Automatic Generation and Evaluation of Recombination Games

Cameron Browne

A dissertation submitted in partial fulfilment of the requirements for the degree of:

Doctor of Philosophy

Faculty of Information Technology
Queensland University of Technology

February 2008

Abstract

Many new board games are designed each year, ranging from the unplayable to the truly exceptional. For each successful design there are untold numbers of failures; game design is something of an art. Players generally agree on some basic properties that indicate the quality and viability of a game, however these properties have remained subjective and open to interpretation.

The aims of this thesis are to determine whether such quality criteria may be precisely defined and automatically measured through self-play in order to estimate the likelihood that a given game will be of interest to human players, and whether this information may be used to direct an automated search for new games of high quality. Combinatorial games provide an excellent test bed for this purpose as they are typically deep yet described by simple well-defined rule sets.

To test these ideas, a game description language was devised to express such games and a general game system implemented to play, measure and explore them. Key features of the system include modules for measuring statistical aspects of self-play and synthesising new games through the evolution of existing rule sets.

Experiments were conducted to determine whether automated game measurements correlate with rankings of games by human players, and whether such correlations could be used to inform the automated search for new high quality games. The results support both hypotheses and demonstrate the emergence of interesting new rule combinations.

Keywords: Combinatorial, Games, Design, Aesthetics, Evolutionary, Search.

Contents

Abstract ii
Contents iii
List of Figures vi
List of Tables viii
List of Code Listings ix
List of Appendices x
Statement of Original Authorship xi
Acknowledgements xii

1 Introduction 1
1.1 Overview 1
1.2 Background to the Research 1
1.3 Research Problem and Hypotheses 1
1.4 Justification for the Research 1
1.5 Methodology 2
1.6 Outline of the Thesis 2
1.7 Scope and Key Assumptions 3
1.8 Contributions 4
1.9 Summary 4

Part I: Background Theory 5

2 Games in General 7
2.1 Introduction 7
2.2 Definitions 7
2.3 Types of Combinatorial Games 10
2.4 General Games 12
2.5 Summary 16

3 Search Methods 17
3.1 Introduction 17
3.2 Move Planning 17
3.3 Evolutionary Algorithms 22
3.4 Summary 26

4 Aesthetics in Game Design 27
4.1 Introduction 27
4.2 Aesthetics 27
4.3 Game Design 31
4.4 Summary 35

Part II: Ludi Framework 37

5 The Ludi Game Description Language 39
5.1 Introduction 39
5.2 Model 39
5.3 The Language 40
5.4 Custom Ludemes 56
5.5 Summary 57

6 The Ludi General Game Player....59
6.1 Introduction....59
6.2 Model....59
6.3 Rules Parser....60
6.4 Game Object....61
6.5 User Interface....65
6.6 Play Manager....68
6.7 Summary....73

7 Strategy Module....75
7.1 Introduction....75
7.2 Model....75
7.3 Advisors....76
7.4 Policies....86
7.5 Policy Optimisation....89
7.6 Move Priorities....93
7.7 Summary....94

8 Criticism Module....95
8.1 Introduction....95
8.2 Model....95
8.3 Operation....97
8.4 Aesthetic Criteria....101
8.5 Summary....124

9 Synthesis Module....125
9.1 Introduction....125
9.2 Model....125
9.3 Initial Population....127
9.4 Parent Selection....127
9.5 Recombination....128
9.6 Mutation....129
9.7 Rule Safety....130
9.8 Baptism....133
9.9 Speed Test....133
9.10 Policy Choice....134
9.11 Inbreeding....135
9.12 Aesthetic Fitness....136
9.13 Final Population....137
9.14 Summary....137

Part III: Results....139

10 Experiment I: Game Ranking....141
10.1 Introduction....141
10.2 Aim of the Experiment....141
10.3 Method....141
10.4 Results....144
10.5 Discussion....145
10.6 Summary....146

11 Experiment II: Game Measurement....147
11.1 Introduction....147

11.2 Aim of the Experiment....147
11.3 Method....147
11.4 Results....148
11.5 Discussion....151
11.6 Summary....152

12 Experiment III: Game Synthesis....153
12.1 Introduction....153
12.2 Aim of the Experiment....153
12.3 Method....153
12.4 Results....155
12.5 Discussion....167
12.6 Summary....168

13 Conclusion....171
13.1 Introduction....171
13.2 Conclusions of the Study....171
13.3 Improvements Over Previous Work....171
13.4 Research Contributions....172
13.5 Limitations....172
13.6 Future Research....173
13.7 Summary....173

Appendices....175
A Formal Definition of the Ludi GDL....177
B Ludi Class Structure....193
C Source Games....197
D Cross-Entropy Method for Paired Comparisons....199
E Matlab Code for Ranking Paired Comparisons....201
F Top Ten Survey Games....203
G Aesthetic Value Correlations....209
H Viable Evolved Games....215
I Analysis of Yavalath....229

Bibliography....235

List of Figures

Figure 2.1 Part of the board implied by a single Trax tile....8
Figure 3.2 The essential steps of an evolutionary algorithm....22
Figure 4.1 Aesthetic scores for some simple ornaments....28
Figure 4.2 Stiny & Gips' criticism algorithm....29
Figure 4.3 Stiny & Gips' design algorithm....30
Figure 5.1 The general game model....39
Figure 5.2 Ludeme tree for Tic Tac Toe....42
Figure 5.3 Basic tiling types: triangular, square, hexagonal and truncated square (4.8.8)....43
Figure 5.4 Direct neighbours and indirect (diagonal) neighbours....43
Figure 5.5 Cell phases....44
Figure 5.6 Board shapes for triangular, square and truncated square (4.8.8) tilings....44
Figure 5.7 Predefined board shapes for hexagonal tilings....45
Figure 5.8 A boardless game in progress....45
Figure 5.9 White and Black regions around the board perimeter....46
Figure 5.10 Piece movement by turtle instructions {f, f, r, f}....48
Figure 5.11 Linecap conversion....49
Figure 5.12 A move allowed by the compound movement rule listed above....50
Figure 5.13 Typical starting positions....51
Figure 5.14 The connection end condition....53
Figure 6.1 The general game player model....59
Figure 6.2 Bridges on the trivalent hexagonal and truncated square tilings....62
Figure 6.3 Bridges on the square grid with and without diagonal adjacencies....62
Figure 6.4 Ludi user interface in tutorial mode....65
Figure 6.5 A ko situation....70
Figure 6.6 A simple game with a huge branching factor....72
Figure 7.1 The advisor model....75
Figure 7.2 The policy model....76
Figure 7.3 Distance metrics for square grids with and without diagonal adjacencies....77
Figure 7.4 Potential 4-in-a-row lines through piece a....83
Figure 7.5 Starting position and end game showing swarming....89
Figure 7.6 A case of detrimental stacking (Black to play)....90
Figure 7.7 Central cell preference....91
Figure 7.8 Board sides protect against jump capture....92
Figure 7.9 Diagonal moves influence more neighbours: 12 (left) and 10 (right)....92
Figure 7.10 Diagonal connections are more dangerous....93
Figure 8.1 The player model....95
Figure 8.2 The aesthetic model....96
Figure 8.3 Move history of a game between White and Black....99
Figure 8.4 Lead history with respect to the winning player (White)....100
Figure 8.5 An uncertain game (left) and a certain game (right)....108
Figure 8.6 A dramatic recovery by White....109
Figure 8.7 A killer move....110
Figure 8.8 Two cases of move recoveries....111
Figure 8.9 A game with four lead changes (indicated)....112
Figure 8.10 A cold move....113
Figure 8.11 Decisiveness threshold....114
Figure 8.12 A decisiveness game (left) and an indecisive one (right)....114
Figure 8.13 White carries the momentum....116
Figure 8.14 A lead increase by White and its subsequent correction....117
Figure 9.1 The game life cycle....125

Figure 9.2 Selected pair of parent games: Tic Tac Toe and Y.....128
Figure 9.3 Recombining the parents into a single Child.....129
Figure 9.4 Mutating the child.....130
Figure 9.5 A rule clash (left) and a vestigial rule (right).....131
Figure 10.1 Classification success rate over 100 epochs.....144
Figure 10.2 Final classification results over all comparisons.....145
Figure 11.1 Aesthetic criteria scores over all games.....148
Figure 11.2 Correlation of the best 17 aesthetic criteria with game scores and rankings.....150
Figure 11.3 The best 17 predictors and their correlation weightings.....150
Figure 11.4 Increase in error following the removal of each criterion.....151
Figure 12.1 Classification success rate over 100 epochs.....156
Figure 12.2 Final classification results over all comparisons.....156
Figure 12.3 Final classification results over all comparisons.....157
Figure 12.4 Starting position for Ndengrod.....158
Figure 12.5 Starting position for Yavalath.....158
Figure 12.6 Yavalath's family tree.....159
Figure 12.7 Starting position for Rhunthil.....160
Figure 12.8 Starting position for Teiglith.....161
Figure 12.9 Starting position for Elrostir.....162
Figure 12.10 Starting position for Lammothm.....162
Figure 12.11 Starting position for Gorodrui.....163
Figure 12.12 Starting position for Pelot.....163
Figure 12.13 Starting position for Hale.....163
Figure 12.14 Starting position for Quelon.....164
Figure 12.15 Starting position for Duath.....164
Figure 12.16 Starting position for Elrog.....164
Figure 12.17 Starting position for Vairilth.....165
Figure 12.18 Starting position for Ninniach.....165
Figure 12.19 Starting position for Pelagonn.....166
Figure 12.20 Starting position for Valion.....166
Figure 12.21 Starting position for Eriannon.....166
Figure 12.22 Starting position for Bregorme.....167
Figure 12.23 Starting position for Pelagund.....167
Figure G.1 Correlation of all 57 aesthetic criteria with game scores and rankings.....209
Figure G.2 Correlation weightings of all 57 aesthetic criteria.....210
Figure G.3 Correlation of the 16 intrinsic criteria with game scores and rankings.....211
Figure G.4 Correlation weightings of the 16 intrinsic criteria.....211
Figure G.5 Correlation of the 30 quality criteria with game scores and rankings.....212
Figure G.6 Correlation weightings of the 30 quality criteria.....212
Figure G.7 Correlation of the 11 viability criteria with game scores and rankings.....213
Figure G.8 Correlation weightings of the 11 viability criteria.....213
Figure G.9 Correlation of the best 17 aesthetic criteria with game scores and rankings.....214
Figure G.10 Correlation weightings of the best 17 aesthetic criteria.....214
Figure I.1 White's forcing move 1 wins the game.....229
Figure I.2 An unblocked size 4 triangle (left) allows its owner to force a win (right).....230
Figure I.3 Optimal blocking pattern.....230

List of Tables

Table 9.1 Mini-tournament for default, parent and hybrid policies..... 134
Table 11.1 Aesthetic score correlations by criteria type.....149
Table 12.1 Viability testing at different thresholds..... 154

List of Code Listings

Listing 6.1 Steps in the new game sequence.....68
Listing 6.2 Pseudocode for move message handling.....69
Listing 6.3 Actions performed for each move.....69
Listing 6.4 Computer move planning.....71
Listing 9.1 The game evolution process.....126
Listing 9.2 Pseudocode for viability testing.....136
Listing E.1 Matlab code for ranking games based on paired comparisons.....202
Listing I.1 Move selection in GUCT playouts.....231

List of Appendices

A Formal Definition of the Ludi GDL 177
B Ludi Class Structure 193
C Source Games 197
D Cross-Entropy Method for Paired Comparisons 199
E Matlab Code for Ranking Paired Comparisons 201
F Top Ten Survey Games 203
G Aesthetic Value Correlations 209
H Viable Evolved Games 215
I Analysis of Yavalath 229

Statement of Original Authorship

The work contained in this thesis has not been previously submitted for a degree or diploma in any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

…….
Cameron Browne

July 2008

Acknowledgements

I am greatly indebted to my supervisor Frederic Maire for his generous help and guidance over the course of this project, in addition to his constant faith that it would eventually be completed despite all evidence to the contrary. The interest he showed in the material, his patience with the many delays and the generosity of his time were beyond the call of duty and much appreciated, as were the occasional games of Hex when time permitted.

Also thanks to my associate supervisor Professor Binh Pham for giving me the opportunity to pursue this project and for helpful guidance, and to Ricky Tunny for assistance with negotiating the various administrative hurdles along the way.

Thanks to the members of the Gamerz.net online board game community for their interest in the project and to those generous individuals who participated in the game design surveys. Special mention goes to Stephen Tavener, Paul van Wamelen, Larry Wheeler and Richard Reilly for fruitful discussions on matters related to game design and Phil Bordelon for advice during the planning stages of the general game player.

The help of the following research students is appreciated: Sam Henwood, Frank Loewenich and Audun Ellertsen.

Thanks to my wife Helen Gilbert for her continued support and the gentle but frequent reminders that I still had a thesis to complete despite life's many distractions.

Finally, a tip for new players: if you think that a PhD might be an amusing thing to do in your spare time and shouldn't take *that* long to knock off... think again.

Chapter 1

Introduction

1.1 Overview

This chapter outlines the background to the research and states the research problem and hypotheses. A justification for the research is presented, followed by a description of the methodology used, its scope and key assumptions.

1.2 Background to the Research

Many new board games are designed each year, ranging from the unplayable to the truly exceptional. For each successful design there are untold numbers of failures; game design is something of an art. Players generally agree on some basic properties that indicate the quality and viability of a game, however these properties have remained subjective and open to interpretation.

This thesis investigates notions of quality in game design using combinatorial games as a test-bed. Combinatorial games are ideal for this purpose as they are typically strategically deep yet described by simple well-defined rule sets.

1.3 Research Problem and Hypotheses

The research problem is twofold. The first question to be addressed is whether there exist aesthetic criteria for combinatorial game design that may indicate game quality through automated self-play. Secondly, the practicality of using such aesthetic criteria to direct the automated search for new games of high quality is investigated.

Two hypotheses are proposed:

Hypothesis I

That there exist fundamental (and measurable) indicators of quality for combinatorial game design.

Hypothesis II

That these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games.

Game quality, in this context, is defined as the likelihood that a given game will be of interest to human players.

1.4 Justification for the Research

Automated analysis tools have proven useful for assisting game designers in evaluating games and speeding up the game design process [Althöfer, 2003]. However, the principles by which games are

evaluated are not precisely defined; while most designers agree on some fundamental principles, such as depth and clarity, even the definitions of these terms are open to interpretation and debate. No previous study has attempted to define or implement a complete aesthetic system for combinatorial games.

Representing problems algorithmically requires an explicit awareness of underlying assumptions and details that may remain hidden using less rigorous methods [Stiny & Gips, 1978]. Simply recognising, defining and implementing the aesthetic criteria for game design quality may yield some deeper insight into them.

Although there has been a significant amount of research into combinatorial games, most attention has been paid to the quality of play rather than the quality of design. Combinatorial game theory is primarily concerned with solving games, or at least analysing them to determine optimal strategies and develop computer players able to challenge human experts. Within the context of this thesis, however, the computer player is of little interest except as a means for providing useful self-play measurements.

There exist some automated systems for assisting designers in the creation of new games using rule randomisation, typically through a constructive mode of creation based on well-defined generative rules [Pell, 1993b]. The creation of new games through the evolution of rule sets, and their subsequent automatic measurement for quality, has not been previously attempted. An advantage of the evolutionary approach is that rule constraints may be relaxed during the evolutionary process to encourage the emergence of unforeseen and interesting rule combinations. Such emergent rules may constitute fully formed games or at least suggest new avenues for game designers to explore.

Lastly, there is the sheer research interest of whether a machine can succeed at the very human creative skill of designing games. Pell [1992] states:

> "If we could develop a program which, upon consideration of a particular game, declared the game to be uninteresting, this would seem to be a true sign of intelligence! So when this becomes an issue, we will know that the field has certainly matured."

1.5 Methodology

To test these ideas, a game description language has been devised to express a general class of combinatorial games, and a general game system called Ludi implemented to play, measure and explore them. Key features of the system include modules for measuring statistical aspects of self-play and synthesising new games through the evolution of existing rule sets.

The results are validated experimentally: can automated aesthetic measurements be correlated with human player preferences and used to successfully guide the search for new high quality games? The success of the hypotheses is gauged by these findings.

1.6 Outline of the Thesis

This thesis is divided into three main parts.

Part I: Background Theory

Part I presents background material relevant to the research problem, specifically:

- A definition of combinatorial games and their key elements,
- Search methods relevant to the problem, and
- Aesthetics and game design.

