will use the Needleman-Wunsch alignment distance as our distance metric for the experiments in this thesis.

3.2.4 Naive Bayes Classification

Naive Bayes classification is a common, simple and (perhaps surprisingly) effective classification and machine learning technique. It is a Bayesian network with a simple, restricted structure. A Bayesian network is a Directed Acyclic Graph (DAG), consisting of a set of nodes, which represent random variables, and a set of arcs which represent conditional dependence between variables. Associated with each node is a Conditional Probability Table (CPT), which lists the probabilities that the variable will take on a given value, given the possible states of its parents in the graph. Taken together, the CPTs are a factorization of the full joint probability distribution (JPD) of the network. Probabilities of interest are calculated by multiplying the CPTs of the network together and summing over unobserved values [Pearl, 1988].

The major strength of Bayesian networks is their ability to compactly represent conditional independence. Unless the graph is very dense, this can result in an exponential savings in space requirements over the pure JPD, because independence between variables is explicitly represented.

It has been shown that in the worst case, inference in a Bayesian network is also NP-hard [Cooper, 1990]. However, quite often this inference can in fact be very efficient, depending on the network structure. The Naive Bayes classifier is one such structure.

In Naive Bayes, there is one node called the class node, which is normally unobserved and represents the class we are trying to predict. All other nodes are attribute nodes, and these nodes are all children of the class node. There are no other arcs permitted in the graph.

Once the structure of a Naive Bayes classifier has been set, the CPTs for each node can easily be learned from data. This is essentially just a counting process, in which we count the number of times a particular attribute value occurred, given the class, and is given by Equation 3.1, where C is the class node, c is a possible value of the class node, A is an attribute node and a is a possible value of the attribute node. The term $p(X = x)$ means the probability a that variable X has a value of x,

while the term $\#(X = x)$ represents the number of times the variable X had a value of x in the sample data. Like the kNN classifier discussed above, Naive Bayes is a supervised learning method and requires that the class of all training data is known.

$$p(A = a|C) = \frac{\#(A = a, C = c)}{\#(A = a)} \tag{3.1}$$

The posterior probability of the class node can then be calculated according to Equation 3.2, given that our n evidence nodes, $A_1, \ldots, A_n$, have the values $a_1, \ldots, a_n$.

$$p(C = c|a_1, \ldots, a_n) = \frac{p(a_1, \ldots, a_n|C = c)p(C = c)}{p(a_1, \ldots, a_n)} \tag{3.2}$$

However, the key assumption of the Naive Bayes classifier is that the attributes are independent given the class. Therefore, Equation 3.2 can be simplified to be read directly from the network. As the denominator is simply a normalizing constant, it too can be dropped from the expression and we may obtain the most likely class by choosing c such that Equation 3.3 is maximized.

$$p(C = c|a_1, \ldots, a_n) = p(a_1|C = c), \ldots, p(a_n|C = c)p(C = c) \tag{3.3}$$

For more details on Bayes nets in general and learning in Bayes nets, see Heckerman [Heckerman, 1996b] or Krause [Krause, 1998].

Naive Bayes has been used successfully in a variety of domains, and due to its simplicity, it is a good first approach to a variety of machine learning problems. One of the better known problems to which it has been applied is handwriting recognition. Frey [Frey, 1998] tests a Naive Bayes classifier on a collection of handwritten digit data (numeric digits from 0 to 9) with a variety of sample sizes. With a training set size of 120, or 12 examples per digit, accuracy of the classifier is around 70%. With 1920 training samples, or 192 examples per digit, the accuracy is higher, approaching 80%. Naive Bayes has been even more successful in the domain of junk email classification, achieving a precision of 95% with a training set of 2500 messages [Sahami et al., 1998]. As a result, many modern email clients now possess such a filter.

3.2.5 Boosting and the ADABoost Algorithm

Boosting is a fairly contemporary technique that is not itself a machine learning method, but rather a method for potentially increasing the accuracy of an existing machine learning algorithm. Boosting was first proposed theoretically by Schapire [Schapire, 1990]. The basic idea is instead of training only one classifier, we train M classifiers, each of which will have a weighted vote on classifying unknown examples. Each of the M classifiers is trained on the same training data, and then tested on this same data; the training data is then weighted for training the next classifier. Samples that were correctly classified by the classifier m will be assigned a lower weight when training classifier $m + 1$, the basic idea being that we already have a good classifier for those samples and now we need to work on building a classifier that handles the 'tougher' ones.

An important property of boosting in general is that for large enough M, it is guaranteed to produce a weighted set of classifiers that perfectly classifies the training set, so long as the machine learning algorithm used is a *weak learning* algorithm — that is to say, better than random guessing [Russell and Norvig, 2003]. For boolean classification problems, this means a 50% classification accuracy on the training set is needed. Schapire shows that for multiclass classification problems, the same is generally true, but boosting algorithms may require modification; however, so long as classifier accuracy is greater than 50% for these multiclass problems, the same algorithms as those used for the boolean case can still be used [Schapire, 2001].

For this thesis, we will make use in particular of the ADABoost algorithm, developed by Freund and Schapire [Freund and Schapire, 1996]. We use the version as presented by Russell and Norvig [Russell and Norvig, 2003] and which appears as follows:

function ADABoost(examples, L, M) returns a a weighted-majority hypothesis

inputs: examples, set of N labelled examples
L, a learning algorithm
M, the number of hypotheses in the ensemble

local variables: w, a vector of N examples weights, initially $1/N$
h, a vector of M hypotheses
z, a vector of M hypothesis weights

for $m = 1$ to M do
h[m] = L(examples, w)
$error = 0$
for $j = 1$ to N do
if h[m](x_j) != y_j then error = error + w[j]
for $j = 1$ to N do
if h[m](x_j) = y_j then w[j] = w[j] * $error/(1 - error)$
w = NORMALIZE(w)
z[m] = $log(1 - error)/error$
return WEIGHTED-MAJORITY(h, z)

The function WEIGHTED-MAJORITY at the end returns the output value with the highest vote from all the hypotheses in h, with the votes rated by the vector z.

The most important steps of the algorithm are calculating the error of the current model, and adjusting the weight of each sample for the subsequent model. For the former, whenever an example is incorrectly classified by the current model, we add the weight of the example to the current model's error. For the latter, if the example was correctly classified, we reduce its weight by a factor of $error/(1-error)$, meaning that if we have a highly accurate model that correctly classifies the example, we significantly reduce the weight of that example for future models.

Boosting is primarily intended as a means of combining several crude, 'rule of thumb' hypotheses to form a strong and accurate classifier [Schapire, 2001]. Russell and Norvig show results for a sample boolean decision problem using restaurant data where boosting with a decision tree model can achieve 80% classification accuracy with a training set of 20 samples, and 90% with a training set of 100 samples. The unboosted decision tree on the same data achieves accuracy of 70% and 75%

respectively [Russell and Norvig, 2003]. Other general problems to which a boosting approach have been applied are spoken language understanding [Tur et al., 2003], text categorization [Schapire and Singer, 2000] and auction price prediction [Schapire et al., 2002].

Chapter 4

An Empirical Study of Abstract Strategy Identification from Data

The overall goal of the experimental part of the study is to answer three general questions. First, do high-level abstract strategies that human players can identify really exist in game data? Second, can this high-level strategy be automatically identified, and with what accuracy? And third, if so, how do available methods compare for this task?

To achieve this goal, the study is divided into four major sections. Section 4.1 describes a pre-data survey of experts for determining domain-specific abstract strategic classes. Section 4.2 describes the collection of game data and labelling by human experts. Section 4.3 describes the methodology for classifying the data using sequence alignment, kNN classification, Naive Bayes and Naive Bayes with boosting. Section 4.4 shows how game balance matrices can be constructed using the results from Section 4.3, and Section 4.5 provides an analysis of the results.

4.1 Pre-Data Survey

The goal of the pre-data survey phase of the study was two-fold: first, to collect participants for the study, and secondly to determine the contents of the classification questionnaire that would be used during data collection. The purpose of the study was not to evaluate the opinions of the participants, but rather to apply their expert knowledge to the data. Therefore, participants were asked to be familiar with the game *Warcraft III: The Frozen Throne*, by virtue of a nominal minimum rank on

the Warcraft III online ladder (see Section A of the Appendix for a brief description of the Warcraft ladder). Beyond this, no evaluation of the participants' expertise or qualifications was performed.

In addition, each participant was given an initial survey. The survey consisted of a list of several potential high-level strategies usable by experienced Warcraft III players, as well as a description of those strategies. The initial list of strategies was hand-picked by the study's author, using prior experience with the game, resulting in a choice of three or four strategies for each of the four races. Warcraft III has four distinct races or factions from which a player may choose, each with unique units and strategic options available. Participants were asked to agree or disagree on each listed strategy as a common and viable strategy on the Warcraft III: The Frozen Throne online competitive ladder. Participants were also asked to include a free-form description of any major strategies they felt had been left out by the study's author. A copy of the distributed survey is included in section B of the Appendix.

A total of 15 people participated in this portion of the study. In general, participants agreed with all of the potential strategies proposed by the study's author, as well as proposing a total of 9 additional strategies. Due to this agreement, all the proposed and suggested strategies were included on the questionnaire used in the Data Collection section below, except for the initial Double Lodge strategy, due to low levels of player agreement. A full summary of the results of this survey is provided in Table 4.1.

4.2 Data Collection

The purpose of data collection was to assign a strategy label to a set of Warcraft III: The Frozen Throne replays. One hundred replays were collected, all from the public website `www.wcreplays.com` [WCReplays, 2006]. At the time of this study, the current version of Warcraft III was 1.20d (it should be noted that replays are not compatible between major patches, and so data used in this study may not be viewable using the Warcraft III game engine in the future). All replays depicted

Table 4.1: Results of the pre-data survey issued to participance, showing the number of participants who agreed that the strategies proposed by the study's author were common and viable Warcraft III strategies.