Part II: Ludi Framework

Part II describes the design and implementation of the Ludi general game system, which constitutes the main contribution of this thesis. The Ludi system features a complete aesthetic model for measuring combinatorial game quality and a complete synthesis process for the creation of new games within this model. Key components include:

- The Ludi game description language,
- The Ludi general game player,
- Strategy module for directing play,
- Criticism module for measuring game quality, and
- Synthesis module for creating new games.

Part III: Results

Part III presents the results of three experiments conducted to test the hypotheses. These include:

- A survey to determine human player preference scores for a set of existing games,
- The correlation of these human player preference scores with aesthetic measurements made during self-play trials, and
- The evolution of new games based on these aesthetic measurements.

1.7 Scope and Key Assumptions

This section describes the scope of the research and relevant key assumptions.

Scope of the Games

This thesis concentrates on two-player combinatorial games, as defined in Chapter 2. To be precise, the Ludi general game system implements a subset of games defined by the game description language, which constitute a subset of combinatorial games, which constitute a subset of all games. Emergent behaviours and rule space complexity ensure that this subset of games is more than sufficient for the purposes of this study.

Scope of the Aesthetic Measurements

The quality of a game is deemed to equate with its intellectual interest to human players. Aesthetic criteria are therefore limited to abstract concerns:

- Do the rules work harmoniously?
- Does the game provide a meaningful contest between its players?
- Do the rules contain flaws that may be exploited?

This precludes any physical concerns such as a game's visual presentation, construction quality, relevance of theme, and so on. The Ludi general game player therefore presents games as generically and anonymously as possible.

Social aspects such as inter-player communication and dynamics are not included, except for the modelling of pathological play styles (obstructionist, defensive, random, and so on) during self-play testing.

Design flaws often become apparent during the game evaluation process. However, this thesis concentrates on the automatic detection of flaws rather than their automatic correction.

Scope of the Computer Player

The success of self-play trials between computer opponents relies on the assumption that the resulting games will display similar trends to those found in games between human opponents. The computer player must therefore perform at a competent level for every game, if each is to be explored and measured in a meaningful way. There is no need to produce expert computer players for every game and no attempt is made to do so; all players, human or computer, will be novices when it comes to the newly evolved games.

Unlike most previous combinatorial game research, this thesis concentrates on quality in game design rather than quality in move planning.

1.8 Contributions

The project provides two main research contributions:

- An aesthetic model for measuring quality in combinatorial game design, and
- A process for the automated creation of new high quality games.

1.9 Summary

This thesis investigates concepts of aesthetics in combinatorial game design, specifically whether such concepts may allow quality measurement through self-play and the automated search for new high quality games.

A general game system called Ludi is implemented to explore these questions, and a series of experiments performed to validate the results.

Part I

Background Theory

Human beings are never more ingenious than in the invention of games.
– Leibniz

Overview

Part I presents background material relevant to the research problem.

Chapter 2 defines the class of combinatorial games to be studied in this thesis, and how these games may be decomposed into their constituent rules.

Chapter 3 describes three methods of search relevant to this project:

- Adversarial search for move planning in general games,
- Competitive search for optimal board evaluation policies, and
- Evolutionary search for new and interesting rule combinations.

Chapter 4 discusses the relevance of aesthetic theory to those aspects of game design generally understood to indicate a game's quality.

Chapter 2

Games in General

> The board is set, and the pieces are moving... the Enemy has the move, and he is about to open his full game.
> – J. R. R. Tolkien, *The Return of the King*
>
> You've spent all your life learning games; there can't be a rule, move, concept or idea… that you haven't encountered ten times before in other games.
> – Iain Banks, *The Player of Games*

2.1 Introduction

This chapter defines the class of games studied in the thesis and examines the elements that make up a game. Systems for working with general classes of games are then introduced, along with their associated game description languages. Games are decomposed into their constituent ludemes (game elements) and techniques for creating new games discussed.

2.2 Definitions

There are many ways to define a game. At its simplest, a game can be described as an exercise with rules that involves overt competition [Ellington et al, 1982] or simply organised play [Huizinga, 1955].

The most useful definition is probably that given by Salen & Zimmerman [2004]: *A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome*. This definition was made after studying the essential elements of many existing game definitions over a broad range of prior studies, most of which shared the following common points:

- Rules,
- Play (conflict), and
- Outcome.

The concept of "play" was common to all of the prior definitions. It is useful to distinguish between strategic and tactical play in this context.

Strategy is the art of war and concerns the overall management of forces in conflict with an enemy. The ability to conceptually decompose play into worthwhile subgoals, such as in Chess, is an indicator of strategy.

Tactics involve the low-level deployment of those forces to achieve specific goals. Larry Levy describes strategy as long-term thought and tactics as those decisions that change with every game turn [2002]. Levy personally prefers the challenges thrown up by each turn of a tactical game, rather than having to study a game as a whole to come up with a perfect strategy.

Tactics therefore equate with short term (local) combinatorial play and strategy with long term (global) planning. The subgoal of removing the opponent's defensive pieces in order to attack a key piece is an example of a strategy, while making a move that constitutes a *fork* (two or more

simultaneous but non-overlapping threats [Epstein, 1991]) in order to actually remove one of these defensive pieces is an example of tactical play.

Parlett [1999] makes the distinction between *informal games* with undirected play, such as that in a school playground, and *formal games* with strictly directed play. Formal games have:

- *Means*: An agreed set of equipment and rules.
- *Ends:* The game is a contest to achieve some goal.

Board Games

The term *board game* can define any game that can be played on a flat surface [Parlett, 1999] and is generally understood to imply the presence of a physical board. De Voogt further states that "a board game differs from a game in that the board should play a crucial role" [1995].

Many games, particularly tile-laying games, feature an implied rather than physical board with respect to the first piece played. For instance, Figure 2.1 shows the board implied by the placement of the first tile in the game of Trax; all subsequent tiles must be placed adjacent to at least one existing tile, forming an implied square grid. Furthermore, Trax is limited to an 8x8 area which imparts physical constraints on this imaginary board.

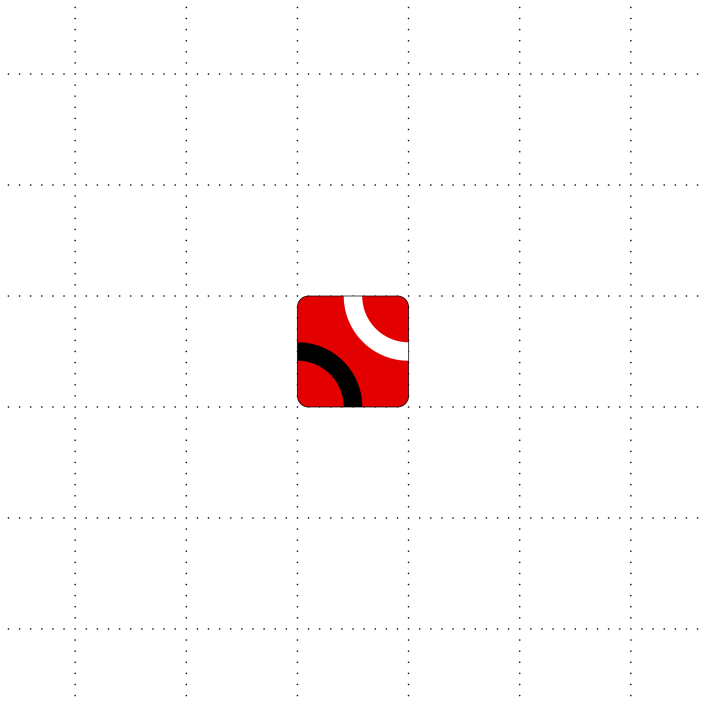


Figure 2.1 Part of the board implied by a single Trax tile.

This concept of a boardless board game is somewhat paradoxical. Furthermore, game boards need not be flat, and an increasing number of board games are incorporating interesting three-dimensional elements.

Abstract Games

A game may be described as *abstract* if it is not a simulation and may be easily decoupled from any underlying theme without affecting play.

This is not a very useful definition as the degree to which a particular game is *themed* is often hotly debated. In addition, the very definition appears to be rather transient; the phrases *abstract games*, *abstract board games*, *abstract strategy games* and even just *abstracts* are used by many players to refer to the same type of game, and each phrase may have significantly different implications for players depending upon their background.

Describing an abstract game as an abstract strategy game generally implies that it is a game of *pure strategy*, that is, luck does not play any part and play is determined exclusively by the players' decisions.

Combinatorial Games

For these reasons, the term *combinatorial game* will be used instead to describe the type of game studied in this thesis. A combinatorial game will be understood to be:

- *Discrete*: Players alternate making moves (no simultaneous moves).
- *Finite*: Produces a well-defined outcome after a finite number of moves.
- *Deterministic*: Chance plays no part.
- *Perfect information*: No hidden information.

Two players will be assumed unless otherwise stated, although this is not a requirement; applications of the ideas raised in this thesis will be discussed in terms of *N*-player games where appropriate. From this point on, the term *game* will refer to a two-person combinatorial game unless otherwise stated.

The term *combinatorial* has in fact a two-fold meaning in this context, as it describes not only a game's combinatorial style of play but also the way in which rule sets themselves may be recombined to create new games.

Less Than Two Players

Games with one player constitute *solitaire puzzles*. Just as moves in two-player games may be described as puzzles that the players set each other [Thompson, 2000], solitaire puzzles may be viewed as combinatorial games played between the player and the inventor; the player wins by achieving the required goal before running out of moves, else the inventor wins.

Salen & Zimmerman [2004] point out that all games involve a conflict, whether it occurs directly between the players of the game or between the player(s) and the challenging activity presented by the game itself.

More Than Two Players

Multiplayer games are those with more than two players. Multiplayer games are distinguished from two-player games due to the coalitions that may arise during play. These coalitions may be tacit and not communicated in any way except through the movements of pieces on the board, but can lead to some fascinating play dynamics.

For example, three-player games are subject to the problem of *petty diplomacy* [Schmittberger, 1992] in which the two losing players at any point are obliged to form micro alliances against the leading player (tall poppy syndrome). Such alliances are described as "micro" as they are unspoken and fleeting; allies one move will become enemies the next if they succeed in their goal of reigning back the leading player.

It is arguable, however, whether this is a problem in all cases. In some games it provides an elegant balancing mechanism and can lead to complex and rewarding passages of play. Conversely, some

games contain mechanics with which a player may deliberately pit their two opponents against each other [Neto, 2005].

Another dynamic to emerge from multiplayer games is the *kingmaker effect* [Kramer, 2000], which is the ability for a player with no hope of winning to decide the eventual winner by their moves. Reilly [2008] points out that this ability to dictate the winner may motivate a losing player to keep playing, however this means that the best player may be eliminated through no fault of their own and is generally deemed to be detrimental to a game.

Combinatorial Game Theory

Combinatorial game theory (CGT) is the study of two-player combinatorial games [1991]. This theory is founded in two famous works: Conway's *On Numbers and Games* [1976] and Berlekamp et al's *Winning Ways for your Mathematical Plays* [1982].

CGT is concerned primarily with the mathematical analyses of games, with a view to forming optimal strategies and solving games where possible. One of the key concepts of CGT is that of *temperature*, which indicates the urgency of making moves in a game. Essentially, if a game is hot then the player may choose between positive (beneficial) moves but if the game is cold then the player must choose between negative (harmful) moves.

The main emphasis of much CGT research is on finding algorithms to outperform human experts. Allis [1994] outlines the progress made in achieving this goal for a number of the more popular games including Checkers, Chess, Go, and so on. In answer to the question "which games will survive?" Allis suggests that "*all* games will survive at *all* levels," even if this takes the form of competition between the programmers of increasingly expert players.

The contribution of this thesis, on the other hand, lies in a significantly different direction. It concentrates instead on the general question of what makes a game interesting and the standard of the game itself rather than the standard of the player. Although necessarily framed within the CGT paradigm, this thesis does not strictly follow the standard CGT approaches, and uses a player-centric rather than mathematical model of games.

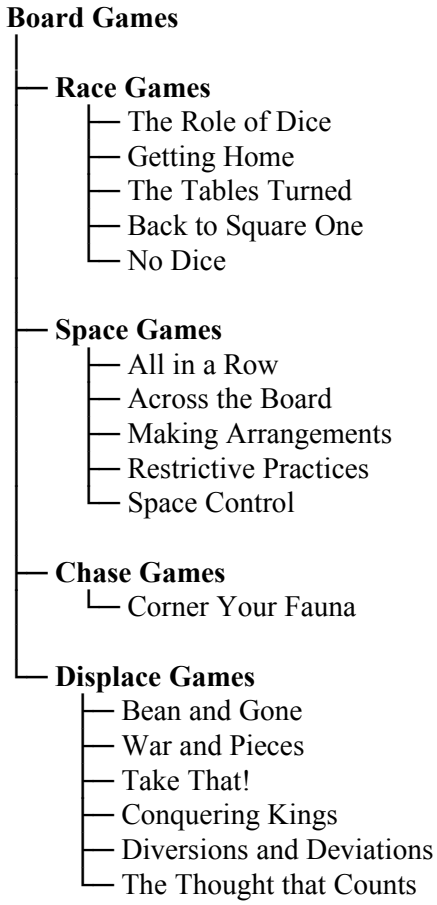
For this reason, custom software was developed for the current project rather than building on the software tools provided by previous combinatorial game researchers, notably the Gamesman's Toolkit [Wolfe, 1994] and the Combinatorial Game Suite [2004].

2.3 Types of Combinatorial Games

Game historian David Parlett [1999] proposed the most comprehensive classification of board games to date, the main categories of which are listed below. Although devised for board games, this classification is also useful for combinatorial games.

Race games involve moving pieces along linear tracks between starting and finishing points. Interaction between opposing players generally consists of leapfrogging opponent's pieces or ousting them back to their starting positions.

Space games involve arranging pieces into patterns or to satisfy some geometric requirement, such as connecting target areas with a path of pieces or enclosing the most territory. Interaction involves players placing pieces to disrupt the spatial relationships between opponent's pieces while developing their own.



Chase games are those in which a player wins by cornering a key enemy piece to capture or entrap it. These games typically involve unbalanced forces in which one player takes on the role of hunter and their opponent the prey, although this is not always the case.

Displace games are those in which players must capture a certain number of enemy pieces or enemy pieces of specific types. This category is further divided into War Games in which players use differentiated pieces of their own colour and Mancala Games in which players share a common set of undifferentiated pieces.

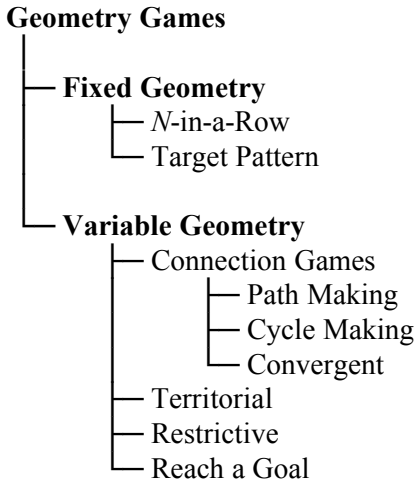
Although both Parlett's classification and the combinatorial games discussed in this thesis are limited to two-dimensional games, this scheme is also generally valid for three-dimensional games.

Geometry Games

Since the publication of Parlett's classification scheme, it has come to light that the class of games described as Space Games may in fact constitute a larger and more important category than was previously realised [Browne, 2005].

An alternative scheme for classifying such games is to distinguish between games of *fixed* and *variable geometry*, as suggested by Ralf Gering [2005]. The subcategories are similar to those proposed by Parlett but their organisation is different.

Games of fixed geometry involve the creation of pre-defined patterns, such as lines of N pieces or particular shapes, whereas games of variable geometry involve geometric goals that may be achieved in a number of ways. Games of variable geometry may be described as *topologically invariant* [Handscomb, 2000] as it is the existence of a geometric property, not its form, that is the goal.



Connection games, in particular, are an important subclass of considerable interest. A connection game is one in which players strive to complete a specific type of connection with their pieces, which might involve forming a path between two or more target regions, forming a closed loop, or gathering all pieces together to form a single connected group [Weisstein, 2005].

2.4 General Games

The methods used to define, play and analyse a particular game may have wider applicability to the general class to which that particular game belongs. The main emphasis of this project is the study of such methods for general games, including the creation of new games within the given class.

2.4.1 General Game Players

A *general game player* is, as the name suggests, a computer programme for playing a general class of games rather than any specific game. Such players are also typically used to assist with the analysis of games and the authoring of new ones, and may be called *general game systems*. Such systems were proposed several decades ago [Pitrat, 1968] but have not received a great deal of attention until recent years.

The term *metagame system* is sometimes used to describe such systems, in the sense that a *metagame* is a strategy that transcends the rules of a particular game and has general applicability to a wider class of games. However, this term is somewhat overloaded as metagaming is also a development of game theory related to optimal decision making in a given scenario [Howard, 1971].

General game systems of note include the Smart Game Board [Kierulf, 1990] which was an extensible game playing program initially designed for Go, but expanded to include Othello, Chess and Nine Men's Morris. The Smart Game Board featured an extensive user interface for stepping through the game tree and board positions. Moves were evaluated using a large number of hand-crafted evaluation functions called *move motives*; new functions had to be added for new games. Müller [1991] describes a collection of software modules for the Smart Game Board that can be customised to implement new games, which can be useful for prototyping games and investigating move trees. One of the motivations behind the Smart Game Board was to allow users to concentrate on new ideas rather than rewriting existing code.

Pell's METAGAMER [1993], a general system for Chess-like games, was able to automatically fine-tune its advisors (evaluation functions) through self-play to optimise play for a given game. New Chess-like games are created by METAGAMER according to probabilities and constraints specified

by the user, three of which are discussed in detail in [Pell, 1996]. Cron's [1999] Masters thesis further investigates search in metagames.

Garcia's GAMESMAN [1995] is a software tool for quickly prototyping new games and playing them against the computer. The computer player plans moves based on a full game tree expansion if the game is simple enough to be solved, otherwise the user must provide a customised move evaluator for each game.

Awale [Guillion & Guillion, 1996] is a general game system for Mancala-style (seed sowing) race games.

Zillions of Games [Mallett & Lefler, 1998] is a commercially available general game player that plays a wide range of games at a reasonable level. The programme automatically tunes its evaluation functions based on the rules of a game using features such as piece mobility, board topology and end conditions. Zillions of Games is no longer under development.

Orwant [2000] describes the Extensible Graphical Game Generator (EGGG) which allows users to create variants of games with minimal programming effort. It provides a generic evaluation routine based on standard minimax search and generic board features including piece power, piece mobility, and distance to goal. Orwant points out that the EGGG static board evaluator performs badly for some games, but in most cases will be better than nothing.

Morphling is a system for designing and playing a class of two-person combinatorial games [Althöfer, 2003; Rolle, 2003]. This system aids the designer by performing measurements that indicate the interestingness of a given rule set, based on a small set of criteria.

Gtkboard [2003] is a public domain software tool for prototyping new games. New games are fitted as plug-ins, and a generic search algorithm used for move planning. Gtkboard is no longer supported.

2.4.2 Game Description Languages

A *game description language* (GDL) is a well-formed grammar that describes all aspects of a game required for a particular system to play it. GDLs are therefore designed with computer interpretation in mind, and the basic principles of good software design outlined by Kernighan & Pike [1999] generally apply:

- Simplicity,
- Clarity,
- Generality, and
- Automation.

A GDL should be *simple* so that complex rule combinations can be described in a non-confusing way. The simpler the language, the easier it is to interpret and manipulate, and the more robust it will be to rule clashes and other errors.

A GDL should be *clear* so that the concepts embodied by the rules and their relationship to each other can be readily understood; ideally a human player unfamiliar with the language should be able to read a description of the game phrased in the GDL and understand how to play it. The behaviour and implications of component rules should also be clear to a game designer using the language.

A GDL should be as *general* as possible to allow the widest variety of games within its scope. The creativity of the end user should not be constrained by the design if possible.

A GDL should support *automation*. The language should be suitable for machine parsing and robust to automated rule manipulation.

In addition, a GDL should ideally be *extensible* so that new behaviours and properties can be easily added without restructuring.

The GDL provides the means with which users of a general game system may customise games and author their own games.

Examples

Pell describes a formal grammar for symmetric Chess-like games specifically designed for use with his METAGAME programme [1993a; 1993b].

The Zillions Rules File (ZRF) language was defined for use with the Zillions of Games player [Mallett & Lefler, 1998]. This language includes a number of game-specific functions in a LISP-like syntax, allowing game authors to define arbitrarily complex rule combinations. The ZRF language is powerful and allows the easy creation of a wide variety of games within its scope; games outside its scope may also be defined but getting such games to work can become something of a game in itself. Despite no longer being supported, Zillions of Games has a dedicated user base who have contributed a large number of customised games to the Zillions community.

The Axiom library [Schmidt, 2007] is plug-in for the Zillions of Games engine that provides a complete Forth-based scripting language in which games may be defined, expanding the flexibility of the system.

Genesereth & Love [2004] describe a general GDL for multi-player games of complete information, using a logic based language much like Prolog.

Love et al [2006] use the term "game description language" to describe the particular game description language devised for their own general game player. The state of the game is described as a set of facts, and the transition function between states (movement rules) described as logical rules.

Other GDLs include Hollisi's XGF [2002] and SGF [2006].

2.4.3 Ludemes

Just as a *meme* is a unit of information that replicates from one person to another [Dawkins, 1976], the elements that make up a game may be described as game memes or *ludemes*. The first use of the term is uncertain, but is most likely that by Alain Borvo in an essay describing a card game with particular rules [1977].

Parlett defines a ludeme as "an element of play, comparable to, but distinct from, a game component or instrument of play" [2007]. The distinction between an instrument of play and an element of play is that between the Chess knight and the rules describing its skewed form of movement, or that between a game board and the fact that it has five cells per side. Parlett goes on to point out that a characteristic property of ludemes is their propensity to propagate by passing not only from one game to another but even between games of entirely different classes.

Although memes and ludemes are typically described as discrete units of information, this does not imply that these units are atomic and cannot be further decomposed. For instance, the following ludeme describes a board shape:

(*shape square*)

This *atomic* ludeme may form part of a *composite* ludeme that embodies a more complex unit of information describing an entire board:

(*board*
    (*tiling square i-nbors*)
    (*shape square*)
    (*size* 3 3)
)

This unit of board information may then become part of an even more complex ludeme that describes an entire game, hence a game's rule set may be represented as a structured *ludeme tree.*

The concept of an entire game as unit of information may be surprising, but is valid; there are many examples of identical games being discovered, fully formed, by independent inventors at around the same time. The most famous example of this is probably the invention of Hex by Piet Hein in 1942 and its reinvention by John Nash shortly afterwards in 1948 [Browne, 2000].

Combinatorial games are typically complex yet described by extremely simple rule sets, making the encapsulation of entire games in a single ludeme feasible.

2.4.4 Variants

Metarules

In the context of combinatorial games, a *metarule* is rule that has wider applicability to a more general class of rules. Metarules are similar to the *mutators* described by Neto [2000].

Examples of metarules include *misere*, which inverts the winning conditions of a game, or an *either-colour* metarule that specifies that players may place pieces of either colour each turn. Metarules specify a fundamental and consistent way in which a game's rules may be modified, and provide a powerful mechanism for the creation of game variants.

Recombination Games

Another way to modify games is to borrow rules from other games and incorporate them to create new rule combinations; such games may be called *recombination games*. Schmittberger [1992] provides many elegant examples of ways in which the shelf life of familiar games may be extended by modifying their rules to create interesting variants. Similarly, Joris [2002] provides an interesting mixture of known games, variants and new games.

Salen & Zimmerman point out that: "through rule-breaking the space of possibility fills with alternative modes of play" [2004]. However, great care must be taken in the manipulation of rule sets as most games are carefully crafted by their designers and optimised with great care; rule sets are generally fragile to the slightest change.

For example, if the board size in Tic Tac Toe were increased to 4x4 then the first player would always have a trivial win by moving in one of the four central cells, and if the board size were reduced to 2x2 then no player could possibly win. Even the smallest change in board size – generally the safest way to create a variant – breaks this rule set.

Game Distance

It may be unclear at what point a modified rule set constitutes a new game rather than just a variant. This distinction may be quantified by assigning a value to each component rule based on its relative importance, for example, rules related to the winning condition will generally be more important than other types of rules.

The total weighted difference between two rule sets will give an indication of the distance between the two games. If this distance is zero then the games are identical, otherwise the greater the distance the more likely it is that the two games are distinct. Some threshold value may be specified to determine at what point a variant becomes a distinct game.

The originality of a given game's rule set may be estimated by its distance from the rule sets of known games.

2.5 Summary

The games studied in this thesis are combinatorial games, which are defined as discrete, finite, deterministic, perfect information games. There already exists a substantial field of study for two-player combinatorial games (CGT), however a different approach is taken in this thesis based upon a player-centric rather than mathematical model of play.

General game systems and their associated game description languages provide a platform for playing and studying general classes of games. Breaking a game down into its component ludemes (chunks of rule information) is a convenient way to represent games for such systems, and for recombining rule sets to create variants and new games.

Chapter 3

Search Methods

> The road to wisdom? Well, it's plain and simple to express:
> Err and err and err again but less and less and less.
> – Piet Hein, *Grooks*
>
> In the long history of humankind (and animal kind, too) those who learned to collaborate and improvise most effectively have prevailed.
> – Charles Darwin

3.1 Introduction

This project involves three main applications of search:

- The search for optimal moves,
- The search for optimal policies, and
- The search for new games.

The search for optimal moves – that is, move planning for computer players – may be achieved using standard adversarial search techniques. The latter two searches, however, are more suitable to evolutionary approaches, as they benefit from the unexpected and innovative solutions thrown up by such methods. The importance of emergence in this context is emphasised.

3.2 Move Planning

Move planning in computer players has received considerable attention from artificial intelligence researchers for many years, in particular for the game of Chess, for which standard techniques have long been in place. With slight modification, similar techniques can be applied to general game players.

The task is complicated by the fact that a general game player must formulate meaningful plays for previously unseen rule combinations. However, the task is simplified in this study by relaxing expectations of the standard of play; it is preferable for the system to play a large range of games at a reasonable level rather than a small number of games at an expert level.

3.2.1 Adversarial Search

Adversarial search problems arise from competitive environments in which agents' goals are in conflict, that is, games [Russell & Norvig, 2003]. Such problems are defined in this context by:

- *Initial state*: Describes the initial board position and player to move.
- *Successor function*: Returns a list of legal moves and their resulting states.
- *Terminal test:* Determines when the game has ended.
- *Utility function*: Assigns a value (outcome) to each terminal node.

Minimax Algorithm

The initial state and the legal moves for each game define the *game tree*. Each node in the game tree describes a possible *board state* or *board position* within the game.

For a game with a fully expanded game tree the *minimax algorithm* makes the optimal move each turn by recursively playing the move that maximises their outcome and minimises the opponent's outcome.

Alpha-Beta Pruning

Alpha-beta pruning is the standard method for pruning branches of the game that cannot possibly influence the final decision of the minimax algorithm. Alpha-beta pruning has been studied extensively elsewhere [Russell & Norvig, 2003].

Even with alpha-beta pruning, however, complete game tree expansion is infeasible for all but the most trivial games. Shannon [1950] suggested a method for dramatically improving the speed of such searches by cutting off the search at a specified depth and returning an evaluation of the board position at that point if it is not a terminal (winning) node.

The depth at which the search is cut off is described as the *ply* of the search. Evaluation functions for estimating the worth of non-terminal nodes are discussed further in Section 3.2.2.

Iterative Deepening

Ideally, we would like a computer player to choose its moves within a reasonable amount of time, both for the comfort of human opponents and to allow sufficient numbers of self-play games for automated learning purposes. It is difficult to specify a single search depth that will process the greatest number of moves within this time limit, as games may have significantly different *branching factors* (number of moves per turn) resulting in game trees of varying complexity.

Instead, the optimal search depth for a given game within a given time limit may be found by a process of *iterative deepening* in which a search is made at depth 1... then depth 2... then depth 3... and so on, until an outcome is guaranteed or the time limit is reached. This provides a method for making the best use of the available time and allows the search to return sensible results (from the last completed ply) if interrupted.

Russell & Norvig note that "in general, iterative deepening is the preferred uninformed search method when there is a large search space and the depth of the solution is not known" [2003]. In practice, iterative deepening to a maximum depth D is not that much slower than a fixed search to depth D as most of the search time is spent in the deeper reaches of the game tree in both cases, and the smaller plies are generally trivial in size by comparison.

Move Ordering

The speed of the alpha-beta search can be improved by *move ordering* [Levy, 1983]. This involves sorting the candidate moves at each point by value and exploring them in order; the optimal move is then more likely to occur earlier in the search, resulting in more pruning.

Move ordering is typically achieved using a 1-ply search on the candidate move set. Such evaluations are described as *shallow evaluations*, as opposed to the *deep evaluations* performed by deeper searches $N>1$ on the same set of moves. Shallow evaluation may be equated with tactical play and deep evaluation with strategic play, although this analogy is somewhat tenuous as strategic play usually assumes some degree of structured planning towards achieving particular goals.

Beam Search

Another obvious way to speed up the search is to reduce its width to only follow *plausible* moves at each turn [Van den Herik, 1983], constituting a *beam search* that considers only the *N* most highly valued moves at each node in the game tree. This has the benefit of imposing a strict limit on branching factor within the tree, reducing search complexity, at the expense of possibly ignoring good moves; meaningful move ordering then becomes critical.

Grimbergen [2000] describes *plausible move generators* (PMG-*N*) that rank moves and only search on the *N* best at each node. This was found to yield significant savings of 46-68% on search time while only excluding 1-7% of the moves that would have been chosen by expert players. Grimbergen & Matsubara [2001] later refined this technique to produce 33.2% savings on search time on average, while producing strong Shogi players that still played 99.5% of moves that human experts would have played.

Many other techniques for speeding up and otherwise improving adversarial search have been proposed. The standard techniques described above have proved sufficient for the current project in which the emphasis is on producing a competent general player rather than a specialised expert one. However, special attention must be paid to the evaluation functions used by the search if it is to succeed for general game play; these are now discussed.

3.2.2 Evaluation Functions

The evaluation must return a reliable estimate of the strength of a given board position relative to the specified player, which it typically achieved by measuring particular *features* of the board position. Levy describes the following stages for building a useful evaluation function for a complex game [1983]:

- Identify the important features,
- Quantify the features, and
- Weight the features.

This process can be difficult for the case of general games which may consist of novel rule combinations upon which no prior analysis has been made and for which no strategic or tactical knowledge exists; indeed, nothing is known about such games apart from their rules.

Advisors

In the context of general game players, *advisors* are evaluation functions that represent some narrow but rational view of the board position [Epstein, 1991]. Each advisor epitomises a different perspective on play, and expresses to the player whether the given board position is favourable or unfavourable to them within this perspective through a numerical value (*advice*) [Pell, 1993b]. Advisors play the part of features in the overall evaluation function.

Pell distinguishes between local and global advisors [1993a]:

- *Local:* Specific to particular pieces or relationships between sets of pieces.
- *Global:* Relevant to the overall board position.

Pell also distinguishes between static and dynamic advisors:

- *Static:* Based on tables pre-calculated once when the game is first loaded (fast).
- *Dynamic:* Recalculated for each given board position (slow).

It is necessary to identify a set of advisors that will realistically provide useful information for any general game described by a given game description language (GDL). Some universal constants of

game play can be relied upon as being relevant to many games: material count, piece mobility, attacking potential, and so on. Additional advisors may be crafted relative to the winning conditions described in the GDL; these will be the yardsticks by which the adversarial search will judge any move.

Once the relevant advisors have been identified, they must be quantified (implemented) according to the specific application.

The weighting of the advisors is now discussed.

Policies

Evaluation functions typically combine features using a weighted linear function [Russell & Norvig, 2002]:

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

where $f_i(s)$ is a function that returns an evaluation of board state s with respect to feature i, w is a vector of weights, and $Eval(s)$ is the overall evaluation of board state s.

The weight vector w constitutes a *policy* that describes the relative importance of each feature (advisor) for a particular game. The *optimal policy* for a game is the weight vector that results in the greatest number of wins for that game; different games will in general have different optimal policies.

A game's policy may be seen as representing tactical knowledge of the game, as it informs the player's immediate moves. The adversarial search may be seen as representing the strategic aspects of play, as the values returned from deeper in the tree represent future goals towards which the player moves.