Orc Strategies				
	Gruntraider	Gruntapult	WyvernRush	DoubleLodge
Count	13/15	11/15	8/15	6/15

Human Strategies			
	Fast Expand	RifleSorc	GryphonRush
Count	12/15	11/15	10/15

Undead Strategies			
	FiendStatue	GhoulGarg	Necrorush
Count	12/15	13/15	10/15

Night Elf Strategies			
	HuntressRush	ArcherTalon	FastBears
Count	14/15	12/15	9/15

play of top-ranked Warcraft III players, and many of the games were from important tournament matches. As all collected games were two-player matches, this resulted in a data-set of 200 individual samples or 'builds.' These were selected randomly from the website, resulting in a distribution of Warcraft III factions as follows: 42 Orc builds, 43 Human builds, 67 Night Elf builds and 48 Undead builds. This sample was selected at random and the prevalence of Night Elf builds would therefore most likely be caused by a higher popularity of the Night Elf faction.

Replays were then distributed in randomly determined sets to participants who had responded to the pre-data survey. Each participant was sent from ten to twenty replays, and asked to classify the strategy of each player in each game according to a multiple choice questionnaire distributed with the replays. The available options for the questionnaire were taken directly from the pre-data survey. In addition, a 'None of the Above' option was included as a valid option, to represent the possibility that a game may not exhibit a common and identifiable strategy. Unlike in the pre-data survey, participants were not asked to qualify the selection of this option. Classification options were not assumed to be mutually exclusive, and participants were invited to select more than one strategy if they felt the game was best described in this manner. A copy of the questionnaire instructions and an example questionnaire are included in Appendix C.

Participation in this phase of the study was somewhat lower than what was hoped for. This resulted in a total of 7 respondants for the one-hundred replays. It was initially hoped to have some redundancy in the study by having the same replay viewed by multiple raters, but with the small number of participants this was judged to be a poor use of available resources.

In general, however, the results of the data collection were positive in that participants indicated that most games fell into one of the strategic categories listed on the questionnaire. Some of the available options on the questionnaire were never chosen by the raters, but as the questionnaire was designed to err on the side of over-including strategies, this was deemed to be acceptable. Unfortunately, as there was no overlap between raters, it is impossible to measure how much human expert

Table 4.2: Results of game data classification, showing the number of times each strategy appeared according to human experts. Note that the total strategies listed will be higher than the total number of games, as participants were permitted to list more than one strategy per game.

Orc Strategies					
	Gruntraider	Gruntapult	WyvernRush	FirelordRush	None
Count	22	4	4	1	8
Human Strategies					
	Fast Expand	RifleSorc	GryphonRush	FootmanCaster	None
Count	7	4	4	15	11
Undead Strategies					
	FiendStatue	GhoulGarg	FiendAbom	GhoulNuke	None
Count	13	11	2	9	7
Night Elf Strategies					
	HuntressRush	ArcherTalon	BearDryad	FastBears	None
Count	4	12	22	16	11

opinion might potentially differ over these labels. All four races ended up with three or four major strategies that occurred with a significant frequency. Ultimately of the one-hundred replays collected, 85 ended up being used for the experimental work of this chapter, due to expert availablility and data corruptions such as an expert being unable to load game files due to a missing Warcraft map file, or otherwise returning an incomplete survey. The exact results of the data collection phase are listed in Table 4.2. Note that some of these strategies are different from those listed in Table 4.1, as some strategies were suggested by experts during the pre-data survey and other strategies did not show up in the experts' assessment of the data.

4.3 Data Classification

Two classification approaches were compared for their performance on this data. For both approaches, data was randomly split between exemplars, or training data, and

test data. Typically in machine learning, this split is 80% training data and 20% test data. However, due to our very small data set, we generally use considerably less training data; the exact percentage of the split varies depending on the approach.

The goal of the classification was to assign a label to each game that matched the label given by a human rater. As previously mentioned, some games have more than one label assigned by the human expert. In general no attempt is made to assign more than one label to a game using our machine classification approaches, and the machine-assigned label is considered correct if it matches any of the labels assigned by the expert.

4.3.1 Structure of Data

For our experiments, we encode each Warcraft III game as an alphabetic string of characters, where each character in the string represents the production of a specific military unit. In this section, we describe in detail how this string is constructed.

The game Warcraft III has a built-in feature for saving games and allowing the entire game to be viewed at a later time. These replays are designed to be viewed using the Warcraft III game engine; as such, certain details of the game are not stored in the replay file but are recreated by the engine during playback, presumably so as to make the replay files themselves as small as possible. However, online documentation exists for the binary format of the replay files, courtesy of Gonera [Gonera, 2003]. For the classification section of this study, data was thus parsed directly from the replay files, using a parser implemented in C++ and based on that made available by Fetter [Fetter, 2005].

The end result of the parsing is a file for each individual player build; thus, two builds from each replay file. The file contains the faction used by the player, the strategies assigned by the human experts (as previously stated, assigning multiple strategies was permitted), and a representation of the game encoded as a string of characters. The features used for this construction consisted of the different military units constructed by the player, in the order they were built. Although the replay file contains other information about the game, such as movement and attack commands

issued to units, the construction of units was chosen as the most informative feature for our purposes. Each unit was mapped to a single English-alphabet character, resulting in a temporally ordered string usually around thirty to forty characters long. For the purposes of classification, builds were only compared against other builds belonging to the same faction. Therefore, using the same alphabetic character to represent multiple units was acceptable so long as all such units were from a different faction. Only production builds of combat units were thus encoded; workers and Heroes were not included in the game description. The alphabetic unit encoding is listed in Section D of the appendix.

4.3.2 Needleman-Wunsch Alignment Methodology

The basic terminology of Needleman-Wunsch sequence alignment has already been described in Section 3.2.2. Here, we present the details of the sequence alignment algorithm used in this study. First we will comment on the algorithm used, and then describe the scoring matrix used for the experiments. We then present the experimental results of this method.

In this study, we use a strict implementation of basic Needleman-Wunsch. We believe that accounting for additional properties such as translation and variable gap cost is not appropriate in our domain. Because the strings are temporally ordered, building a host of one type of unit followed by a host of another is very different from building the units in the opposite order. The biological translation process is not one that occurs in a game with high frequency. Similarly, including one or two extra characters, or units, into a string does not seem especially important, because it does not represent a significant strategic investment by the player. Therefore, a naive, linear gap cost in fact seems like the best choice in this domain. Finally, as our strings are at most a few hundred characters, instead of the millions that are typical in bioinformatics, a quadratic time complexity is perfectly acceptable.

The scoring matrix used in sequence alignment is supposed to represent, in some sense, the likelihood that one character would be substituted for another. We first present results using a naive, edit distance matrix (with a score of 0 for a match and -1

for a mismatch), and then present results using a matrix customized using knowledge of the domain. For the customized scoring matrix, we use a system based on the food cost of the unit from the actual game, as described in Chapter 3.1. Two units of the same food cost typically represent a reasonably equivalent level of resource investment on the part of the player. A successful match results in a high positive score of sixteen multiplied by the unit's food cost. The constant of 16 was chosen to emphasize the importance of aligning identical units, and seemed to result in good performance compared to other values. A mismatch results in a negative score based on the difference in food cost between the mismatched in units. These penalties are to represent the fact that replacing a small unit, in terms of food cost, with a very large one represents an important strategic difference and therefore such a mismatch is more heavily penalized. In addition, it is also a larger strategic difference to replace one unit with a unit from a different production building, because of the overhead cost of creating different production buildings. There is therefore an additional mismatch penalty if the mismatched units come from different production buildings. The exact matrices used in the alignment are included in Section D of the Appendix.

For the experiment, we first separate the builds based on the four Warcraft III races. We then randomly select one build per distinct strategy to be the exemplar for that strategy. These builds are essentially our 'training data' for this experiment. The remainder of the data is our test set. For each test build, we compute its alignment score with every exemplar of the same race. We then assign to the build the same strategy as the exemplar to which it was closest, in the manner of a one-nearest-neighbor approach as described in Section 3.2.3. If the exemplar had more than one strategy associated with it, all of its strategies are assigned to matching builds. The entire process starting from the random selection of exemplars is repeated 100 times to obtain average accuracy and standard deviation. A sample is considered correctly classified when the automatically assigned label matches the label assigned by the human expert (or at least one label, in the case of a multiply labelled sample). We present the accuracy as a percentage of the games where the correct label was assigned.

A particularly tricky label to deal with is the 'None' label. It does not make sense to assign an exemplar for the 'None' strategy, because by definition games with a 'None' label should have no real strategic similarity, and nothing meaningful in common with each other; if they do, it would be because a common strategy missed being listed in the survey phase of the study. Therefore, we set a threshold such that if the threshold is higher than the best matching alignment score, we assign the build the 'None' label. The threshold is dependent on the length of the strings because we would expect to find more mismatch errors in longer strings, especially if they differ in length. For these experiments, we use a threshold score as shown in Equation 4.1 for the edit distance scoring matrix, and in Equation 4.2 for the customized scoring matrix. In both cases, S_1 is the length of the first string and S_2 is the length of the second string.

$$threshold = -\frac{1}{2} * (S_1 + S_2) \tag{4.1}$$

$$threshold = 3 * (S_1 + S_2) \tag{4.2}$$

The constants listed simply produced the best performance of all values tried.

Table 4.3 shows the average results for 100 runs of the above experiment. The accuracy shown is the total percentage of samples that were correctly classified; the numbers in parentheses in the same cell show the number of correctly classified samples over the number of total samples. It should be noted that not all totals will be multiples of 100 due to games with multiple labels, which are sometimes randomly selected to be exemplars of different strategy classes. We also show the maximum and minimum accuracy that occurred over all 100 random sets of exemplars. Because misclassifying a 'None' build is something of a a different kind of error, we show results both including and excluding the 'None' builds from the test set. We also show results broken down by each individual strategy to better show which strategies tend to be successfully identified and which are more difficult. This breakdown is shown in Tables 4.4 and Table 4.5. Again, the numbers in parentheses show the number of correctly classified and total samples that fell into each cell.

Table 4.3: Results of 100 test runs using Needleman-Wunsch sequence alignment for classification and selecting exemplars at random.

	Avg Accuracy	Std Dev	Max Acc.	Min Acc.
All Data, Edit Distance	49.0% (7442/15200)	4.30	61.8%	32.9%
None-labels Excluded, Edit Distance	63.3% (7278/11500)	5.61	79.1%	43.5%
All Data, Custom Score	53.6% (8146/15200)	5.37	65.1%	38.8%
None-labels Excluded, Custom Score	69.9% (8038/11500)	7.14	85.2%	48.7%

Table 4.4: Breakdown of Needleman-Wunsch alignment results by game strategy using the edit-distance scoring matrix.

Race	Strategy	Average Accuracy
Orc	Gruntraider	80.8% (1570/1943)
Orc	WyvernRush	74.3% (223/300)
Orc	Gruntapult	54.3% (163/300)
Human	FootmanCaster	64.4% (880/1367)
Human	GryphonRush	43.3% (114/263)
Human	FastExpand	52.3% (301/575)
Human	RifleSorc	46.0% (138/300)
Night Elves	ArcherTalon	72.2% (770/1066)
Night Elves	BearDryad	56.1% (1147/2044)
Night Elves	FastBears	80.1% (1191/1487)
Night Elves	HuntressRush	18.0% (54/300)
Undead	FiendStatue	81.2% (875/1077)
Undead	GhoulNuke	62.5% (500/800)
Undead	GhoulGarg	40.9% (409/1000)
All	None	4.4% (164/3700)

Table 4.5: Breakdown of Needleman-Wunsch alignment results by game strategy using the customized scoring matrix.

Race	Strategy	Average Accuracy
Orc	Gruntraider	74.1% (1439/1941)
Orc	WyvernRush	94.3% (283/300)
Orc	Gruntapult	86.0% (258/300)
Human	FootmanCaster	62.8% (862/1372)
Human	GryphonRush	80.5% (214/266)
Human	FastExpand	66.6% (385/578)
Human	RifleSorc	84.3% (253/300)
Night Elves	ArcherTalon	70.0% (755/1078)
Night Elves	BearDryad	76.9% (1581/2055)
Night Elves	FastBears	71.8% (1069/1489)
Night Elves	HuntressRush	74.0% (222/300)
Undead	FiendStatue	95.2% (1033/1085)
Undead	GhoulNuke	58.3% (466/800)
Undead	GhoulGarg	47.1% (471/1000)
All	None	2.9% (108/3700)

Table 4.6: Results a test run using Needleman-Wunsch sequence alignment for classification and selecting a set of 'good' exemplars according to a simple heuristic.

	Accuracy
All Data	58.6% (89/152)
None-labels Excluded	75.7% (87/115)

In the tests above, the selection of exemplars is done at random. However, in a real design environment, designers would be unlikely to select exemplars randomly; they would select one that is generally quite clear and is representative of the overall strategy it is supposed to capture. A random selection of exemplars may therefore not be representative of how the algorithm would perform in practice. At the same time, it is unlikely a great deal of effort would be put in to select an optimal exemplar. Therefore, we heuristically select a 'good' exemplar using the following heuristic: for each set of strategies, select as the exemplar the strategy that results in the highest average alignment score with the other builds of the same strategy. In other words, we select the example that is closest to all other examples in the same class. The complexity of this operation is quadratic in the size of the largest strategy set. Using the exemplars thus selected gives the results in Table 4.6. The accuracy is slightly higher than the average accuracy of a random selection, and is considerably higher than the minimum accuracy seen in the random test set. However, in real data we would be unable to filter the 'None'-labelled examples, meaning we can expect performance indicated by the 'All Data' results in Table 4.6.

4.3.3 3-nearest Neighbor Classification

In Section 4.3.2, we used only a 1-nearest neighbor approach to classify builds against our database of exemplars. In this section, we present results using a 3-nearest neighbor approach, still using the same Needleman-Wunsch alignment score as our distance metric just as in Section 4.3.2. We show results only for tests using our custom alignment score matrix.

For this experiment, instead of selecting only 1 build per class as the exemplar,

Table 4.7: Results of 100 test runs using Needleman-Wunsch sequence alignment with a 3-nearest neighbor approach for classification.

	Average Accuracy	Std Dev	Max Acc	Min Acc
All Data	54.94% (6868/12500)	3.68	64.0%	45.6%
None-labels Excluded	77.48% (6818/8800)	5.26	90.9%	64.8%

we instead select 3, so that in the best case, all 3 nearest neighbors to an unlabelled sample can potentially be of the same class. Then, for each build in the test set, we compute the 3 nearest neighbors according to the Needleman-Wunsch alignment score. If two or more of these exemplars agree on the same class, then that class is assigned to the test sample. If the three nearest exemplars all have a different class, we assign the class of the single closest neighbor. If the alignment score with this single nearest neighbor is less than $10 * \frac{s1+s2}{2}$, where $s1$ is the length of the first string and $s2$ the length of the second, we assign a label of 'None' to the build.

The results of 100 test runs of this experiment are presented in Table 4.7. As before, we also present a breakdown of the results by strategy; these are shown in Table 4.8. We note that some strategies from previous tables, particularly the GryphonRush strategy, do not appear in this table due to insufficient data when we use 3 exemplars per strategy instead of 1.

4.3.4 Naive Bayes Classification Methodology

For this part of the study, we construct a Naive Bayes classifier with one class node and eighty attribute nodes. The class node represents the strategy being used, while the eighty attribute nodes represent the characters x_1 to x_{80} in the original data string, X. This structure is depicted in Figure 4.1. This representation was chosen so as to be directly comparable to the sequence alignment approach, keeping the structure of the data the same for both techniques. The value of 80 was chosen arbitrarily based on the observation that all the games in the dataset except for one were eighty characters or less. Most games were in fact around the forty mark, and for these games the additional attribute nodes will have no effect on classification.

Table 4.8: Breakdown of Needleman-Wunsch alignment results with 3-nearest neighbor approach by game strategy.

Race	Strategy	Average Accuracy
Orc	Gruntraider	84.0% (1390/1655)
Orc	WyvernRush	100.0% (100/100)
Orc	Gruntapult	83.0% (83/100)
Human	FootmanCaster	77.9% (856/1099)
Human	FastExpand	75.1% (251/334)
Human	RifleSorc	86.0% (86/100)
Night Elves	ArcherTalon	86.2% (706/819)
Night Elves	BearDryad	75.8% (1329/1754)
Night Elves	FastBears	71.3% (905/1270)
Night Elves	HuntressRush	88.0% (88/100)
Undead	FiendStatue	95.2% (796/836)
Undead	GhoulNuke	61.2% (367/600)
Undead	GhoulGarg	73.8% (590/800)
All	None	1.4% (50/3700)

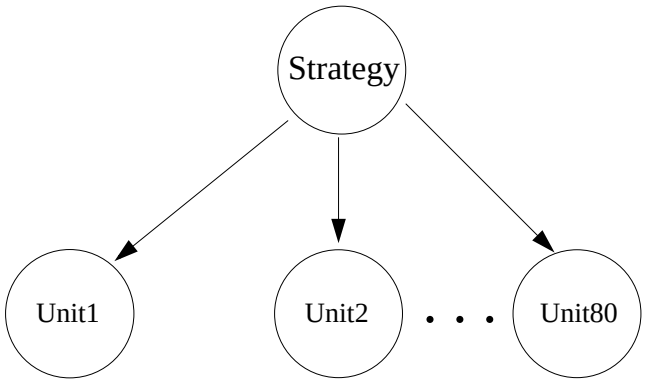

Figure 4.1: The structure of the Naive Bayesian classifier for strategy prediction. Each UnitX node represents the character at position X in the game description.

This network was implemented in Java, using the commercial Bayes net software package Norsys Netica [Netica] to build the Bayes net and perform learning and inference.

For the evaluation of this classifier, data was first divided based on the Warcraft III race being used. It was then sorted by strategy, and randomly split into 50 percent training data and 50 percent test data. An 80-20 split is more common for machine learning, but with our small amount of data, a 50-50 split seemed more appropriate. For game descriptions less than eighty characters in length, the values of excess character positions were simply left as unobserved for the purposes of both training and testing; this has the same effect on classification as if the nodes were not even present in the network.

Again, the 'None' strategy label poses a dilemma. Unlike with the sequence alignment approach, however, the Bayes net provides us with a principled way of dealing with it. By including it as an option in the class node, we can have the Bayes net tell us when it is the most probable label. The accuracy of this depends on the meaningfulness of the 'None'-labelled training examples. The only commonality between 'None'-labelled examples is that they are not like any of the named classes, and we would not expect them to have anything in common with each other either.

Table 4.9 shows the average results for 100 runs of the above experiment. We

Table 4.9: Results of 100 test runs using Naive Bayes classification, randomly selecting half of the data as a training set for each run.

	Average Accuracy	Std Dev	Max Acc.	Min Acc.
All Data	62.9% (5599/8900)	3.26	73.0%	55.1%
None-labels Excluded	73.3% (4949/6900)	3.68	84.1%	63.8%

show results both including and excluding the 'None' builds from the test set. Again, accuracy is defined as the percentage of correctly labelled samples, and the numbers in parentheses show the number of correctly labelled samples over the total number of samples.