Policies may change over the course of a game. For instance, a game may reward positional development in its early stages but defensive cohesion in its later stages. Higashiuchi & Grimbergen [2005] show how savings in search efficiency can be made in the game of Amazons by dynamically adjusting move ranking estimates according to the progress of the game.

Böhm et al [2005] suggest the following enhancements to linear weighting for policies:

$$R_e(s) = \sum_{i=1}^{n} w_i f_i(s)^{e_i}$$

$$R_d(s) = \sum_{i=1}^{n} w_i \left| f_i(s) - d_i \right|^{e_i}$$

where $R_e(s)$ is an exponential rating function that includes an exponent for each feature, and $R_d(s)$ is an exponential rating with displacement so that values need not have their origin at zero.

3.2.3 Noise

It is likely that two computer players, using the same starting conditions and move planning algorithm, will follow similar lines of play game after game. It can be desirable to nudge such players out of this rut in order to explore the game's move space more fully.

For this purpose, small random disturbances applied to board position evaluations can be used to slightly rearrange the move orderings. Sadikov et al [2005] demonstrated that increasing the search depth does not actually filter out the noise but does help to restore correct move ordering according to strength. Further, Althöfer & Heuser [2005] found that applying random disturbances to board position evaluations considerably improved the playing strength in some cases of single-agent search.

Novice Players

Given that general game players will be required to play arbitrary games with novel rule combinations, the system may be considered a novice at such games just as much as any human opponent.

Májek & Iida [2004] model novice players by *forgetting* some of the possible moves during move planning. The moves to be forgotten are chosen at random and represent the tendency for the novice player to fail to recognise and explore some lines of play that an experienced player would follow. The forgotten moves are stored in a hash table so that they do not crop up again in different parts of the game tree visited by the same search. Markovitch & Scott [1988] demonstrate that random forgetting can sometimes lead to substantial improvements in performance in machine learning tasks.

In a sense, using beam width to narrow the search width achieves a similar purpose in a more brutal way; it simply eliminates the less desirable moves each turn from the player's consideration. It may be argued that this is not a good model for novice play as only the highest-ranking N moves are considered, however the fact that these moves are ordered by shallow lookahead (such as might be performed by a novice player) gives this observation some credence. The move choice may be randomised further by applying a *stochastic beam search* which chooses N moves from distribution based on shallow evaluation rather than necessarily the N best moves themselves.

A good reason for modelling novice players is that if a new game is not enjoyable among novices then it cannot develop into a popular game [Májek & Iida, 2004].

3.2.4 Upper Confidence Bounds for Trees

The Upper Confidence Bounds for Trees (UCT) method [Kocsis & Szepesvári, 2006] is an extension of Monte Carlo methods for search planning based on the multi-armed bandit UCB1 proposed by Auer et al [2002]. UCT has recently been applied with considerable success to the demanding task of move planning in computer Go [Gelly et al, 2006] with the development of the MoGo player. UCT has also been applied to Hex and Y, two other games for which move planning is difficult [Raiko, 2006].

UCT gains knowledge of a game through large numbers of random playouts. The reason for its success lies in the innovative method by which UCB1 balances *exploration* for new knowledge with *exploitation* of existing knowledge during the search. For each choice, untried options are tried first then the option that maximises $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$ is tried, where $\bar{x}_j$ is the average reward obtained from choice j, n_j is the number of times choice j has been tried so far, and n is the overall number of choices made so far.

The performance of the UCT algorithm for a given game can be significantly improved by incorporating domain knowledge relevant to that game. This is achieved by improving random playouts to avoid plays that are known to be disastrous and prefer plays that are known to be strong, typically based on local piece patterns [Gelly et al, 2006].

An attractive property of UCT is that it can perform at a high level even in the absence of such domain knowledge. For example, the Mambo UCT player [Browne, 2007] can beat most human players with purely random playouts and a search time of 10 seconds per move, at approximately 2,000 random playouts per second.

For a given game, the UCT algorithm only requires the following information to plan reasonable moves:

- The legal moves for any given board position, and
- Whether a given board position ends the game.

This ability to plan moves in the absence of any tactical or strategic knowledge about a game – in fact, the absence of almost any knowledge about the game at all – makes UCT an attractive proposition for general game players, which should ideally be able to formulate reasonable moves for previously unseen rule sets. This avoids the need to define and implement advisors and determine policies specific to each game.

Another attractive feature of the UCT algorithm is that it adjusts its search depth asymmetrically as required; more promising lines of play are followed more often as their worth becomes more certain.

Unfortunately, speed is a limiting factor. Gelly [2006] reports that MoGo generates around 400,000 nodes per move (at around 12,000 nodes per second) to compete with the world's best Go programmes in tournament play. Given the Mambo UCT player's success with only 2,000 random playouts per second, this suggests a ball-park figure for a game to be suitable for UCT; as a general rule of thumb, at least 1,000 random playouts per second should be achievable.

Move complexity is another issue. The game at which MoGo excels, 9x9 Go, has a branching factor that never exceeds 81 and is in practice much less on average. This number is further reduced by domain knowledge about the game.

3.3 Evolutionary Algorithms

Evolutionary algorithms (EAs) are probabilistic optimisation algorithms based on the model of natural evolution [Bäck & Hoffmeister, 1994]. The essential steps of an EA are shown in Figure 3.2.

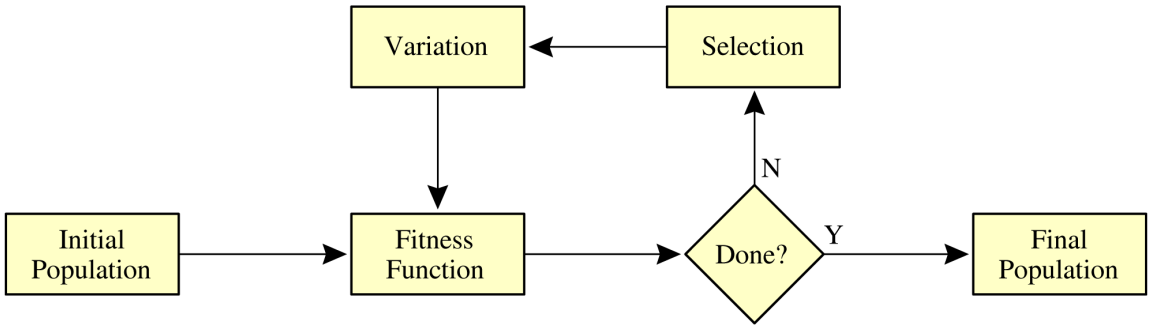


Figure 3.2 The essential steps of an evolutionary algorithm.

Starting with an initial population of individual examples of the problem to be optimised, a *fitness function* is used to estimate the relative worth of each individual. Most EAs use scalar fitness functions to give a single measure of each individual's usefulness.

Alternatively, *pareto scoring* may be used, in which individuals are measured component by component; one individual is said to dominate the other if none of its components are worse than the

corresponding components of the other and at least one is better [Langdon, 1996]. Pareto scoring means that individuals that make an improvement on any part of the problem are preferred, which may produce offspring that are more likely to make improvement on the overall problem.

The following steps are then repeated until the process completes, typically when a solution of sufficient fitness is found:

- Select parents,
- Apply variation to produce offspring,
- Measure offspring.

Selection

Typically two parents are *selected* from the population to mate. Stochastic universal sampling (SUS) is an efficient way to choose individuals from a sorted population such that all individuals are considered but fitter individuals are preferred [Baker, 1987].

Care must be taken to avoid *inbreeding* which may lead to premature convergence of local solutions. Eshelman & Schaffer [1991] describe an *incest prevention* technique to reduce inbreeding; parents are not bred if their genetic distance is not above some threshold. The genetic distance is typically measured by the *Hamming distance* between the individuals, which is the number of differences in their component symbols.

Variation

The parents are mated to produce offspring, typically using the following operations:

- Recombination, and
- Mutation.

Recombination involves merging the genetic material of both individuals into a single individual, for instance by a genetic crossover operation, and *mutation* is the random variation of the resulting offspring. Optimal probabilities with which crossover and mutation should be applied are discussed by De Jong & Spears [1991] and Hesser & Männer [1991] respectively.

The offspring are then measured for fitness. Fit individuals survive to be integrated into the population, while unfit individuals are culled.

3.3.1 Competitive Learning

Newly created games can be assigned default policies based upon their rule sets, but these are unlikely to be the optimal policy for that game's particular requirements. Competitive learning techniques can be used to optimise game policy through self-play; this is the process of weighting the features as described by Levy [1983].

Samuel first described a machine learning method for fine-tuning feature weightings for board evaluation in the game of Checkers [1959]. Since then many techniques have been successfully applied.

Epstein describes a method of *supervised learning* in which the general game player HOYLE learns to improve its advisors weightings in response to games against human opponents [1989]. The programme behaves like an intelligent novice that starts off with a poor understanding of the game that improves over time; policy improvements may in fact provoke higher levels of play from the human opponent and accelerate this learning curve.

Rather than requiring a human babysitter, more efficient supervised learning can be achieved by training the player on a database of games between human experts. Walker et al [1993] had moderate success with the *temporal difference* TD(λ) learning algorithm [Sutton, 1988] for Othello, using databases of 1728, 2400 and 1200 positions. Thrun achieved more success for the game of Chess with the TD(λ) using a database of 120,000 expert games [1995].

Schraudolph et al [2000] also report success with TD(λ) in learning to evaluate Go positions. They estimate that there exist 500,000 record games between Go professionals and serious amateur players in machine-readable format. Ghory [2004] used a database of 200,000 games to train his TD(λ) Go program.

Unsupervised learning techniques in this context generally involve self-play tournaments between competing policies, in which the eventual winner is the optimal policy. This is typically achieved through the *co-evolution* of populations of competing policies. Competitive learning via co-evolution has been successfully applied to Checkers [Chellapilla & Fogel, 1999] and Go [Lubberts & Miikkulainen, 2001].

Tesauro has had great success applying temporal difference learning algorithm TD(λ) to Backgammon in his master-level TD-Gammon program [1992; 1995]. Tesauro believes that this success is largely due to the non-deterministic nature of Backgammon, and that the randomness of the dice rolls leads self-play into a much larger part of the search space than it would be likely to explore in a deterministic game [Pollack & Blair, 1998]. On the other hand, Patist & Wiering [2004] demonstrate that training a TD(λ) system from a database of games results in better play than learning against a random player.

To give an indication of the number of games required for effective learning, TD-Gammon reached strong amateur level after 300,000 games of self-play [Tesauro, 1995].

Unsupervised self-play suffers the problem of learning systems reaching suboptimal solutions due to the minimisation of feedback within the system [Angeline & Pollack, 1993]. This occurs when evolved players fall into cooperative stable states (common lines of play) and do not explore regions of the search space that may occur in general play against new opponents. Evolved players that perform well against other evolved opponents may thus be vulnerable to unconventional moves by the opponent, even those of a beginner.

Unfortunately, much of this previous work is of little help to the research task at hand, namely fast policy optimisation for newly created games. No databases of expert games of any size are available (in fact the games have never been played before), no human sparring partner is readily available, and the number of self-play games is critically limited by the need to quickly optimise policies for a large number of bred games. On the positive side, expert play is not initially required for the bred games, and workable rather than optimal policies will suffice.

The *two-membered evolutionary strategy* (1+1)-ES described by Bäck et al [1991] provides a convenient solution in which the population consists of only two members; a parent individual and a descendant. Policy optimisation using this method is discussed further in Section 7.5.

3.3.2 Genetic Programming

Genetic programming (GP) is an evolutionary algorithm for the optimisation of computer programmes. It extends the genetic model of learning into the realm of structured languages; the populations being optimised are not descriptions of solutions, they are the solutions themselves.

Individuals in GP are typically represented as structured trees. Specifically, the individuals of the population are hierarchical compositions of primitive functions and terminals appropriate to the particular problem domain [Koza, 1994]. The genetic operators used mate parents in GP, crossover and mutation, are similar to those for other EAs.

Crossover in GP typically involves switching nodes and branches between the parent trees. *Directed crossover* may be preferred in some cases, in which those parts of the individual that are working correctly are identified and propagated unchanged, and crossover is only performed on those parts of the individual that need improvement [Langdon, 1996].

Mutation in GP typically involves modifying information within nodes of the individual or modifying the nodes themselves.

Rule safety is of paramount importance in genetic programming; new rule combinations created by crossover and mutation must form valid rule sets.

3.3.3 Emergence

Emergence is the manner in which complex patterns and behaviours can arise out of relatively simple interactions; its hallmark is the sense of much coming from little [Holland, 1998]. In terms of games, one can describe all of the rules but not necessarily all of the products of the rules [Salen & Zimmerman, 2004].

The importance of emergence in this context cannot be overstated. Given that the eventual result of this project will be games randomly evolved from existing predefined rule sets, then any true innovation must come from unexpected and fruitious recombinations of these rules.

Emergence is the reason for the success of evolutionary approaches to a number of artistic applications in which unexpected works of art emerge from simple instructions. See for example Dawkins [1989], Sims [1991], Smith [1991] and Todd & Latham [1994].

In terms of the current study, emergence can be expected in two forms:

- The strategies learnt for new games, and
- The rule sets and play mechanics of the games themselves.

It is hoped that interesting new strategies will emerge from the linear advisor weightings determined during self-play. It is also hoped that interesting new games will emerge from the evolution of existing rule sets.

It is easy to create new games with complex combinations of ludemes. However, rule complexity does not necessarily mean that a game is better; it may simply mean that the game is harder to play [Browne, 2005]. The ideal game has simple, elegant rules, with complex emergent behaviour.

Games are a simple example of the emergence of great complexity from simple rules or laws [Holland 1998]. For example, the game of Hex is as conceptually simple as the game of Tic Tac Toe; players place a piece of their colour each turn and attempt to form a path between their opposite sides of the board. However, Tic Tac Toe is one of the shallowest games in the world and Hex is one of the deepest.

The difference does not entirely lie in the fact that Hex is played on a much larger board and has a therefore much greater move complexity. The key is that Hex taps into fundamental mathematical themes based on connectivity; the concepts involved in the game are intuitively clear, allowing even

novice to players to formulate complex strategies [Bordelon, 2004]. This is a case of depth emerging from simple rules.

Vestigial Rules

Given that rule sets are generally fragile to the slightest modification, it is likely that the random crossover and mutation of working rule sets will result in broken rules or rules left over from parents that are not relevant to the child. Such *vestigial rules* are generally undesirable and culled in genetic programming tasks, however it shall be demonstrated that such superfluous rules can actually be beneficial in game evolution provided that they are irrelevant rather than illegal and do not violate the system's rule safety.

Serendipity

A special case of emergence of particular interest is the ability for two broken rule combinations to combine to produce a working – even exceptional – rule combination. In the absence of any known term, this phenomenon will be referred to as *serendipity*.

The possibility of such cases of a child game not only transcending but repairing the genetic makeup of its parents is of special relevance to the evolution of games. It is very likely that the random nature of the evolutionary process will result in evolved games that include inferior rules which will (hopefully) recombine into superior rule combinations in future generations.

3.4 Summary

Move planning in general games can be achieved by standard adversarial search techniques used for specific game players, provided that the board evaluation function is general enough. This is achieved by maintaining for each game a policy that defines the relative importance of a number of advisors (dedicated feature evaluators).

The UCT algorithm may be preferable for move planning in general games as it would avoid the need for advisors, policies and policy optimisation, however its dependence on speed eliminates may be a limiting factor.

Competitive learning may be used to automatically optimise policies during self-play. However, previous studies have required large databases of expert games or large numbers of self-play games.

Genetic programming, the development of computer programmes using evolutionary methods, provides a method for evolving structured rule trees. Standard genetic methods for recombination (crossover) and mutation may be applied, however care must be taken to ensure rule safety afterwards.

The emergence of complex behaviours out of simple interactions is required if interesting new rule combinations are to evolve from combinations of existing rule sets. Serendipity, the concept of two inferior parents combining to produce a superior child, is especially important for this application, as rule combinations are brittle to change and evolved games will most likely carry at least some inferior rules through their evolution.

Chapter 4

Aesthetics in Game Design

> The beauty of a move lies not in its appearance but in the thought behind it.
> – Siegbert Tarrasch
>
> It is because combinations are possible that Chess is more than a lifeless mathematical exercise. They are the poetry of the game; they are to Chess what melody is to music. They represent the triumph of mind over matter.
> – Reuben Fine

4.1 Introduction

This chapter introduces the relevant aspects of aesthetic study, in particular methods for aesthetic measurement and algorithmic aesthetics. Game design is discussed in this context, with an emphasis on the aesthetic qualities of game designs and how they may be measured.