In general, Naive Bayes performs well on strategies that are generally characterized by building a large number of two or three distinct units, such as the GruntRaider or ArcherTalon strategies. It performs less well for strategies that are characterized more by the presence of just a few units of high strategic importance. An example of this sort of strategy is the Gruntapult strategy, which is characterized by the player building only a few catapult units — perhaps two or three — but the event is strategically significant. We can modify the Naive Bayes approach to deal with this problem, at least in part. We do so by adding to the classifier ten additional boolean-valued attribute nodes, each one being a child of the strategy class node, with each such node corresponding to the presence or absence of one particular unit type in the build. A depiction of this enhanced structure can be found in Figure 4.2. This modification results in roughly a 3% total increase in accuracy, as shown in Table 4.11. For this improved structure, we show results by strategy in Table 4.12, which shows that the increase in accuracy is largely coming from strategies that were poorly classified before, such as the Gruntapult and WyvernRush strategies. Some strategies however, such as the GryphonRush strategy, continue to be very difficult to identify even with this enhancement.

Table 4.10: Breakdown of Naive Bayes classifier results by strategy.

Race	Strategy	Average Accuracy
Orc	Gruntraider	100.0% (1201/1201)
Orc	WyvernRush	38.0% (76/200)
Orc	Gruntapult	31.0% (62/200)
Human	FootmanCaster	92.7% (738/796)
Human	GryphonRush	14.4% (29/202)
Human	FastExpand	45.8% (160/349)
Human	RifleSorc	22.5% (45/200)
Night Elves	ArcherTalon	90.7% (604/666)
Night Elves	BearDryad	75.0% (835/1114)
Night Elves	FastBears	74.0% (594/803)
Night Elves	HuntressRush	76.5% (153/200)
Undead	FiendStatue	86.3% (619/717)
Undead	FiendAbom	100.0% (166/166)
Undead	GhoulNuke	45.6% (228/500)
Undead	GhoulGarg	68.3% (410/600)
All	None	26.95% (539/2000)

Table 4.11: Results of 100 test runs using Naive Bayes classification, with the enhancement of 10 additional nodes representing the presence or absence of individual unit types in the build.

	Average Accuracy	Std Dev	Max Acc	Min Acc
All Data	65.6% (5842/8900)	3.65	75.3%	59.6%
None-labels Excluded	76.2% (5255/6900)	4.00	85.5%	68.1%

Table 4.12: Breakdown of Naive Bayes classifier with enhancement of 10 additional nodes results by strategy.

Race	Strategy	Average Accuracy
Orc	Gruntraider	99.7% (1193/1196)
Orc	WyvernRush	68.0% (136/200)
Orc	Gruntapult	61.5% (121/200)
Human	FootmanCaster	91.3% (734/804)
Human	GryphonRush	11.3% (22/194)
Human	FastExpand	46.8% (162/346)
Human	RifleSorc	23.5% (47/200)
Night Elves	ArcherTalon	92.7% (608/656)
Night Elves	BearDryad	73.6% (813/1105)
Night Elves	FastBears	76.3% (608/797)
Night Elves	HuntressRush	81.0% (162/200)
Undead	FiendStatue	92.0% (651/707)
Undead	FiendAbom	100.0% (100/100)
Undead	GhoulNuke	50.6% (253/500)
Undead	GhoulGarg	68.2% (409/600)
All	None	29.35% (587/2000)

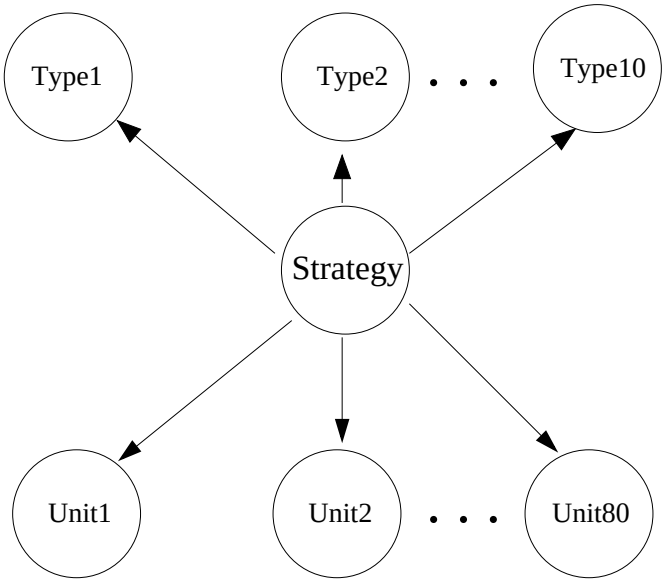

Figure 4.2: The structure of the enhanced Naive Bayesian classifier for strategy prediction. Each UnitX node represents the character at position X in the game description, and each TypeX node is a boolean value representing the presence or absence of a specific unit type.

4.3.5 Naive Bayes with Boosting Methodology

For our final machine learning experiments, we applied the ADABoost algorithm to our Naive Bayes classifier in order to generate a boosted Naive Bayes model. The structure of the network was exactly the same as the second, enhanced Naive Bayes classifier in Section 4.3.4, the only difference being that for each of the 100 iterations in the test run, we would apply the ADABoost algorithm with M equal to 5 so as to generate five Naive Bayes classifiers with identical structures but different CPTs and a weight for each of these classifiers. For each sample of the test set, the 5 classifiers then cast a weighted vote, assigning the class with the highest weighted total to the test sample.

The results from this experiment are presented in table 4.13. As before, we also present a listing of the results broken down by strategy in Table 4.14. As can be observed, the boosting algorithm in this case actually led to worse performance than the unboosted Naive Bayes classifier.

Table 4.13: Results of 100 test runs using Naive Bayes classification, using the ADABoost boosting algorithm with M equal to 5.

	Average Accuracy	Std Dev	Max Acc.	Min Acc.
All Data	62.5% (5562/8900)	4.10	71.9%	53.9%
None-labels Excluded	70.6% (4873/6900)	4.90	82.6%	59.4%

Table 4.14: Breakdown of the boosted Naive Bayes classifier M equal to 5 for the ADABoost boosting algorithm.

Race	Strategy	Average Accuracy
Orc	Gruntraider	91.5% (1103/1205)
Orc	WyvernRush	64.5% (129/200)
Orc	Gruntapult	53.0% (106/200)
Human	FootmanCaster	89.1% (714/801)
Human	GryphonRush	16.3% (34/208)
Human	FastExpand	47.6% (167/351)
Human	RifleSorc	28.5% (57/200)
Night Elves	ArcherTalon	84.9% (555/654)
Night Elves	BearDryad	65.5% (719/1098)
Night Elves	FastBears	71.4% (573/803)
Night Elves	HuntressRush	63.5% (127/200)
Undead	FiendStatue	84.3% (602/714)
Undead	FiendAbom	99.6% (154/155)
Undead	GhoulNuke	46.2% (231/500)
Undead	GhoulGarg	66.3% (398/600)
All	None	34.45% (689/2000)

4.4 Constructing Game-Balance Matrices

Now that we have presented machine learning techniques for automatically classifying a game's strategy, we can go directly from raw game data to a game matrix as discussed in Chapter 2. This section illustrates that process. An analysis of our experimental results will follow in Section 4.5.

We present the matrices in Tables 4.15, 4.16 and 4.17 that are constructed using the data and results from the experimental methods above, according to the definitions outlined in Chapter 2. For our example here, each matrix represents the Warcraft match-up of the Orc faction as the row player against the Night Elves faction as the column player — in our dataset, we have 14 games where this match-up occurs. Each row and column corresponds to one of the high-level strategies identified in Section 4.1; strategies that did not occur in the Orc against Night Elf match-up (such as the Orc WyvernRush strategy) are not shown in the matrices. As before, the payoffs a_{ij} are to the row player and are calculated according to equation 4.3, where $Wins_{ij}$ indicates the number of times a player playing strategy i defeated strategy j, and $Losses_{ij}$ indicates the number of times strategy i lost against strategy j. Table 4.15 is constructed using the labels provided by human experts; Table 4.16 is constructed using labels obtained through Needleman-Wunsch sequence alignment; and Table 4.17 is constructed using labels obtained from Naive Bayes classification. In each case, the number in parantheses next to each payoff a_{ij} represents the number of games in the data that fell into cell ij.

$$a_{ij} = \frac{Wins_{ij} - Losses_{ij}}{Wins_{ij} + Losses_{ij}} \tag{4.3}$$

This has the desired result that if $Strategy_i$ always defeats $Strategy_j$, the payoff is 1; if is always loses, the payoff is -1; and if it is just as likely to win as to lose, the payoff is 0.

These matrices demonstrate the manner in which game balance matrices might be constructed, but with only 14 games, they are of limited use. Some cells differ as much as going from a 0 to payoff to a 1.0, but only because many of the zeroes in

each table exist because there was no data in the cell. In fact, except for the very popular ArcherTalon and Gruntraider strategies, only one or two examples existed per strategy match-up in our very sparse dataset.

Using a small amount of labelled data in order to classify a large amount of unlabelled data and using the total results to build the matrices would seem likely to produce more useful results. To illustrate what results of this sort would look like, we present the game matrix in Table 4.18. For this matrix, we use the 3-nearest neighbor Needleman-Wunsch alignment approach, as it demonstrated the best overall performance of the machine learning approaches we examined. The training set is the same as for our experiment above; however, for constructing this matrix, we obtained 100 unlabelled samples specifically of the Orc against Night Elf match-up, none of which were part of our previous test or training set. Of these 100 games, 54 were won by Orc players and 46 were won by Night Elf players. As these 100 games were not examined by human experts, we cannot construct the corresponding matrix using the 'true' labels; however, the matrix we present serves as an example of the ultimate goal of the synthesis of all the techniques examined in this thesis to date.