4.2 Aesthetics

Aesthetics, at its simplest, is the appreciation of beauty.

Birkhoff [1933] describes the *aesthetic feeling* as the intuitive feeling of value that accompanies auditory and visual perceptions, that is clearly separable from sensuous, emotional, moral, or intellectual feeling. He goes on to describe aesthetics as the branch of knowledge concerned with this aesthetic feeling and the objects that produce it.

However, many theoreticians would argue that aesthetics is as valid to intellectual pursuits as it is to sensual ones. For instance, Gardner [2004] describes the mathematician's preference for beautiful (aesthetically meaningful) abstractions, and Aristotle points out that:

> "Those are mistaken who affirm that the mathematical sciences say nothing of beauty or goodness… The main elements of beauty are order, symmetry, definite limitation, and these are the chief properties that the mathematical sciences draw attention to" [Birkhoff, 1933].

Stiny & Gips shift the emphasis in their study of algorithmic aesthetics as follows:

> "Aesthetics is concerned with questions about how existing works of art can be described, interpreted, and evaluated and with questions about how new works of art can be created" [Stiny & Gips, 1978].

It is this emphasis on the interpretation, evaluation and creation of aesthetically meaningful works that shall be followed in this thesis.

4.2.1 Aesthetic Measure

Birkhoff [1933] introduced the concept of *aesthetic measure* as a way to quantify the aesthetic experience using the following basic formula:

$$M = f\left(\frac{O}{C}\right)$$

where:

- O is the *order* of an object based on inherent qualities such as its simplicity, symmetry, balance and harmony,
- C is its *complexity* based on qualities such as number of components and level of detail that might affect the ease with which the viewer perceives it, and
- M is the resulting aesthetic measure.

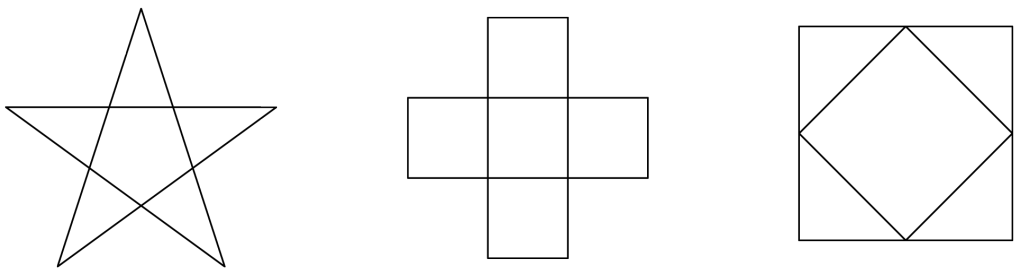
Birkhoff demonstrates the application of this basic principle to a number of specific problem domains, such as the aesthetic evaluation of ornamental art. For this purpose, the general formula is broken down into a number of *aesthetic criteria* specific to that domain:

$$M = f\left(\frac{O}{C}\right) = f\left(\frac{V + E + R + HV + S - F}{C}\right)$$

where:

- V is the object's vertical symmetry,
- E is its equilibrium or balance (essentially, whether the object has a supporting base),
- R is its rotational symmetry,
- HV is the object's invariance to horizontal-vertical transformations,
- S is the similarity of polygons within the shape, and
- F is a measure of unsatisfactory form, a gathering of negative elements of order such as some vertices being too close together, some interior angles being too small, potential ambiguities of shape, and so on.

The overall order O of the object is defined as the sum of these qualities, and the complexity C as the total number of line segments that form part of the object's exterior or outer face. Figure 4.1 shows aesthetic scores of some simple ornaments calculated according to this formula.



0.90 0.87 0.12

Figure 4.1 Aesthetic scores for some simple ornaments.

These measurements are obviously subjective, and whether they actually capture an aesthetic truth is questionable; Birkhoff freely admits that the extreme variability of aesthetic values will limit the precision of any system designed to measure them [1933]. While Staudek later supported Birkhoff's method for the measurement of ornamental vases [1999], Eysenck conducted experiments on subjects' aesthetic preferences for polygonal forms and in fact found $M = f(O \times C)$ more accurate for this case [1968]; the relevant measurement is domain-specific.

Arnheim pointed out in the 1970s that applying information theory to reduce aesthetic form to quantitative measurement had remained largely unrewarding until that point [1971], and suggested that order was a necessary but not sufficient condition for aesthetic excellence.

Entropy, in this context, is the degree of tension or uncertainty in a system. In terms of visual artwork this may describe jarring visual components that do not sit together harmoniously, and in terms of games this may describe rules that do not work together efficiently. Art profits from the economy of orderly structure but this does not mean a lessening of effort; parsimonious structure can be the fruit of laborious struggle [Arnheim, 1971].

As Arnheim states, order is not sufficient, and an elegant game with harmonious rules will not necessarily be interesting to play. However, it is argued that the intellectual appreciation of a well-designed game is a legitimate form of aesthetics with universal themes that are to some degree measurable. The following sections of this chapter outline how such measurements might be approached.

4.2.2 Algorithmic Aesthetics

The *algorithmic aesthetics* model proposed by Stiny & Gips [1978] is a robust framework for the computer-based evaluation and design of objects. The model consists of two main algorithms:

- A *criticism* algorithm.
- A *design* algorithm.

These algorithms are ideal for the two main research tasks of this thesis; the automated evaluation and design of combinatorial games.

An advantage of the algorithmic aesthetics model is that it requires an explicit statement of underlying assumptions and details that would otherwise remain hidden using less rigorous methods [Stiny & Gips, 1978]. In addition to any results the system might produce, the mere act of analysing a field in preparation for the system may yield new insights into it. Franke [1989] also points to the benefits of the aesthetic study of mathematical experiments, in that the deeper understanding of the problem may open up new possibilities for representation and expression.

Criticism Algorithm

The criticism algorithm produces a response to an object as a work of art.

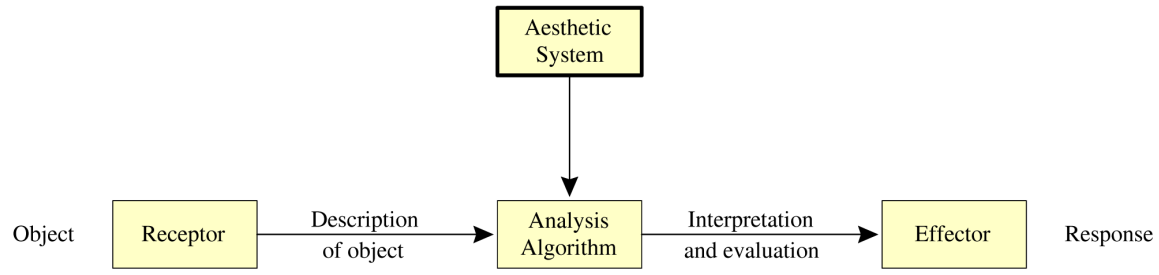


Figure 4.2 Stiny & Gips' criticism algorithm.

The structure of the criticism algorithm is shown in Figure 4.2. The input object is converted into a description that is interpreted by the algorithm and evaluated by an associated aesthetic system, and an aesthetic score produced in response.

Design Algorithm

The design algorithm produces an object as a work of art in response to some initial conditions.

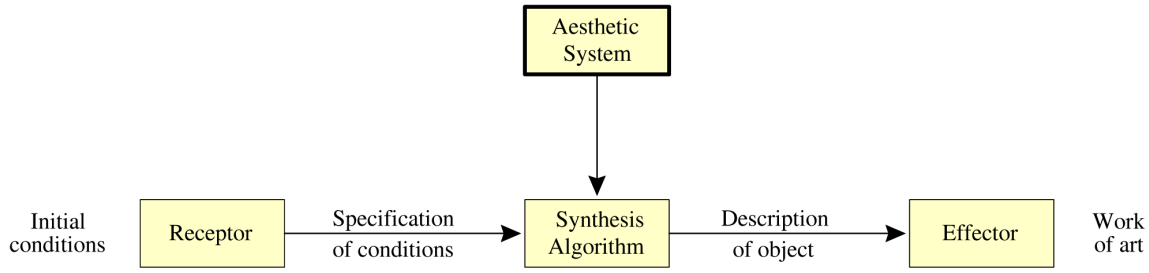


Figure 4.3 Stiny & Gips' design algorithm.

The structure of the design algorithm is shown in Figure 4.3. This algorithm takes as input a set of initial conditions which inform a synthesis algorithm, under the direction of an associated aesthetic system, which produces a description of a new object as a work of art.

Aesthetic System

The core of both algorithms is the aesthetic system, which consists of the following components:

- *Interpretation algorithm*: Interprets the specified object into a symbolic form suitable for the aesthetic system.
- *Reference algorithm:* Indicates whether the interpretation is valid for the specified object for this aesthetic system.
- *Evaluation algorithm:* Produces an aesthetic value for an interpretation.
- *Comparison algorithm:* Compares two aesthetic values.

The interpretation algorithm may be *constructive* or *evocative* in nature. In the constructive mode, objects are understood by telling how they can be constructed or generated. In the evocative mode, objects are understood in terms of associations, emotions or ideas they evoke. For the evocative mode to succeed, it is necessary to specify:

- The aspects of the object which are of interest, and
- The conventions by which these aspects are appreciated.

This distinction between constructive and evocative modes is equivalent to the distinction between the rationalist and romanticist views described by Reich [1993] in relation to the aesthetics of architectural design. The rationalist uses aesthetic laws in the generation process, while the romanticist classifies objects by aesthetic laws after they have been generated.

Pell's METAGAMER [1993b] provides an example of games generated in constructive mode. A *stochastic context-free grammar* (SCFG) method is used in which a probability is assigned to each choice in the game description language, and new rule sets constructed directly from the grammar based on these probabilities according to user preferences and certain constraints. Similarly, Rolle's Morphling [2003] allows the user to experiment with rule variations through a number of constructive steps guided by the user interface.

An advantage of such constructive approaches lies in knowing that the individuals created will be well-formed and should behave in ways that are more or less expected. An advantage of the evocative approach, as embodied by the evolutionary method outlined in this thesis, is that surprising new behaviours may emerge through unexpected combinations, at the expense of a high attrition rate of bad formations.

4.3 Game Design

Game design is the process by which a game designer creates a game, to be encountered by a player, from which meaningful play emerges [Salen & Zimmerman, 2003]. They key term here is *meaningful play*; aesthetics in game design can be described as the study of whether a game provides the player with a meaningful play experience, or simply how interesting a game is to play. As with any artistic medium, good examples should satisfy experts but still be enjoyable to those unskilled in the art.

Although a player's judgement of a game will often be swayed by its presentation, especially in the case of themed games, such considerations will be avoided in this thesis. Combinatorial games are ideal in this respect as the art of the game is in the rule set; such games may be stripped down to their essential rules and played in different formats without changing their basic nature. For this reason, a unified user interface for presenting all games in a similar manner is beneficial.

Games as Art

Combinatorial games neatly fit the dictionary definition of art as "a human creative skill or its application" [Hughes et al, 1992]. Most such games are carefully crafted by their designers, and even those games automatically generated in this thesis are based upon human-defined templates and informed by user preferences. As Ho states, "true art is never created out of nothing" [1991].

More importantly, the function of art is to provoke an emotional aesthetic response; it aims at affecting or moving the perceiver [Nahm, 1933]. Combinatorial games provoke a similar response in many players, however it is intellectual rather than emotional in nature. Allis et al [1991] point out that games which survive do so because they provide an intellectual challenge at a level which is neither too simple to be solvable, nor too complex to be incomprehensible.

Some games catch players' imaginations and pique their curiosity, and must be played again and again until that curiosity is satisfied. Sometimes this curiosity is never satisfied, and the player may devote their life to the study of the game or the study of games in general, both as a career and a recreation; such self-flagellation indicates the strong passions provoked by games.

4.3.1 Game Quality

Qualities of games that trigger such aesthetic responses in players are now investigated.

4.3.1.1 Viability and Quality

It is useful to distinguish between the viability and quality of a game.

Viability

A game is *viable* if its rules combine to work in a basic capacity without serious flaws; the game is playable, offers a fair and meaningful contest between the players and has achievable goals.

A game is seriously flawed if both players must cooperate for it to succeed; an obstructive player should not be able to sabotage the game by always playing for a draw and achieving it.

Quality

The *quality* of a game refers to how well that game plays, or the degree of interest that the game will hold for the human player. In the terms of this thesis, the quality of a game rests entirely in the

intellectual reward offered by its rules and does not include any physical aspects of theme or presentation.

It follows that a game's quality cannot be reliably measured unless that game is viable.

4.3.1.2 Thompson's Big Four

One of the definitive works on combinatorial game design is a short article by amateur game designer Mark Thompson [2000]; the points raised in this article will ring true for most experienced players. Thompson describes four main qualities that a game should possess to have lasting merit:

- Depth,
- Clarity,
- Drama, and
- Decisiveness.

Depth

A game is generally understood to be *deep* if human players are capable of playing it at many different skill levels. Depth gives a game lasting interest because players continue to learn more about the game the more they play it; it is the capacity for a game to surprise even the expert player.

The true depth of most games only becomes apparent after years of study by many players; it is therefore extremely difficult to gauge the depth of newly invented games.

Clarity

Clarity is the ease with which players can form a judgement about which are the most promising moves for a given board position. Games with good clarity allow players to see further down the game tree and spend more of their mental energy formulating strategies. Games with confusing rule sets or unclear goals tend to be *opaque*.

Gobet et al [2004] describe the related concept of *mutational complexity* which is expressed in terms of changes on the board and number of pieces involved in a move. This highlights a subtle distinction between the clarity of the game's movement rules, which determine the amount of trivial bookkeeping required to make legal moves, and the clarity of the game's goals, which generally involves the more interesting strategic aspects of the game. This may be described as the difference between *clarity of means* and *clarity of ends*, or as the difference between *search-space complexity* and *decision complexity* [Iida et al, 2003].

Depth and clarity are closely related concepts. Depth depends on how far ahead, or how many choices, a player can see, and this is decided by the game's clarity [Abbott, 1975]. That is, the clearer a game's rules and objectives are, the deeper a player can look down the game tree with some certainty in order to formulate strategies.

Drama

A game has drama if a player in a weaker position is able to recover and win the game, or, as Schmittberger describes it, the tendency of a game to have reversals of fortune and close finishes [1992]. A losing player should still have a vested interest in the game.
Thompson [2000] goes on to point out that a player's recovery should not occur in a single killer move, but that the suspense should build up over an extended campaign.

Decisiveness

A game is decisive if it is ultimately possible for a player to achieve an advantage from which the opponent cannot recover. Ideally, the game should end quickly when this critical point is reached as the outcome is more or less decided.

Drama and decisiveness are complementary concepts. The key to distinguishing between them is to define the point at which a player is said to have a decisive lead.

Neto [2004] suggests methods for quantifying some of these measurements, and their possible implementation.

Game as Puzzles

Thompson [2000] describes a game as a family of potentially interesting logic puzzles that the players pose to each other; in addition to the above criteria, a good game design should therefore allow an inexhaustible supply of interesting puzzles to be discovered within its possible moves.

The famous Chess puzzler Sam Loyd stated that his goal was to compose puzzles whose solutions require a first move that is contrary to what 999 players out of 1000 would propose [Hayward & van Rijswijck, 2006]. Althöfer [2005] jokingly replies that it may be a better strategy if the solution were somewhere between the two extremes; if the least obvious move were the solution to every puzzle, then players would no longer consider the obvious moves.

Thus *obfuscation* in the moves of a game, namely the need to search beyond the obvious to find the optimal move, is a good quality to possess. When asked what makes a good conjecture, mathematician John Conway immediately replied that "it should be outrageous" [Colton et al, 2000].

4.3.1.3 Other Criteria

Kramer

Professional game designer Wolfgang Kramer presents a longer but less detailed list of qualities that good games should possess [2000]. These points are generally self-explanatory and cover some social and presentational aspects:

- Originality,
- Freshness and replayability,
- Surprise,
- Equal opportunity,
- Winning chances,
- No kingmaker effect,
- No early elimination,
- Reasonable waiting times,
- Creative control,
- Uniformity,
- Quality of components,
- Target groups and consistency of rules,
- Tension,
- Learning and mastering a game, and
- Complexity and influence.