From a matrix such as Table 4.18, a designer could extract various pieces of useful information. For instance, the value of the game is 0.04 indicating a slight advantage for the Orc players, as manifested in the slightly higher number of Orc wins in the dataset. Furthermore, the optimal strategy for the Orc player is to always play the Gruntraider strategy, whereas for the Night Elf player it is optimal to always play the ArcherTalon strategy. If the results of this matrix are accurate, we would expect to see the largest number of games fall into this cell, and that is indeed what we see in the data. However, the game theoretic techniques used to determine the optimal strategies of the game do not take into account the number of games used, only the payoff values in the matrix. This implies that the matrix is a reasonably good model of the true properties of the game. We also see that while the Orc WyvernRush strategy is useful against some strategies, it is almost never used as it is weak against the very common Night Elf ArcherTalon strategy. Based

Table 4.15: A game balance matrix, with rows and columns constructed using strictly the labelling provided by human experts over 14 games.

	HuntressRush	ArcherTalon	BearDryad	FastBears	None
Gruntraider	0 (0)	0 (8)	0 (0)	-1.0 (1)	1.0 (1)
Gruntapult	-1.0 (1)	0 (2)	-1.0 (1)	0 (0)	0 (0)
None	0 (0)	0 (0)	1.0 (1)	1.0 (1)	1.0 (1)

Table 4.16: A game balance matrix, with rows and columns constructed using the labelling provided by Needleman-Wunsch sequence alignment over 14 games.

	HuntressRush	ArcherTalon	BearDryad	FastBears	None
Gruntraider	-1.0 (1)	0.25 (7)	-0.33 (3)	0 (0)	0 (0)
Gruntapult	0 (0)	1.0 (1)	0 (0)	1.0 (1)	0 (0)
None	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)

Table 4.17: A game balance matrix, with rows and columns constructed using Naive Bayes classification over 14 games.

	HuntressRush	ArcherTalon	BearDryad	FastBears	None
Gruntraider	-1.0 (1)	0.11 (9)	0 (0)	0 (2)	0 (0)
Gruntapult	0 (0)	1.0 (1)	0 (0)	0 (0)	0 (0)
None	0 (0)	1.0 (1)	0 (0)	0 (0)	0 (0)

on these results, a designer might infer that the effectiveness of the Orc Gruntraider strategy needs to be slightly reduced so as to move the value of the game closer to 0, improving external balance, and to make other strategies more viable for both sides, improving internal balance.

4.5 Results and Analysis

The results from Section 4.3 show a total average accuracy of 55% including the 'None' labels, and 77% excluding the 'None' labels, for 3-nearest neighbors sequence alignment, and average accuracy of 65% including the 'None' labels, and 75% excluding the 'None' labels, for Naive Bayes. We recall that the 'None' labels refer to games for which the human expert indicated no high-level strategy was used during

Table 4.18: A game balance matrix, with rows and columns constructed using the kNN Needleman-Wunsch alignment classifier, run on 100 unlabelled Orc vs. Night Elf games.

	HuntressRush	ArcherTalon	BearDryad	FastBears
Gruntraider	0.1 (20)	0.04(25)	0.56 (9)	0.67 (6)
Gruntapult	-0.16 (19)	0 (16)	-0.09 (11)	-0.2 (15)
WyvernRush	0.33 (3)	-0.33 (3)	0 (0)	1.0 (1)

the game. As can be seen, this makes the two methods comparable when we exclude the 'None' labels, but Naive Bayes handles this trickier case better than sequence alignment. Naive Bayes also has a significantly lower standard deviation, which is expected since half of the data is used for training rather than a single exemplar in the case of sequence alignment. Applying the boosting algorithm to Naive Bayes only decreased the prediction accuracy. Schapire admits that in some cases, boosting can lead to overfitting [Schapire, 2001]; furthermore, in our case, the original model often performed very well on its own training data, making the boosting process unlikely to result in any performance gains. It seems likely these factors explain the failure of the boosting algorithm in our situation.

Analyzing the results when broken up by strategy shows that the Naive Bayes classifier is only really performing well on the more common strategies in the dataset. Essentially, this is because the classifier learns that if it always chooses the common labels, it will be right most of the time. Therefore, the accuracy on common strategies is very high whereas it is much lower for uncommon strategies: as low as 11% in some cases. 3-nearest neighbor sequence alignment, on the other hand, has no strategies other than the 'None' label below 60% accuracy and all but one are greater than 70%. This would seem to strongly indicate that the 3-nearest neighbor sequence alignment is the best tool for our problem out of those tried in this thesis, at least when the size of the dataset is small.

We also see that when a set of 'good' exemplars is heuristically chosen for sequence alignment, the accuracy of sequence alignment is comparable with both 3-nearest neighbors and Naive Bayes. Furthermore, we see that the best set of exemplars

that occurred during the 100 random runs of 3-nearest neighbor sequence alignment exceeds this value by more than 10%, achieving an accuracy of almost 91% when 'None' labels are excluded from the test set. Again, it is not surprising that the performance of sequence alignment is highly dependent on choosing a good set of exemplars.

We now return to the three questions posed at the beginning of this section. The pre-data survey and subsequent data classification by experts strongly implied that build sequences with common strategic elements identifiable by humans really do occur in the online competitive environment of Warcraft III, since every race had at least one major strategy that significantly outranked the 'None' option, and some (especially the Night Elves faction) had several. These results seem to validate at least the potential use of strategic abstractions as a way of thinking about and classifying game data.

The base classification results of 55% for sequence alignment and 61% for Naive Bayes are significantly lower than machine learning classification results in other domains, such as the 80% accuracy for handwritten digit recognition with 1960 training samples reported by Frey [Frey, 1998] and the 95% accuracy for junk email filtering reported by Sahami with 2500 training samples [Sahami et al., 1998]. The approach of increasing the size of the training set is a common solution in many machine-learning domains. With a vastly increased training set size of 60,000, Russel and Norvig show an accuracy of 98% or more for the handwritten digit problem, for a variety of contemporary machine learning techniques [Russell and Norvig, 2003]. However, as our results tables in this section show, many of our errors are due to erroneously assigning a label to a game that the human rater marked as having no particular strategy. When we exclude such errors, the classification accuracy is similar to, and in some cases slightly better than other results in domains such as handwriting recognition with similar amounts of training data; Frey reports an accuracy of 70% with 120 training examples, or about 12 examples per sample [Frey, 1998], while Rowley et al. report about 82% accuracy in classifying Japanese kanji characters [Rowley et al., 2002] using 10 samples per character. By comparison, our

training set for the Naive Bayes classifier is smaller yet, from 1 to 11 samples per strategy, depending on the popularity of the strategy. For our 3-nearest neighbor method, we use only 3 samples per strategy. Excluding the 'None' examples seems in some ways a more fair comparison, since the studies by Frey [Frey, 1998], Russell and Norvig [Russell and Norvig, 2003] and Rowley et al. [Rowley et al., 2002] do not include a 'random scribble' character among the test set, and it is our impression that it is not common practice in the handwriting domain to do so. These results suggest that merely increasing the size of the training set could result in accuracy comparable to simple attempts at character recognition, as Rowley shows an increase from 82% to 95% accuracy by increasing training set size one-hundred-fold [Rowley et al., 2002]. Frey reports on other classifiers that perform better in the handwriting domain, such as logistic autoregressive classifiers [Frey, 1998]; it is possible these other classification techniques would also result in performance gains in our domain, although for most of these the structure of the data may have to be changed.

In our case we must ask the questions of how much data is feasible, what is the best accuracy we can expect, and how much accuracy is really required. We recall that the purpose of this work is to provide tools to game designers so they can more easily diagnose balance problems during the design phases of their games. Although preliminary player tests (often called beta tests in the industry) can provide large amounts of raw game data, it seems to us unreasonable to expect a game designer could have access to 10,000 *labelled* game examples to use for training. Thus, for methods in this domain to be useful, they must be so with only limited amounts of training data available.

For the purpose of this thesis, our primary goal was to show first that high-level strategies exist, at least with sufficient frequency so as to be useful in the testbed game of Warcraft III, and secondly that they can be automatically identified through machine learning. The results of our experiments seem to indicate strongly that they can, even with very small datasets and very simple data models and machine learning techniques.

While we have sometimes compared this problem to other classification problems

such as handwritten character recognition, the domain has at least one important difference, and that is the class label itself is, by definition, abstract and does not necessarily objectively exist in all cases. In handwriting, the writer always has a real, objective intent for every character she writes, which in English is usually one of 26 different characters, regardless of whether or not the reader can correctly interpret it. Warcraft players, on the other hand, may have a particular strategy in mind or they may not. Their goal is simply to win the game, and strategic abstractions are useful to human players only insomuch as they help to organize the player's thoughts about the game. Strategic similarities that appear due to this behavior are emergent, rather than intentional on the parts of the players.

Because of the nature of this problem in that not all examples will have a clear class and some of the classes themselves may bear similarities and overlap to some degree, we would contend the most important aspect of this domain is correctly classifying the 'clear' examples, examples that a human player might describe as a 'textbook' implementation of the strategy. In other words, the baseline accuracy of the problem is not 100%; rather it is only as high as the percentage of human raters who would agree on a game's class. Even for digit recognition, for instance, Russell and Norvig report that human accuracy is only 97.5% [Russell and Norvig, 2003]. It seems unlikely to us that human 'accuracy' (or rather, consensus) for the problem of strategy classification would be as high as that; future study would be needed to set a more precise upper limit.