Interestingness

Althöfer [2003] describes a method for calculating the interestingness of a game by making a series of self-play games and measuring key statistics of the outcomes. These include:

- *Game length:* Games should not be too short or too long.
- *Drawing quota:* Games should not end in draws too often.
- *Balance/advantage:* Games should not be biased towards either player.
- *Variability:* Some variation in play is preferred.
- *Performance loss due to variability:* Too much variation leading to random play.
- *Deepening Positivity:* Deeper searches outperform shallower searches.
- *Smoothness:* Discrepancy in move estimates between search plies.

These measurements were implemented in the general game player Morphling [Rolle, 2003] which was used to aid the game design process for a class of combinatorial games. The interestingness of the game is then given by a weighted linear combination of these features; this is essentially the method used in this thesis to produce aesthetic measurements for games.

Althöfer [2003] throws out the challenge that computers are number crunchers and "unsuited to invent completely new nice games in fully automatic mode. Game inventing will always remain a job for creative humans".

Uncertainty

Hiroyuki Iida defines an *entertainment measure* of a game based on the sound premise that a game is interesting as long as its outcome is uncertain [Iida et al, 2004]. In this context, the uncertainty $U_p(H)$ of a group of similar board positions H with respect to player p, is given as a measure of the entropy of the probability p_i of outcome i occurring:

$$U_p(H) = -\sum_{i=1}^{k} p_i . \ln(p_i)$$

where k is a count of the total number of outcomes (three in the case of Chess).

Interaction

Conflict within a game manifests itself as *interaction* between opponents; in a combinatorial game this interaction occurs through the puzzles that players set each other through their moves [Browne, 2005]. Games in which players can safely ignore the opponent's replies and concentrate solely on their own moves are lacking this crucial element of interaction and are generally flawed.

This is closely related to the concept of *tension* within a game, which in turn is closely related to drama. Kramer [2000] emphasises tension as one of the most important aspects of good game design, stating that games should avoid long periods of low tension, and that several peaks of tension should ideally occur throughout the duration of a game.

Originality

A game may surprise players not only in the move combinations that it throws up during play, but also surprise them with its very rules. Players are generally interested in truly novel rule combinations until they are solved or proven flawed; if neither, then they will generally continue to hold players' attentions to at least some degree.

Given that games evolved from a limited population of source games must express themselves as recombinations of those ancestors, any originality in child games must come through the emergence of novel and interesting rule combinations. Penalising inbreeding within the population will encourage the development of original rule combinations.

Tactics and Strategy

Ideally these two aspects will complement each other in a given game. The player should be able to develop creative, deep strategies and be able to lay subtle traps for opponents within the practical choices available to them each turn. Conversely, strong combinatorial play should generally improve a player's position and open up new avenues to explore.

Overly tactical games can get bogged down in the minutiae of low-level combinatorial play that rewards rote learning rather than over-the-board thinking; overly strategic games can feel intangible and boring to all but expert players.

Cultural and Personal Preferences

There exist many other metrics for measuring games involving the presentation of the board, quality of components, and so on (see for example Markley [2003]). Such concerns are beyond the scope of this thesis, as are cultural preferences, as far as they can be avoided.

Game Repair

Many of the quality tests for a game, especially the viability tests, entail searching for pathological flaws in its design. The next step is obviously the correction of such flaws, once detected; this is, however, outside the scope of this thesis.

4.4 Summary

Early studies in aesthetic measure demonstrate how artistic ideals may be quantitatively measured, with varying success. The model of algorithmic aesthetics then expands this idea to define a general framework for complete aesthetic systems. It is argued that games can be considered works of art and such systems applied for their evaluation and creation.

Although most combinatorial game designers seem to agree on a number of fundamental game design principles, these are not precisely defined and therefore hard to measure. A distinction is made between a game's viability (whether the rules basically work) and its quality; only viable games should be measured for quality.

Part II

Ludi Framework

I had noticed that while the machine played quite perfectly, and pounced at once on any mistake of mine, its perfection lay purely in response. It never laid traps for me or did anything unexpected, and after a while I began to suspect that it did not really understand the game very well, and might be disconcerted by unorthodoxy.
– Nigel Balchin, *God and the Machine*

Overview

Part II describes the design and implementation of the Ludi software system developed to investigate the research problem. This framework for the automatic measurement and synthesis of games, and its underlying conceptual models, form the main contribution of this thesis.

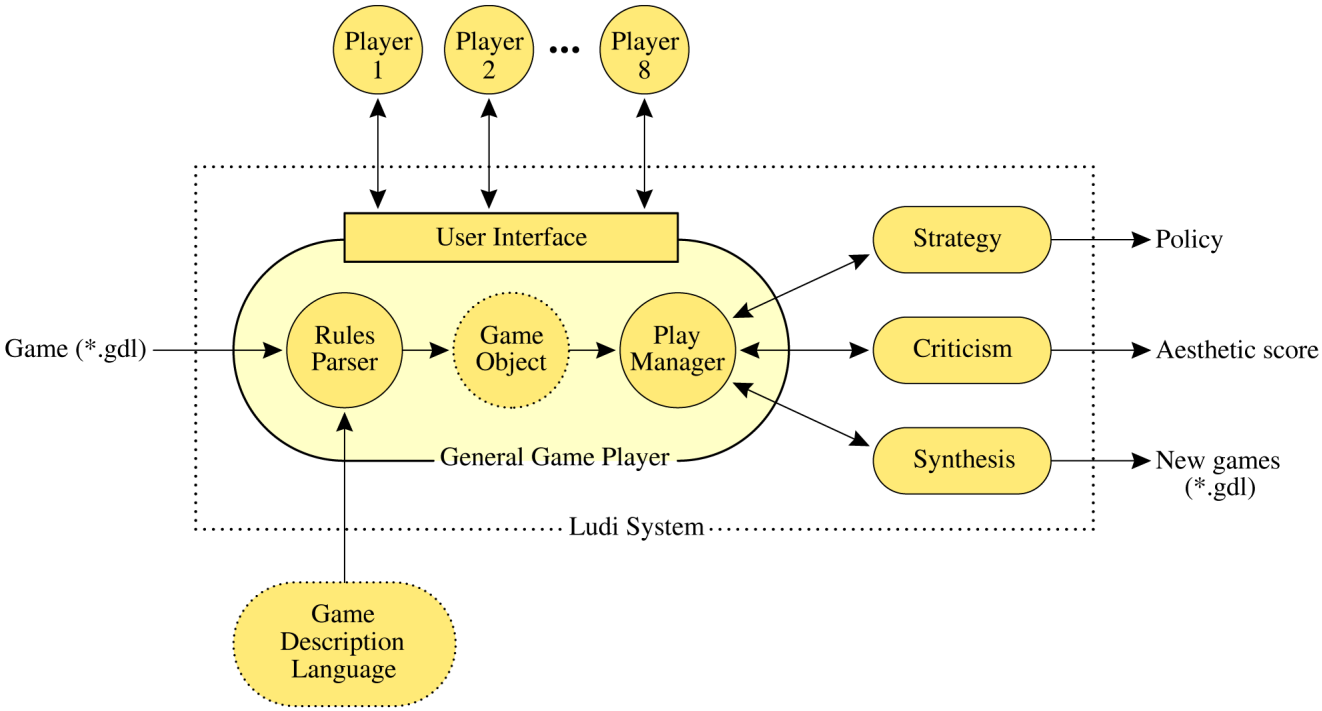


Figure ii.1 Overview of the Ludi system.

The name *Ludi* is derived from the Latin *ludo* meaning "I play". However, the purpose of the system is much broader than to just play, as it is able to:

- Parse rules,
- Play games,
- Evaluate games according to some aesthetic criteria, and
- Create new games.

The core of the system is the Ludi general game player which takes as input a text file (*.gdl) describing the rules of a game in a custom game description language (GDL). Further details of the GDL are given Chapter 5 and a full definition in Appendix A.

The Ludi general game player parses the game's rules for correctness according to the GDL, instantiates a game object, then coordinates play between a number of human and/or computer opponents. The system supports any number of opponents from one to eight, however two opponents are assumed unless otherwise stated. Opponents interact with the system via its user interface as shown in Figure ii.1. Further details of the Ludi general game player are given in Chapter 6.

The Ludi system is supported by three key modules:

- Strategy module,
- Criticism module, and
- Synthesis module.

The Strategy module directs move planning based on a panel of board evaluation advisors. The relative importance of each advisor is dictated by a recommended policy for each game, which may be automatically determined through self-play. The Strategy module ensures that the system can play general games at least at a competent novice level, and is described in Chapter 7.

The Criticism module performs aesthetic measurements on the current game, and produces in response an aesthetic score that is a prediction of how interesting a human player would find the game. This module constitutes one of the key contributions of this thesis and is described in detail in Chapter 8.

The Synthesis module facilitates the creation of new games through the evolution of existing rule sets, and produces as output a number of text files (*.gdl) each describing a new rule set. The adaptation of standard evolutionary methods to achieve this is described in Chapter 9, and constitutes another key contribution of this thesis.

Chapter 5

The Ludi Game Description Language

> You may invent your own men and assign them arbitrary powers. You may design your own boards. And you can have rule games... The possibilities are endless.
> – Henry Kuttner, *Chessboard Planet*
>
> What you see on the board is only the outcrop of a much larger world, like mountain peaks above the mist.
> – Ian Watson, *Queenmagic, Kingmagic*

5.1 Introduction

This chapter describes the Ludi game description language, a grammar specifically devised for this thesis and the associated Ludi general game player. A general game model is proposed, the syntax and general structure of the language is introduced, and key elements briefly explained with examples.

This chapter provides a basic working knowledge of the language and indicates the scope of games that it allows.

5.2 Model

The definitions given in Chapter 2 suggest the general game model shown in Figure 5.1, in which a game is described in terms of its *means*, *play* and *ends*.

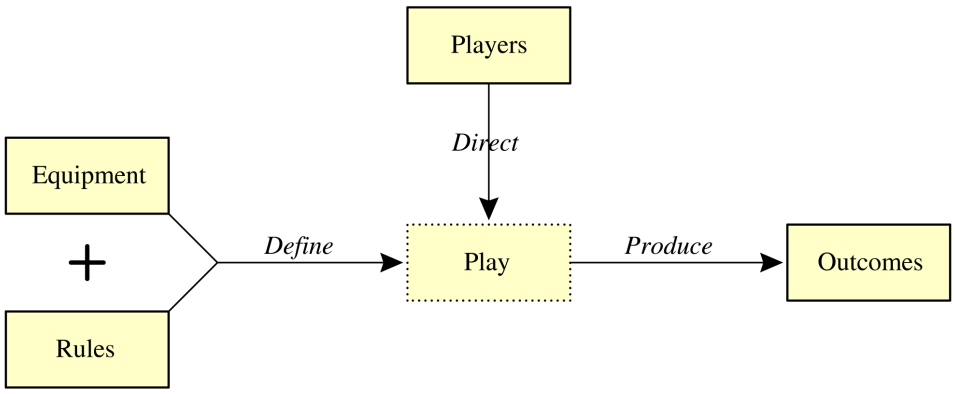


Figure 5.1 The general game model.

Means

The means of a game includes its:

- Rules, and
- The equipment used to play it.

This is a distinction between the abstract and physical constraints that define the game. The term *game mechanic* is also used to describe a rule or group of rules that work together.

Play

The play category includes the players who direct the course of the game through their movement choices and the more intangible concept of the game play itself, which will be studied in more detail in Chapter 8.

Ends

Each game should ideally produce a clearly defined outcome. Typically outcomes include:

- Win,
- Loss,
- Tie (two or more players win),
- Draw (no players win), or
- Abandoned (time or move limit reached).

Tavener [2007] points to other possible types of outcome, such as games in which the game itself may defeat the players. However, only games with outcomes of the above types will be considered in this study.

5.2.1 Ludi GDL

The Ludi GDL provides a language for defining general games based upon this model. This is the game language interpreted by the Ludi general game player.

The GDL is designed with the following principles in mind [Kernighan & Pike, 1999]:

- Simplicity,
- Clarity,
- Generality, and
- Automation.

The language is designed to provide a wide variety of games while ensuring that rule sets remain readily comprehensible to human readers, easily edited and parsable for rule safety. In addition, the rule trees defined by the language are well suited to the task of genetic recombination and mutation as applied by evolutionary algorithms.

Note that the Ludi GDL was designed primarily as a proof of concept within the context of this thesis. It allows the easy description of a wide variety of combinatorial games and is reasonably comprehensive, but does not provide a complete solution for the general description of games. In addition, not all of the language features have been implemented in the Ludi general game player; these are denoted where appropriate.

The Ludi GDL differs from other game description languages in its native support of geometry games, in particular connection games.

The complete grammar for the Ludi GDL can be found in Appendix A, and examples of complete games described in the GDL can be found in Appendices F and H.

5.3 The Language

The following section gives an overview of the syntax and general structure of the Ludi GDL and provides a detailed description of its key elements. This description is useful for delimiting the scope of games allowed by the Ludi GDL.

5.3.1 Syntax

Games are described in the Ludi GDL as recursive trees of elements of the following form:

(*element* [attributes] [(element ...)s])

where:

- *element* is the element's unique name,
- [attributes] denotes an optional set of attributes for this element, and
- [(element ...)s] denotes an optional set of nested subelements.

Italicised terms are keywords – typically predefined element and attribute names – and capitalised terms are names for games, players or pieces. Capitalised terms may also be used as element names in some circumstances.

Optional elements are denoted by square brackets [].

Thus the Ludi GDL has an XML-like structure and feel. XML would have been an obvious choice for language due to its universal usage and portability, however a simpler syntax was chosen to keep game descriptions as short and easily comprehensible as possible; this is very important when analysing or comparing a large number of games by their rule descriptions, especially for novice users.

Note that some predefined element names are not unique within the language, for instance *stack* and *score* subelements may occur in either the *end* or *advisor* elements. In such cases, scoping will resolve any ambiguity.

5.3.2 General Structure

Each game is described by a single text file containing a single tree of elements. The root element of this tree must of type *ludeme* and contain the name of the game. Rules files written in the Ludi GDL are denoted by a *.gdl extension.

The general structure of a game described in the Ludi GDL is as follows:

```
game  →   (ludeme GameName [args]
              (players ...)
              (board ...)
              [(pieces ...)]
              [(start ...)]
              [(play ...)]
              (end ...)
              [support]
          )
```

The optional arguments in the root ludeme are global values that may be accessed and instantiated by variables within their scope, for instance an argument of 7 might be instantiated for the board size or the target number of pieces to capture. The variable %1 instantiates the first argument, %2 instantiates the second argument, and so on.

The *pieces*, *start* and *play* elements are optional and default values are used for the corresponding items if absent.

For example, Tic Tac Toe may be described in the Ludi GDL as follows:

```
(ludeme TicTacToe
    (players White Black
    (board
        (tiling square i-nbors)
        (shape square)
        (size 3 3)
    )
    (end (All win (in-a-row 3) ) )
)
```

Figure 5.2 shows this rule set in its ludeme tree form. Each node corresponds to an element (ludeme) which may contain a number of attributes and subelements.

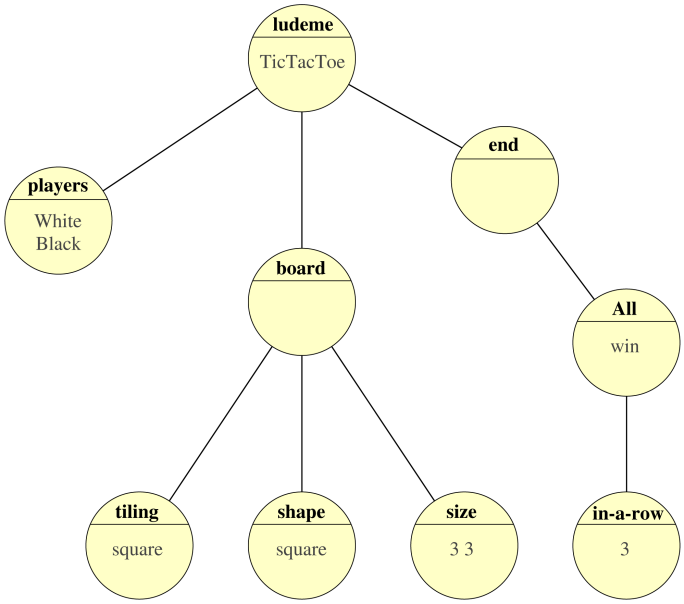


Figure 5.2 Ludeme tree for Tic Tac Toe.

The main element types are now discussed in detail.

5.3.3 Players

The mandatory *players* element describes the number of players, their names and direction of play, if any. A player's direction of play defines a base movement direction for their pieces.

If not specified, a player's default direction of travel is *all* directions, that is, each piece may move in any adjacent direction from its current cell.

Theoretically it should not be necessary to include a *players* element for every game, as a default setting of two players named White and Black can be assumed for most cases. However, it was deemed desirable to explicitly state the nature of these two-person games for the benefit of human readers.

5.3.4 Board

The mandatory *board* element describes the topology of the board upon which the game is played:

```
board  →    (board
                (tiling ...)
                (shape ...)
                (size ...)
                [(region ...)s]
            )
```

The optional list of regions describes specially designated sets of cells.

Tiling

Four predefined types of board tiling are supported:

- Triangular,
- Square,
- Hexagonal, and
- Truncated square (4.8.8 tiling).

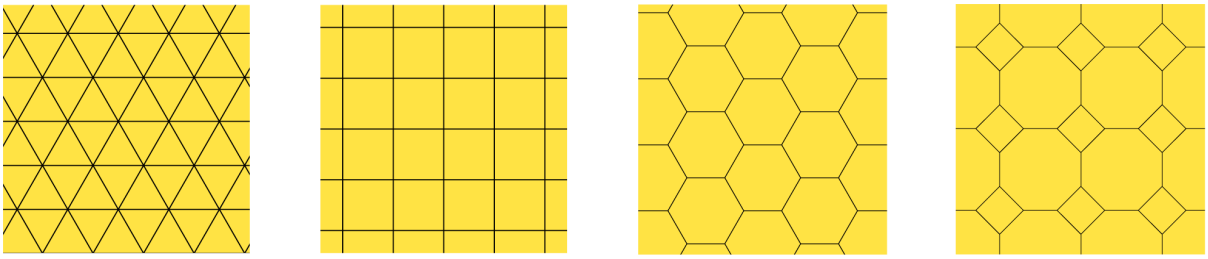


Figure 5.3 Basic tiling types: triangular, square, hexagonal and truncated square (4.8.8).

Adjacencies between neighbouring board cells are critical to game play. For the triangular and square tilings it is necessary to distinguish between *direct neighbours* that share an edge and *indirect neighbours* that share a corner but not an edge. This distinction is equivalent to the distinction made in image processing between *4-adjacent* (orthogonal) and *8-adjacent* (diagonal) neighbours on the square grid [Gonzalez & Woods, 2008], however there is no guarantee in this case that the underlying tiling will be square.

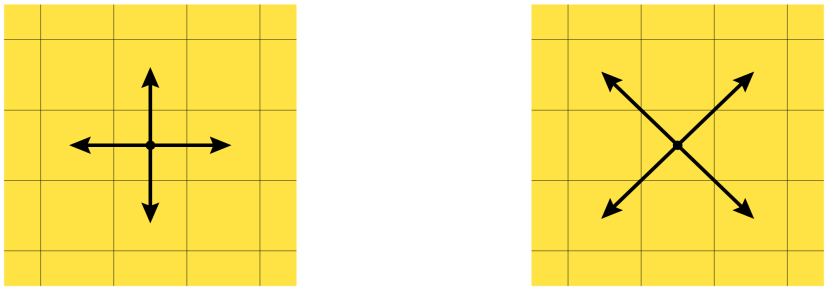


Figure 5.4 Direct neighbours and indirect (diagonal) neighbours.

The hexagonal and truncated square tilings do not have this distinction as intersections at which the corners of neighbouring cells meet are *trivalent* in nature; indirect neighbours do not exist for these tilings.

Trivalent tilings are attractive for board games as all neighbours are of the same direct type. This simplifies movement and avoids deadlock problems in some games; it is, in fact, an essential requirement for the success of many connection games. See Browne [2005] for further details on the importance of trivalence in such games.

For similar reasons, Ellington et al [1982] strongly recommend that board game designers choose hexagonal tiling over the square tiling when creating a new game. The primary motivation behind adding the truncated square tiling to the list of predefined tilings was that it should be theoretically easy to map games directly from a hexagonal tiling to give similar mechanics with a more exotic feel.

The board cells may be assigned *phases*, depending on the game's movement rules and winning conditions, such that no two directly adjacent neighbours exist in the same phase. Ludi supports the single minimal phase space inherent to each tiling. Phases are useful for constraining move choices or bestowing different properties on pieces depending on the phase of their current cell.

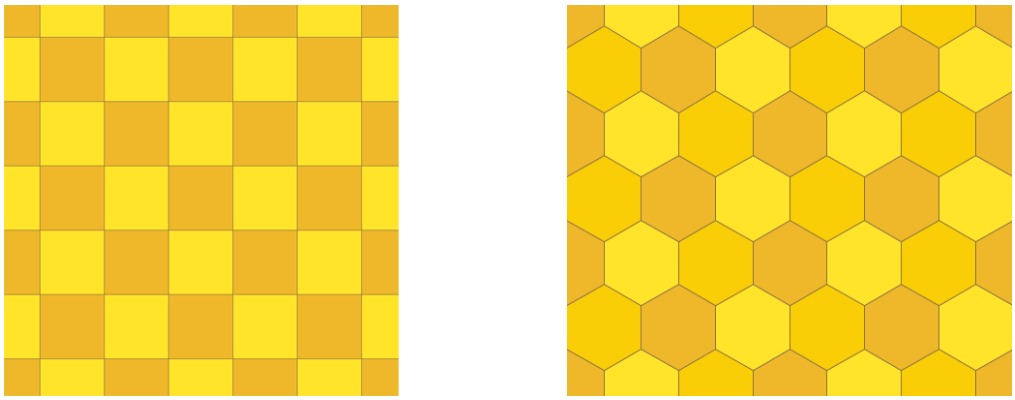


Figure 5.5 Cell phases.

In addition to the predefined tiling types, the GDL allows designers to define boards of arbitrary adjacency. However, this feature is not implemented in the Ludi player, hence playable board surfaces are limited to two dimensions. Three-dimensional play can be simulated to a limited extent through the use of piece stacking.

Shape

The language supports the following predefined board shapes:

- Triangle,
- Square,
- Hexagon,
- Rhombus, and
- Trapezium.

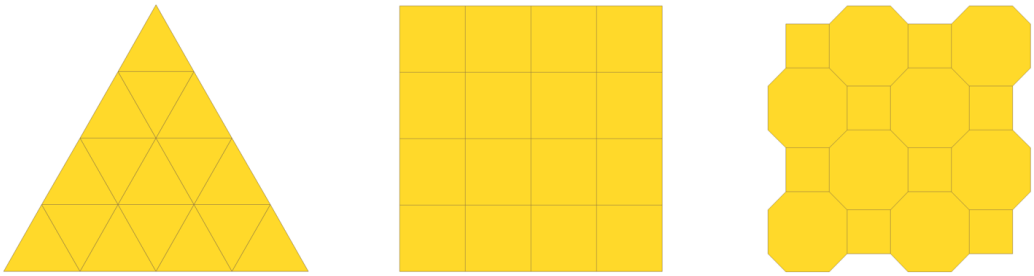


Figure 5.6 Board shapes for triangular, square and truncated square (4.8.8) tilings.

Not all shapes are compatible with all tilings. Figure 5.6 shows the shapes available for the triangular, square and truncated square tilings. Note that a truncated square board will not have reflective symmetry about its axes unless the board size is odd. Figure 5.7 shows the wider variety of shapes available for hexagonal tilings.

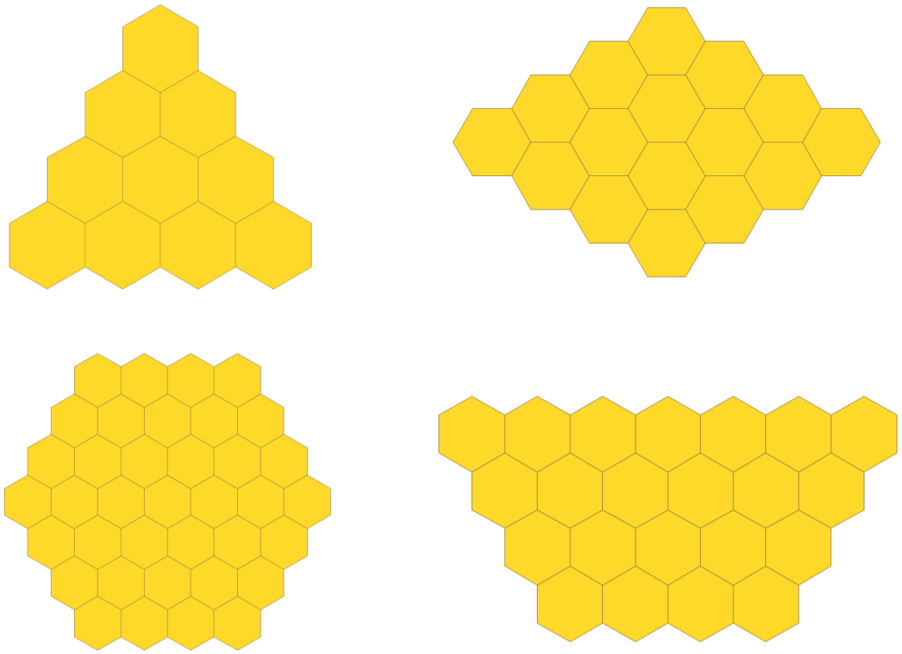


Figure 5.7 Predefined board shapes for hexagonal tilings.

The predefined tiling types are all *regular* tessellations with the same number of potential neighbours per cell, except for the *semiregular* truncated square tessellation which has four and eight potential neighbours for it square and octagonal cells respectively.

Games may also be unbounded or *boardless*, starting off with an empty playing area that increases in a connected but arbitrary manner as the pieces are placed. Figure 5.8 shows a boardless game in progress, with potential moves marked with dots. Boardless games must still obey an underlying tiling despite having no bounded shape.

In addition to the predefined tiling types, the GDL allows designers to define boards of arbitrary shape, although this feature is not implemented in the player.

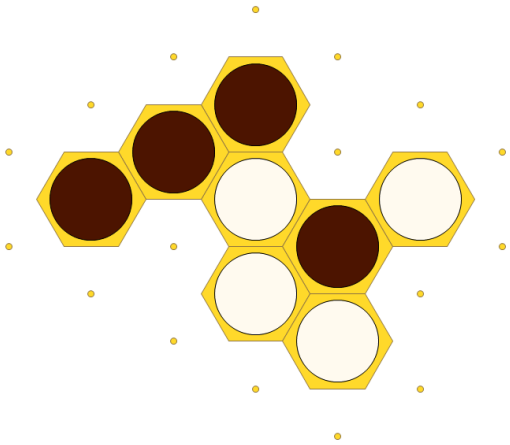


Figure 5.8 A boardless game in progress.

Size

Board size is defined by the number of cells along the shortest side; each of the boards shown in Figure 5.6 and 5.7 are of size four. Square boards may be defined by two numbers to make them rectangular.

Regions

Regions are sets cells designated as special, typically for designating starting positions, constraining movement or specifying end conditions such as target cells to reach or target regions to connect.

Regions may be described in a number of ways, including:

- Compass direction,
- Player direction,
- Board topology,
- Cell coordinates, and so on.

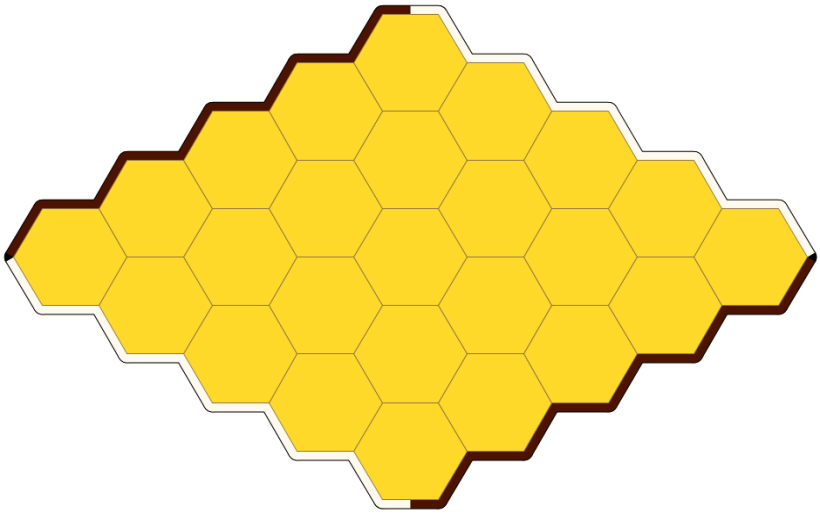


Figure 5.9 White and Black regions around the board perimeter.

Each region is associated with one or more players, and is indicated by an appropriately coloured border where possible. For example, Figure 5.9 shows the regions defined by the keyword *alternating-sides*.

Additional Equipment

The GDL makes provision for other types of playing equipment such as dice or similar probabilistic devices, but these are beyond the scope of this thesis and not implemented in the Ludi player.

5.3.5 Pieces

The optional *pieces* element contains a subelement for each type of piece used in the game, each of which in turn contains a piece definition describing the initial state of the piece and the movement rules specific to that piece.

The *pieces* element therefore describes a merging of the physical (the pieces) and the abstract (their movement) aspects of the rules. However, this is the most convenient and logical way to structure this information.

5.3.5.1 Piece Definitions

Each piece is defined by a subelement of the following form:

```
piece_defn →    (PieceName PlayerType
                    [(label ...)]
                    [(value ...)]
                    [(state ...)]
                    [(flags ...)]
                    (moves ...)
                )
```

Each piece element is denoted by the unique capitalised piece name rather than a keyword. This is the internal name by which the piece is referred to in the rules; the user may optionally specify a label that is shown for pieces of that type in the board display.

The piece definition includes the name of the player to which the piece belongs, which may be "All" for all players. The piece definition may optionally specify initial:

- *state*: Typically indicates promotion or constrains movement.
- *value*: Indicates the relative value of the piece.
- *flags*: Typically used to indicate allowed directions of travel.

Board cells also provisionally have such state, value and flag values in the language, although these are not implemented for cells in the Ludi player. Piece flags were not implemented as incorporating flagged movement and piece rotations would dramatically increase the branching factor of many games, leading to excessive move search times.

5.3.5.2 Movement Rules

Every piece definition contains a *moves* subelement listing its movement rules. Each movement rule is of the following form:

```
move_defn → (move
                [(dirn ...)]
                [(pre ...)s]
                (action ...)s
                [(post ...)s]
            )
```

The piece has an optional *dirn* element specifying its base direction of travel. If not specified, the default value is set to the owner's direction of travel.

Piece movement is generally described by relationships between the move's *from* and *to* cells. Specific moves may be programmed relative to the *from* and *to* cells using:

- Turtle instructions { *l* | *r* | *f* | *b* } for left, right, forwards or backwards, relative to the piece's base direction,
- Absolute compass directions { *n* | *s* | *e* | *w* | *ne* | *se* | *sw* | *nw* } which may have different connotations regarding the board tiling type, or
- Some other indicator such as flagged direction, cell phase or region.

For example, Figure 5.10 shows the right-handed knight's move described by the turtle instructions {*f, f, r, f*}.

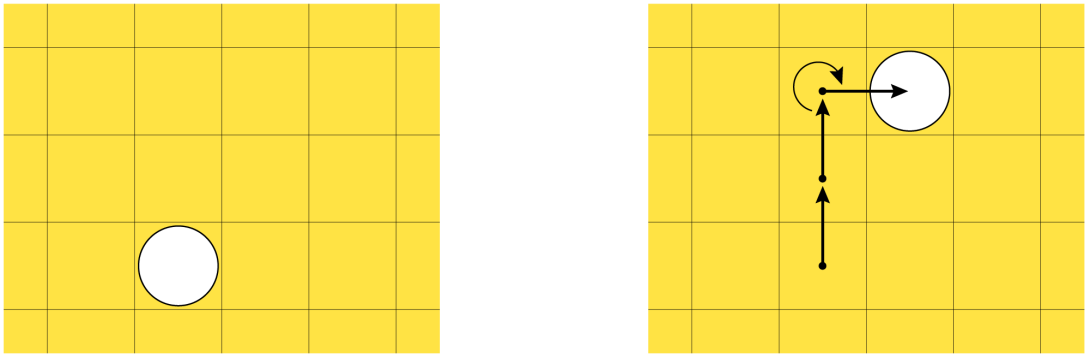


Figure 5.10 Piece movement by turtle instructions {f, f, r, f}.