With the data collected for this study, it is difficult to objectively say how well the methods tested here succeed in the regard of classifying the 'clear' examples. One possible way to do this would have a large number of raters all rate the same set of games, and compare the classifier's accuracy on games on which the raters all agreed on the strategy versus ones on which the raters disagreed. With the small number of participants we had available, this was infeasible in our case. However, an entirely subjective viewing of the classification results suggest that correct classification of clear examples is indeed going on in many cases.

Another metric for evaluation in this domain that can be performed with the

Table 4.19: A measurement of the difference in alignment score and posterior probability between correctly and incorrectly classified samples, over 100 random test runs.

	Correct	Incorrect
Sequence Alignment	755	649
Sequence Alignment excluding None-labels	771	671
Naive Bayes	93.0%	87.2%
Naive Bayes excluding None-labels	94.1%	84.9%

available data is, in the sequence alignment case, to measure the average alignment score for both correct and incorrectly classified samples, and in the Bayes Net case, to measure the posterior probability for both correct and incorrect classifications. Table 4.19 shows the results of such a test, again using 100 random test runs for both methods.

Table 4.19 shows that in that in all cases, the scores or posterior probabilities in correctly classified samples are higher than for those that are incorrectly classified. This means that at least the samples that the machine-learning algorithms believe to be clear are being correctly classified.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we first proposed framing the problem of game balance using the language of game theory. We presented definitions for external and internal balance of a game and show how these properties can be detected using game theoretic tools. We then proposed the concept of strategic abstractions, that is to say high-level groupings of strategic competitive game play, in order to construct these matrices and perform an empirical study to demonstrate whether such abstractions really exist in game data, whether they can be automatically detected, and, if so, how to best go about it.

The study has shown that strategic abstractions can and do occur in real data and can be identified by experts, at least in the testbed domain of the Warcraft III real-time strategy game. Furthermore, even with very small datasets, our experimental work has shown that these strategies can also be identified automatically using simple machine learning techniques. Bioinformatics sequence alignment with a k-nearest neighbor classification approach can, with only 3 exemplars of each strategy, correctly identify the strategy used in 55% of cases using all data, and 77% of cases on data that experts indicated actually had a strategic class. Naive Bayes classification achieves similar results, with 65% accuracy on all data and 75% accuracy on data rated to have an actual class.

Finally, we demonstrated how these machine learning methods can be combined with game-theoretic tools to build game-balance matrices, which can then serve as a formal and analytic aid to the game design process. Taken together, these tools

provide for a principled means of analyzing and synthesizing the experience of a large number of expert human players, without the need to solicit each individual expert for an opinion. Rather than forcing them to rely on the potentially conflicting personal opinions of human players, we thus provide game developers with a snapshot of what is actually going on in their game, allowing them to leverage available game play data to improve their game design process.

5.2 Future Work

Section 4.5 discussed the possibility of performing cross-validating expert rating of game data in order to get an idea of how much experts agree on the label of a game. This would also allow the performance of the machine classification algorithms to be measured on examples which human raters considered very clear, based on the amount of consensus between raters. Such a study would also aid to set upper bounds of accuracy in this problem, based on the percentage of games on which there is consensus among the raters.

A particular problem of this work was the fact that 21% of games were rated not to possess a common and identifiable strategy and were thus given a 'None' label by the raters. Correctly classifying the 'None'-labelled examples was the single largest source of error in our experiments. There is substantial room for improvement in how best to correctly separate them from the rest of the data. It also raises the question of whether other domains, such as handwriting recognition, would encounter similar problems if analogous samples (random scribbles in the case of handwriting) were included in the test set for these problems.

Of course, it bears saying that Naive Bayes classification is one of the simplest classifier technologies available; as such, it was deemed suitable for a proof-of-concept study such as this one, but there are many other classifier technologies that could be applied to this problem. They key in this regard is to find one that works well with only small amounts of data.

Clustering is another techonology that could be applied to this problem. Clus-

tering techniques have been applied in domains ranging from music classification [Logan and Salomon, 2001] to gene-finding in bioinformatics [Eisen et al., 1998] and many others [Jain et al., 1999]. In particular, it could be very interesting to see whether unsupervised clustering can produce clusters that match labelled clusters of strategies. Currently, the techniques applied in this paper require that the different strategic classes be available via expert knowledge a priori. If clustering can successfully cluster games by similar strategy, not only can this expediate the process of assigning strategy labels to games, but could even result in designers being able to discover new strategies which players are using, but which had not been previously identified. The X-means algorithm by Pelleg and Moore [Pelleg and Moore, 2000] could be one approach to apply to this task.

Another entirely different direction would be to gather large amounts of unlabelled data and apply the game-theoretic methods discussed in this thesis to discover interesting results on player behavior. For instance, with the labelling techniques in this thesis, we can construct matrices of the type in section 4.4. Using these matrices, we can then easily calculate the Nash equilibria of the resulting games. We can then answer questions of whether or not the player population is actually playing the equilibria, or if a certain subset (say, the high-level players) are playing the equilibrium strategy. We can also determine how payoff values in the matrix change for individual players when they play against the full online population, and monitor how the effectiveness of each strategy differs from the average effectiveness of the strategy, as well as monitoring how the Nash equilibria of the game may change for individuals who are more or less skilled with certain strategies.

In this thesis, our experimental testbed was a real-time strategy game. Although our goal is to present general balance principles that can apply to all games, applying the techniques we discuss to games of radically different genres seems like a non-trivial task. The major challenges here are finding a suitable feature set to describe the game and identifying distinct high-level strategies for the purpose of strategic abstraction. In particular, these problems seem difficult for any game for which the focus is not a discrete set of strategic choices but rather an almost continuous

process, such as a car-racing game where the player is continually trying to maintain maximum speed by staying on the road and avoiding obstacles. Investigating the application of our techniques to such an environment is another potential area of future study.

Yet another angle to examine is the problem of balance at different levels of player skill. Game theory itself makes the assumption that all agents are perfect players and know the optimal strategy to a game. In reality, it is questionable whether experts are in fact 'perfect' players, and novices certainly are not. Developing definitions of balance that can hold across all levels of player skill seems like a non-trivial task, but ensuring that a game is balanced at the novice level of play is perhaps more important for some games than game balance at the expert level. This is because no player of any game is initially an expert, and many may never become one. Therefore, if some game faction or element seems too weak at a novice level of skill, players may become frustrated and never invest the necessary time to discover that the game element in question is indeed balanced once they discover how to properly use it, since they will quit playing the game long before they have the skill to make this realization.

Finally, the ultimate desired outcome of this work would be a complete set of tools that help designers first design games, then analyze them for balance properties, detect balance problems and then allow the designer to modify the game in a way that corrects the problems. While the game theoretic matrix properties and strategic abstractions proposed in this paper facilitate analysis and detection of balance problems, and can suggest solutions at a very abstract level, a more direct mapping between the game balance matrices and the actual game changes a designer needs to make would be a useful contribution to complete this cycle.

Bibliography

Ernest Adams. A symmetry lesson. *Gamasutra, www. gamasutra. com*, 2, 1998.

Ernest Adams. Balancing games with positive feedback. *Gamasutra, www. gamasutra. com*, 2002.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI–BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997. URL `citeseer.ist.psu.edu/altschul97gapped.html`.

Salman Azhar, Andrew McLennan, and John H. Reif. Computation of equilibria in noncooperative games, duke university technical report cs-1991-36. In *Proceedings of the Workshop for Computable Economics*, December 1992.

BattleReports. `http://www.battlereports.com`, 2006.

E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for your Mathematical Plays*. New York: Academic Press, 1982.

Blizzard Entertainment. Starcraft real-time strategy game, 1998.

Blizzard Entertainment. `http://www.blizzard.com`, 2005.

Daniel Bozinov and Jorg Rahnenfuhrer. Unsupervised technique for robust target separation and analysis of dna microarray spots through adaptive pixel clustering. *Bioinformatics*, 18(5):747–756, 2002.

M. Brudno and B. Morgenstern. Fast and sensitive alignment of large genomic sequences. In *Proceedings of IEEE Computer Society Bioinformatics Conference, Stanford University, California*, pages 138–147, August 2002.

M. Buro. How machines have learned to play othello. *IEEE Intelligence Systems Journal*, 14:12–14, 1999.

Tom Cadwell. Techniques for achieving play balance. *Gamedev.net, www. gamedev. net* , 2002.

Adam Carpenter. Applying risk analysis to play-balance rpgs. *Gamasutra, www. gamasutra. com*, 2003.

Chessgames. `http://www.chessgames.com/chessstats.html`, April 2005.

G.F. Cooper. Probabilistic inference using belief networks is NP-hard. *Artificial Intelligence*, 42:393–405, 1990.

A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucl. Acids. Res.*, 27(11):2369–2376, 1999. URL `citeseer.ist.psu.edu/delcher99alignment.html`.

M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proceedings of the National Academy of Sciences of the USA*, volume 95, pages 14863–14868, 1998.

Michael D. Ernst. Playing Konane mathematically: A combinatorial game-theoretic analysis. *UMAP: The Journal of Undergraduate Mathematics and its Applications*, 16, 1995.

Daniel Fetter. `http://www.cs.iastate.edu/~dfetter/`, 2005.

E. Fix and J.L. Hodges. Discriminatory analysis. nonparametric discrimination. consistency properties. *Technical Report 4, US Air Force School of Aviation Medicine. Randolph Field, TX.*, 1957.

A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for n*n chess requires time exponential in n. In *Proc. 8th Int. Coll. Automata, Languages, and Programming*, pages 278–293, 1981.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

Brendan J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

Juliusz Gonera. Warcraft 3 replay parser, `http://toya.net.pl/~julas/w3g/`, 2003.

D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.

D. Heckerman. A tutorial on learning with bayesian networks. *Technical Report MSR-TR-95-06. Microsoft Corporation, Redmond, USA*, 1996b.

Robert C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

Dimitrios Kalles and Panagiotis Kanellopoulos. On verifying game designs and playing strategies using reinforcement learning. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 6–11, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-287-5.

David Kennerly. Better game design through data mining. *Gamasutra, www.gamasutra.com*, 2003.

Craig A. Knoblock. Automatically generating abstractions for planning. In *Artificial Intelligence, 68(2)*, pages 243–302, 1994.

D.E. Knuth and R. E. Moore. An analysis of alpha beta pruning. *Artificial Intelligence*, 6:293–326, 1975.

Alexander Kovarsky and Michael Buro. Heuristic search applied to abstract combat games. In *Canadian Conference on AI 2005*, pages 66–78, 2005.

P. Krause. Learning probabilistic networks. *http://www.auai.org/bayesUSkrause.ps.gz*, 1998.

Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

B. Logan and A. Salomon. A music similarity function based on signal analysis, 2001. URL `citeseer.ist.psu.edu/logan01music.html`.

S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

Norsys Netica. *http://www.norsys.com*.

Guillermo Owen. *Game Theory*. Academic Press, third edition, 1995.

Christos H. Papadimitriou. Computing correlated equilibria in multi-player games. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, 2005.

Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc, 1998.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of 17th International Conference on Machine Learning*, pages 727–734, 2000.

R. Rouse III. *Game Design: Theory and Practice*. Wordware Publishing, second edition, 2005.

Henry A. Rowley, Manish Goyal, and John Bennett. The effect of large training set sizes on online japanese kanji and english cursive recognizers. In *Proceedings of the 8th International Conference on Frontiers in Handwriting Recognition*, pages 36–40, 2002.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05. URL `citeseer.ist.psu.edu/sahami98bayesian.html`.

A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*, pages 210–229, 1959.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2): 197–227, 1990.

Robert E. Schapire. The boosting approach to machine learning: An overview. *In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, Mar. 2001. See* `http://stat.bell-labs.com/who/cocteau/nec/` *and* `http://www.research.att.com/~schapire/boost.html`, 2001.

Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

Robert E. Schapire, Peter Stone, David McAllester, Michael L. Littman, and Janos A. Csirik. Modeling auction price uncertainty using boosting-based conditional density estimation. In *Machine Learning: Proceedings of the Nineteenth International Conference*, pages 143–160, 2002.

C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

T. Smith and M. Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, pages 195–197, 1981.

C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.

D.J. States, W. Gish, and S.F. Altschul. Improved sensitivity in nucleic acid database searches using application-specific scoring matrices. *Methods: A Companion to Methods in Enzymology*, 3(1):66–70, 1991.

G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38:58–67, 1995.

G. Tesauro and T. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390, 1989.

Sebastian Thrun. Learning to play the game of chess. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 1069–1076. The MIT Press, Cambridge, MA, 1995.

Gokhan Tur, Robert E. Schapire, and Dilek Hakkani-Tur. Active learning for spoken language understanding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 276–279, 2003.

WCReplays. `http://www.wcreplays.com`, 2006.

WCStrategy. Warcraft strategy `http://www.warcraftstrategy.com`, 2006.

Appendix A

The Warcraft III Ladder

The Warcraft III: The Frozen Throne ladder system is an online system provided by Blizzard Entertainment [Blizzard Entertainment, 2005] to facilitate online play of the game. The system features an Automatic Match-Making system, which matches players for competitive play based on the perceived skill level of each. In general, the system will try to match players who are perceived to be relatively equal in skill.

The ladder also has an experience-based system, the intention of which is to allow more successful players to "climb" the ladder and ultimately provide a player ranking system. Each player is assigned a level which represents his or her skill, and the level increases as the player plays and wins games. New players start at level 1, with the maximum level attainable being 50.

Appendix B

Pre-Data Survey

The following is the pre-data survey that was distributed to players at the outset of the experimental portion of this study.

Warcraft III: The Frozen Throne

Strategy Questionnaire

The following questionnaire is for a Computer Science Masters thesis project. The topic of this project is automatically identifying high-level player strategies in competitive, multi-player games. The research is not about your answers to this questionnaire but rather about the computers ability to identify strategies; your answers are needed to help measure how well the computer is doing. Both your identity and your answers will be kept anonymous, and your participation in this study is greatly appreciated.

When complete, please return this form to the mailbox of:

Jeff Long

Department of Computer Science,

176 Thorvaldson Building,

110 Science Place

Instructions

Under each playable race from Warcraft III: The Frozen Throne, several major strategies (sometimes referred to as builds) are listed. The term major is deliberately rather vague, but is intended to refer to a short, high-level description that you might use to capture the important strategic elements of the game. For each of the listed strategies, please indicate whether you agree or disagree (yes or no) that it is a viable and commonly used build by experienced Warcraft III players. In addition, if you feel there is a common build worth mentioning that has not been included in the current list, please add it under the heading *Other* at the bottom of each races section. Similarly, if you feel something important has been left out of one of the existing strategy write-ups, please feel free to add it.

Orc

1. **Gruntapult:** A significant build-up of early Grunts; Demolishers enter production as soon as possible; an attack mounted on the enemy base before Tier 2 production buildings can really kick in:
    - Yes:
    - No:
2. **Gruntraider:** An early build-up of Grunts; Beastiary built as soon as possible; Production of several raiders with Ensnare:

- Yes:
- No:

3. **Wyvern Rush:** Minimal build-up of early units; base is compactly built to resist a rush; double Beastiaries built in Tier 2; significant Wind Riders (or Wyverns as they were once called) production:
   - Yes:
   - No:
4. **Double Lodge:** Medium early game build-up; emphasis on Spellcaster production at Tier 2, usually focusing on Shaman and Witchdoctors. Often fast upgrade to Bloodlust is a priority:
   - Yes:
   - No:
5. **Other:**

Human

1. **Rifle-Sorc:** Little to no footman production; Early blacksmith and rifleman production begins as soon as possible. Sorceresses are added at Tier 2, often from double Arcane Sanctums:
   - Yes:

- No:

2. **Gryphon Rush:** Base is heavily fortified with towers; Usually involves Hero harass with Archmage and/or Bloodmage; Gryphon and Dragonhawk production begins as soon as possible.

    - Yes:

    - No:

3. **Fast Expand:** Heavy early peasant production; militia used to help clear a nearby goldmine; additional peasants then used to quickly power-build a new Townhall; Expansion is usually fortified with towers:

    - Yes:

    - No:

4. **Other:**

Night Elf

1. **Archer-Talon:** Significant Archer production at Tier 1, usually with a Demon Hunter hero to soak damage; Druid of the Talon production begins as soon as possible; massing of Archers and Talons continues:

    - Yes:

    - No:

2. **Huntress Rush:** No Archer production, very early Huntress Hall; Huntress production begins, usually from double Ancients of War; emphasis is on seizing map control and often an early resource expansion:

    - Yes:
    - No:

3. **Fast Bears:** Ancient of War is often skipped at Tier 1; heavy Hero harassment, usually by a Demon Hunter or Warden; Double Ancients of Lore produced at Tier 2; Druid of the Claw production begins shortly, sometimes with Dryad production first:

    - Yes:
    - No:

4. **Other:**

Undead

1. **Fiend-Statue:** Ghouls produced only for lumber, Graveyard is built very early; Crypt Fiend production begins as soon as possible; Slaughterhouse built immediately at Tier 2, several statues are produced; upgrade to Destroyers later is possible:

    - Yes:
    - No:

2. **Necro Rush:** A small Ghoul creeping force is produced; upgrade to Tier 2 is usually early; Necromancer production begins in conjunction with a Slaughterhouse and Meatwagons; an attack is mounted on the enemy base before significant dispel magic is available:

    - Yes:
    - No:

3. **Ghoul-Garg:** Significant Ghoul production at Tier 1; Hero combination is usually Death Knight and Dread Lord; Gargoyle production begins at Tier 2, usually out of double Crypts:

    - Yes:
    - No:

4. **Other:**

Appendix C

Questionnaire

The following are the instructions to the questionnaire that were provided to participants along with the game replays they were asked to classify.

Warcraft III: The Frozen Throne

Replay Classification Instructions

The following questionnaire is for a Computer Science Masters thesis project. The topic of this project is automatically identifying high-level player strategies in competitive, multi-player games. The research is not about your answers to this questionnaire but rather about the computer's ability to identify strategies; your answers are needed to help measure how well the computer is doing. Both your identity and your answers will be kept anonymous, and your participation in this study is greatly appreciated.

When complete, please return this form to the mailbox of:

Jeff Long

Department of Computer Science,

176 Thorvaldson Building,

110 Science Place

Instructions

Under each playable race from Warcraft III: The Frozen Throne, several major strategies (sometimes referred to as builds) are listed. The term 'major' is deliberately rather vague, but is intended to refer to a short, high-level description that you might use to capture the important strategic elements of the game. You will be sent a series of replays by high-level Warcraft III players. For each replay, please indicate which strategy from the available list you believe that player to be using. You may list more than one strategy if you feel this is appropriate. Please do this for both players in each replay. For your reference, a full list of strategies is included in this document along with their descriptions. The accompanying classification document will list the strategies by name and number only.

Orc

1. **Gruntapult:** A significant build-up of early Grunts; Demolishers enter production as soon as possible; an attack mounted on the enemy base before Tier 2 production buildings can really kick in.

2. **Gruntraider:** An early build-up of Grunts; Beastiary built as soon as possible; Production of several raiders with Ensnare. Spirit Walkers are often added later.

3. **Wyvern Rush:** Minimal build-up of early units; base is compactly built to resist a rush; double Beastiaries built in Tier 2; significant Wind Riders (or Wyverns as they were once called) production.

4. **Tauren Chief Headhunters:** A risky strategy consisting of a first-hero Tauren Chieftain. Headhunters are produced in large numbers and upgraded to berzerkers swiftly. The Chieftain keeps the enemies stunned while the Headhunters do the damage. Taurens are sometimes eventually added later in the game.

5. **Firelord Rush:** Significant Grunt build-up early on. Starting hero is usually Farseer or Shadow Hunter with Serpent Wards. The Firelord is added at Tier 2 to mount an immediate attack on the enemy base.

Human

1. **Rifle-Sorc:** Little to no footman production; Early blacksmith and rifleman production begins as soon as possible. Sorceresses are added at Tier 2, often from double Arcane Sanctums. Sometimes complimented by Mortar Teams.

2. **Gryphon Rush:** Base is heavily fortified with towers; Usually involves Hero harass with Archmage and/or Bloodmage; Gryphon and Dragonhawk production begins as soon as possible.

3. **Fast Expand:** Heavy early peasant production; militia used to help clear a nearby goldmine; additional peasants then used to quickly power-build a new

Townhall; Expansion is usually fortified with towers.

4. **Footman-Caster:** Frontline of footmen built up early. Spellcasters of all types produced at Tier 2. Often a 3rd Hero is added at Tier 3.

5. **Breaker-Priest-Knight:** Usually begins with modest Footman production. Arcane Sanctums at Tier 2 produce Spellbreakers and priests to create a highly defensive army. Footmen are replaced with Knights as they become available.

6. **Tank Bait:** Early footmen are produced for defense along with towers. Steam Tanks (or Siege Engines as they're now called) are secretly massed to destroy enemy buildings.

Night Elf

1. **Archer-Talon:** Significant Archer production at Tier 1, usually with a Demon Hunter hero to soak damage; Druid of the Talon production begins as soon as possible; massing of Archers and Talons continues.

2. **Huntress Rush:** No Archer production, very early Huntress Hall; Huntress production begins, usually from double Ancients of War; emphasis is on seizing map control and often an early resource expansion.

3. **Fast Bears:** Ancient of War is often skipped at Tier 1; heavy Hero harassment, usually by a Demon Hunter or Warden; Double Ancients of Lore produced at Tier 2; Druid of the Claw production begins shortly, sometimes with Dryad production first.

4. **Ranger-Moon:** Priestess of the Moon is chosen as first Hero, with Searing Arrows and Trueshot Aura. Tier 1 units of Huntresses and Archers are massed to gain an early advantage. Focus fire on enemy units with this army is a priority.

5. **Bear-Dryad:** Similar to Fast Bears in that dual Ancients of Lore are involved; however, Tier1 production is usually not skipped. Huntresses are produced early game and supplemented with Dryads. Upgrade to Bears comes somewhat slower but they eventually replace the Huntress Frontline. Usually Staves of Teleportation are involved to save dying bears.

Undead

1. **Fiend-Statue:** Ghouls produced only for lumber, Graveyard is built very early; Crypt Fiend production begins as soon as possible; Slaughterhouse built immediately at Tier 2, several statues are produced; upgrade to Destroyers later is possible.

2. **Necro Rush:** A small Ghoul creeping force is produced; upgrade to Tier 2 is usually early; Necromancer production begins in conjunction with a Slaughterhouse and Meatwagons; an attack is mounted on the enemy base before significant dispel magic is available.

3. **Ghoul-Garg:** Significant Ghoul production at Tier 1; Hero combination is usually Death Knight and Dread Lord; Gargoyle production begins at Tier 2, usually out of double Crypts.

4. **Ghoul-Nuke:** Primarily a Hero-nuking strategy. The Death Knight is produced at Tier 1 along with Ghouls. The Lich is added at Tier 2 for Frost Nova. Upgrade to Tier 3 as soon as possible for Ghoul Frenzy.

5. **Fiend-Abom:** Usually opens as a Cryptfiend build with a Death Knight. Slaughterhouses are constructed at Tier 2, and began producing Abominations as soon as Tier 3 is reached. The Death Knights coil effectively heals both units. Sometimes banshees are added if resources allow.

Appendix D

Sequence Alignment Scoring Matrices

The following are the similarity scoring matrices used for the sequence alignment portion of the experimental study. They are presented by Warcraft III race, since each race has different types of units available and thus a different encoding. For all races, the underscore (_) character represents the gap character in the alignment.

Table D.1: The character encoding key for Orc and Human military units.

Orc Units		Human Units	
Game Unit	Character	Game Unit	Character
Grunt	G	Footman	F
Headhunter	H	Rifleman	R
Catapult	C	Knight	K
Raider	R	Sorceress	S
Windrider	W	Priest	P
Batrider	B	Spellbreaker	B
Kodo Beast	K	MortarTeam	M
Shaman	S	FlyingMachine	L
WitchDoctor	D	SteamTank	T
Spiritwalker	I	Dragonhawk	D
Tauren	T	Gryphonrider	G

Table D.2: The character encoding key for Undead and Night Elf military units.

Undead Units		Night Elf Units	
Game Unit	Character	Game Unit	Character
Ghoul	G	Archer	A
Crypt Fiend	F	Huntress	H
Gargoyle	R	Glaive Thrower	G
Necromancer	N	Dryad	D
Banshee	B	Druid of the Claw (Bear)	B
Obsidian Statue	O	Mountain Giant	M
Abomination	A	Hippogryph	I
Meat Wagon	M	Druid of the Talon	T
Destroyer	D	Faerie Dragon	F
Frost Wyrm	W	Chimaera	C

Table D.3: The scoring matrix used for Orc strategy classification.

	G	H	C	R	W	B	K	S	D	I	T	_
G	48	-2	-2	-2	-5	-5	-3	-3	-3	-2	-4	-3
H	-2	32	-3	-3	-6	-4	-4	-2	-2	-3	-5	-2
C	-2	-3	64	-3	-4	-6	-2	-4	-4	-3	-3	-4
R	-2	-3	-3	48	-4	-4	-2	-3	-3	-2	-4	-3
W	-5	-6	-4	-4	64	-3	-3	-6	-6	-5	-5	-4
B	-5	-4	-6	-4	-3	32	-5	-4	-4	-5	-7	-2
K	-3	-4	-2	-2	-3	-5	64	-4	-4	-3	-3	-4
S	-3	-2	-4	-3	-6	-4	-4	32	-2	-2	-5	-2
D	-3	-2	-4	-3	-6	-4	-4	-2	32	-2	-5	-2
I	-2	-3	-3	-2	-5	-5	-3	-2	-2	48	-4	-3
T	-4	-5	-3	-4	-5	-7	-3	-5	-5	-4	80	-5

Table D.4: The scoring matrix used for Human strategy classification.

	F	R	K	S	P	B	M	L	T	D	G	_
F	32	-2	-3	-2	-2	-3	-3	-5	-3	-5	-6	-2
R	-2	48	-2	-3	-3	-2	-2	-6	-2	-4	-5	-3
K	-3	-2	64	-4	-4	-3	-3	-7	-3	-3	-4	-4
S	-2	-3	-4	32	-1	-2	-3	-5	-3	-5	-6	-2
P	-2	-3	-4	-1	32	-2	-3	-5	-3	-5	-6	-2
B	-3	-2	-3	-2	-2	48	-2	-6	-2	-4	-5	-3
M	-3	-2	-3	-3	-3	-2	48	-5	-2	-4	-5	-3
L	-5	-6	-7	-5	-5	-6	-5	16	-5	-4	-5	-1
T	-3	-2	-3	-3	-3	-2	-2	-5	48	-4	-5	-3
D	-5	-4	-3	-5	-5	-4	-4	-4	-4	48	-2	-3
G	-6	-5	-4	-6	-6	-5	-5	-5	-5	-2	64	-4

Table D.5: The scoring matrix used for Undead strategy classification.

//	G	F	R	N	B	O	A	M	D	W	_
G	32	-2	-3	-2	-2	-3	-4	-4	-7	-9	-2
F	-2	48	-4	-3	-3	-2	-3	-3	-6	-8	-3
R	-3	-4	32	-4	-4	-5	-6	-6	-5	-7	-2
N	-2	-3	-4	32	-2	-3	-4	-4	-7	-9	-2
B	-2	-3	-4	-2	32	-3	-4	-4	-7	-9	-2
O	-3	-2	-5	-3	-3	48	-2	-2	-5	-8	-3
A	-4	-3	-6	-4	-4	-2	64	-2	-4	-7	-4
M	-4	-3	-6	-4	-4	-2	-2	64	-4	-7	-4
D	-7	-6	-5	-7	-7	-5	-4	-4	80	-4	-5
W	-9	-8	-7	-9	-9	-8	-7	-7	-4	112	-7

Table D.6: The scoring matrix used for Night Elf strategy classification.

//	A	H	G	D	B	M	I	T	F	C	_
A	32	-2	-2	-3	-4	-7	-4	-2	-4	-7	-2
H	-2	48	-2	-2	-3	-6	-5	-3	-5	-6	-3
G	-2	-2	48	-2	-3	-6	-5	-3	-5	-6	-3
D	-3	-2	-2	48	-2	-5	-5	-3	-5	-6	-3
B	-4	-3	-3	-2	64	-4	-6	-4	-7	-5	-4
M	-7	-6	-6	-5	-4	112	-9	-7	-9	-6	-7
I	-4	-5	-5	-5	-6	-9	32	-3	-2	-5	-2
T	-2	-3	-3	-3	-4	-7	-3	32	-3	-7	-2
F	-4	-5	-5	-5	-7	-9	-2	-3	32	-5	-2
C	-7	-6	-6	-6	-5	-6	-5	-7	-5	80	-5