Preconditions

The optional *pre* element describes conditions that must be true for the move to occur, typically that the piece belongs to the current player, the destination cell is empty, and so on.

Arbitrarily complex preconditions may be constructed using logical operators. There are approximately 50 predefined boolean and integer functions described in the language, allowing a rich variety of movement constraints; see Appendix A for a complete listing.

Actions

The mandatory *action* element describes the actual piece movement, which may be any combination of the following operations:

- *pop* (remove) the piece from the *from* cell, and
- *push* (add) the piece to the *to* cell.

The number of pieces that the actions apply to may be specified. Stack moves may optionally leave a trail of pieces by unstacking along the line of movement.

Postconditions

The optional *post* element describes additional actions to be performed immediately after moving the piece, subject to certain conditions. Typically these involve:

- Capturing or converting enemy pieces,
- Promoting the piece,
- Rotating the piece,
- Updating the piece state,
- Modifying the piece flags,

and so on.

For example, Figure 5.11 shows a linecap conversion move described by the following movement rule:

(*move*
  (*pre* (*empty to*))
  (*action* (*push*))
  (*post* (*convert cap*))
)

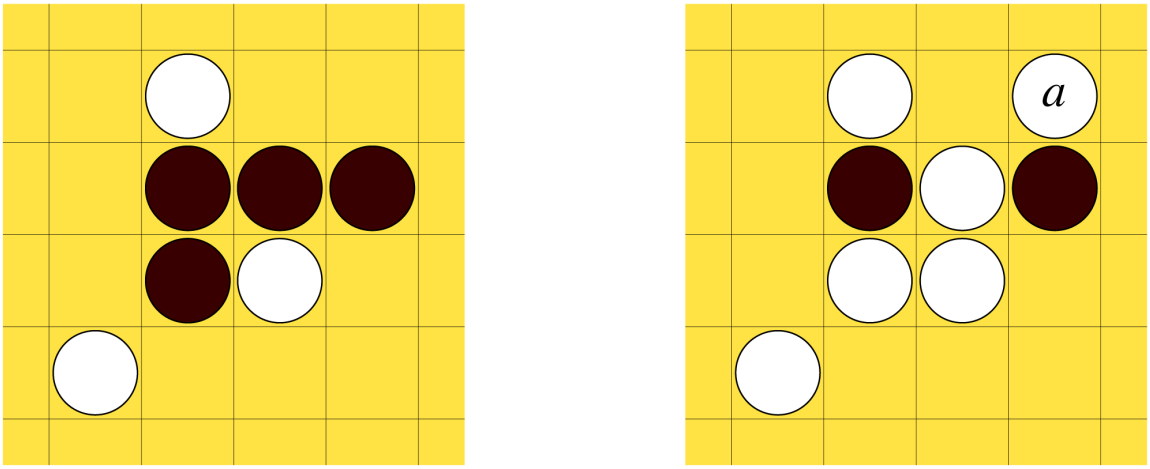


Figure 5.11 Linecap conversion.

White places piece ***a*** at an empty cell as per the precondition (*empty to*) and action (*push*), following which the line of enemy pieces capped at both ends is converted to White as per the postcondition (*convert cap*).

As with the *pre* element, arbitrarily complex postconditions may be defined using logical constructions of the language's predefined boolean and integer functions. For example, Figure 5.12 shows a complex move described by the following compound movement rule:

```
(move
    (pre
        (and
            (owner from)
            (empty to)
            (line)
            (or
                (> (num-between empty) 0)
                (= (num-between enemy) 1)
             )
        )
    )
    (action (pop) (push) )
    (post
        (capture (if (>= (height current) 2) )  d-nbors)
        (inc-state)
    )
)
```

This compound movement rule states that piece belonging to the current player may move in a line to an empty cell, provided that the line contains either no empty spaces or a single enemy piece, following which directly neighbouring enemy stacks of height two or more are captured and the piece state is incremented.

The numbers 1 and 2 displayed on the piece indicates its state; the absence of such values on other pieces implies that they are in state 0.

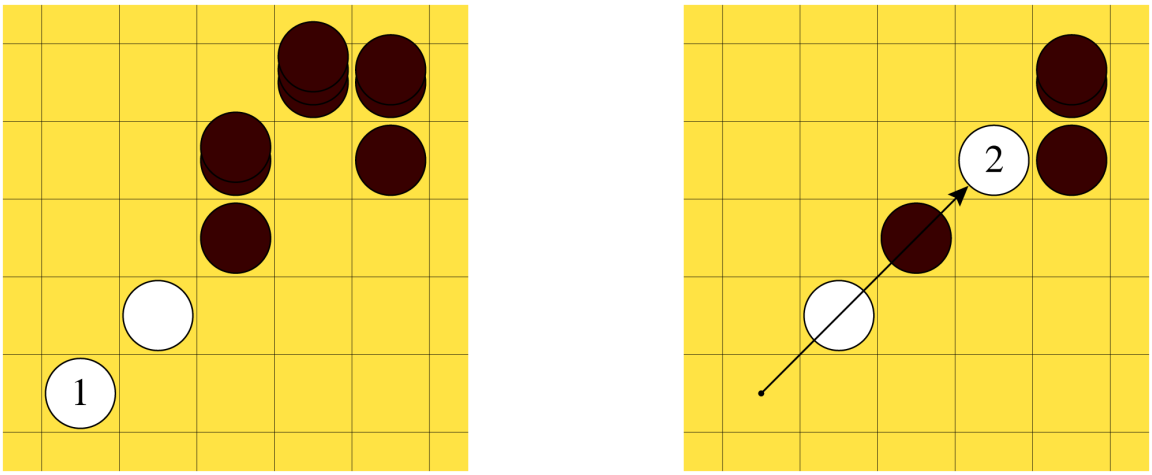


Figure 5.12 A move allowed by the compound movement rule listed above.

This exact movement rule is unlikely to occur in an actual game, but indicates how complex moves can be constructed from simple predicates.

Default Piece Values

Pieces start with *state* and *flags* initialised to 0 and a *value* of 1, unless otherwise specified.

If no *pieces* element is specified then the following default piece type is assumed:

```
(pieces
    (Piece All
        (moves
            (move
                (pre (empty to) )
                (action (push) )
            )
        )
    )
)
```

In this situation, players have access to an unlimited number of such default pieces of their colour, and may play one at an empty cell each turn.

5.3.6 Starting Position

By default, games start with an empty board and an infinite supply of pieces. More interesting starting positions may be defined using the optional *start* element:

```
start  →    (start
                [(in-hand ...)]s
                [(place ...)]s
            )
```

Pieces In Hand

The optional *in-hand* elements specify how many pieces of particular types particular players start with. This is useful for limiting the number of pieces that may enter into play, or for imposing a strict limit on the number of moves in a game.

Placements

The optional *place* elements specify pieces to be placed on the board at the start of each game, to achieve a particular starting position. Placement locations are typically:

- Players' home areas,
- Particular board edges,
- Particular regions, or
- Particular cells.

The number of rows to be filled may be specified, as may the number of pieces to be placed at each position. For instance, Figure 5.13 shows two common starting position corresponding to the placement rules (*place All* (*home* 2)) and (*place All opposed* 3) respectively.

The first example (left) requires that the player be assigned directions from which their home regions are deduced. The second example (right) places pieces at the opposed corners defined by the first and last cell to be created by the Ludi player for that board. The type of placement is not precisely defined for triangular and trapezoidal board shapes.

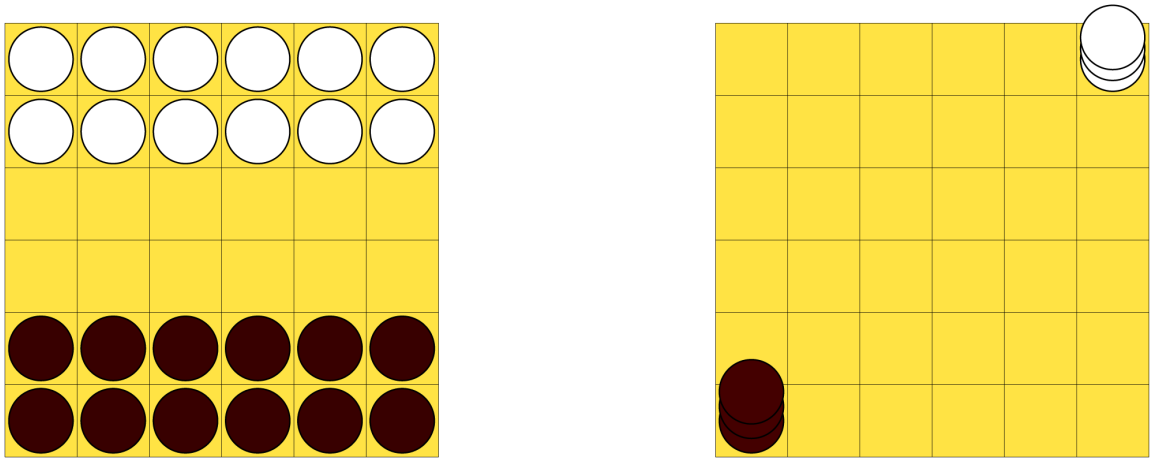


Figure 5.13 Typical starting positions.

5.3.7 Play Order

The optional *play* element governs the order of play throughout the game. The default play order is *discrete* and *cyclic*: players take turns making moves.

Other types of play order might include:

- *Opening contract:* To reduce first move advantage (e.g. swap option).
- *Multiple moves:* Players may make more than one move per turn.
- *Simultaneous movement*: Two or more players move at once.
- *Move order:* Next player is decided as a function of play.

While these variations and others like them increase the diversity of possible games and can lead to interesting play mechanisms, they can also dramatically increase the complexity of the games to which they are applied. In the interests of minimising complexity, only the default discrete cyclic play order is implemented in the Ludi player; this proved sufficient for the range of games studied in this thesis.

5.3.8 End Conditions

The mandatory *end* element specifies the conditions under which the game ends, that is, the winning conditions of the game. This is the most critical aspect of almost any game and describes, in conjunction with movement rules, the essential nature of the game.

The end element contains one or more end clauses, as follows:

end_clause → (PlayerName { *win* | *lose* | *draw* } conditions)

Each end clause may be applicable to a specific player or to All players. The specified conditions, when satisfied, may result in a *win*, *loss* or *draw* for that player (the Ludi player does not distinguish between draws and ties in this context).

The conditions themselves are described by an arbitrarily complex logical expression similar to the *pre* and *post* movement conditions. For example, the following end condition describes that either player wins if they have no available moves and have either connected their regions with a path of their pieces or formed a stack of height 4, and that White loses by forming a stack of height 5:

```
(end
    (All win
        (and
            (no-move)
            (or
                (connect own-regions)
                (stack 4)
             )
        )
    )
    (White lose (stack 5) )
)
```

The fact that these end conditions are unbalanced in Black's favour means that the game's starting position or movement rules should benefit White in some way to redress the balance.

The following types of end condition are supported.

Capture

The *capture* end condition is satisfied when the specified player captures the specified number of target pieces.

Reach

The *reach* end condition is satisfied when the specified player reaches the specified goal. The goal may be:

- A region,
- A cell,
- The far side of the board,
- A specified side of the board, or
- Any side of the board.

Players may reach the goal with any of their pieces, unless a particular piece type is specified or it is specified that all pieces must reach the goal.

***N*-in-a-Row**

The *n-in-a-row* end condition is satisfied when the specified player form a line of *N* pieces of their colour, where *N* must be specified. The line may be specified as lying along direct neighbours, indirect neighbours or both, depending on the board tiling.

Connect

The *connect* end condition is satisfied when the specified player connects the specified regions with a connected path of their pieces. Piece connectivity may be specified as direct, indirect or both, depending on the board tiling. At least two regions to connect must be specified.

For instance, Figure 5.14 shows a game in which players must form a path between their two alternating sides of the board (left) and a game of Y in which player must connect all three sides of the board with a path of their pieces (right). Region colours are shown around the borders of the Y board as both players share all edges.

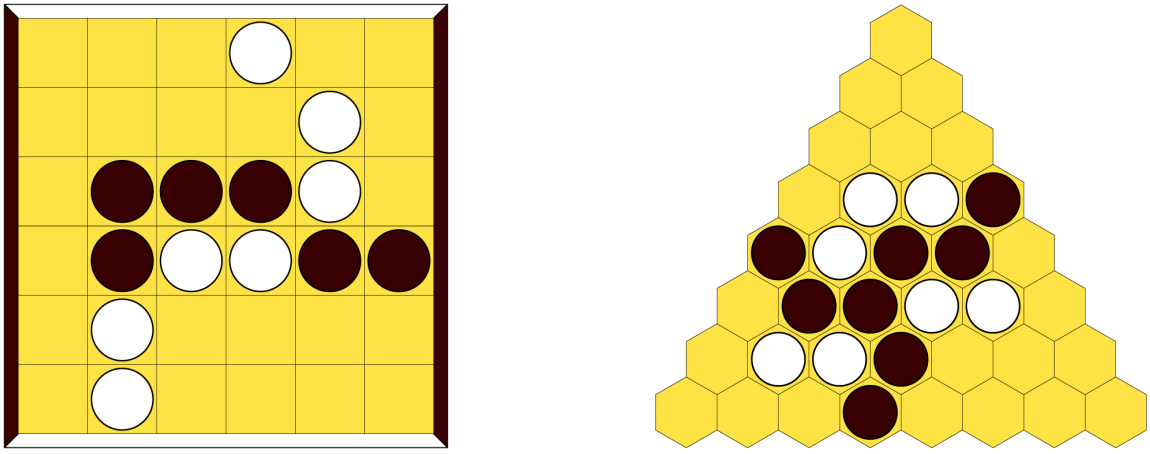


Figure 5.14 The connection end condition.

The game on the left shows a win for White if indirect (diagonal) connections are specified; if not, this particular game is deadlocked and cannot be won by either player. Such ambiguities do not exist for the game on the right (won by Black) due to the trivalent nature of the hexagonal tiling.

Group

The *group* end condition is satisfied when the specified player gathers all of their pieces into a connected group. Connectivity may be specified between direct neighbours, indirect neighbours or both, depending on the board tiling. Stacked piece may be specified as hidden or not.

Stack

The *stack* end condition is satisfied when the specified player creates a stack of a certain height. The stack height may be given by either the number of the owner's pieces in the stack or the total stack height, as specified.

State

The *state* end condition is satisfied when the specified player achieves the specified state with one of their pieces.

No Move

The *no-move* end condition is satisfied when the specified player has no available moves on their turn.

Not Implemented

The following end conditions are defined in the language but not implemented in the Ludi player. This was due mostly to time constraints, their complexity and resulting impact on programme performance, or simply lack of generality and hence applicability for this project. However, their definitions are left the language as reminders that additional useful end conditions do exist.

Cycle

The *cycle* end condition is satisfied when the specified player completes a cycle (closed loop) of their pieces. The connectivity of the pieces may be specified, and whether the cycle must be full or not.

This end condition was removed from the implementation due to the poor performance of the cycle advisors.

Eliminate

The *eliminate* end condition is satisfied when the specified player eliminates the specified opponent(s). This was not implemented as it is more relevant to multiplayer games and a similar effect can be achieved using the (*capture all*) end condition.

Pattern

The *pattern* end condition is satisfied when the specified player achieves the specified pattern of pieces.

Score

The *score* end condition is satisfied when the specified player achieves the specified score.

5.3.9 Support Elements

The support elements are not ludemes as such, but constitute *metadata* relevant to the game that is embedded in the game's description for convenience. The support elements provide additional information about the game to the Ludi player, in order to assist it with move planning, game evaluation and game creation. These elements are not recombined when games are evolved.

support → {
[(*advisors* ...)]
[(*ancestry* ...)]
[(*ranking* ...)]
[(*viable* ...)]
[(*score* ...)]
[(*description* ...)]
[(*aim* ...)]
}

Advisors

The optional *advisors* element contains a number of subelements listing the advisors relevant to the game and their relative weightings. This set of values therefore represents the recommended policy for this game.

A complete listing of the advisors and their implementation details may be found in Chapter 9.

Ancestry

The optional *ancestry* element contains information regarding the genealogy of evolved games. This includes:

- The names of its two parents,
- The maximum number of generations leading to this game, and
- The minimum genetic distance of this game to its parents, all source games, and all other games in the population.

Ranking

The optional *ranking* element describes a discrete ranking of each source game within its data set. Rankings are determined in Experiment I then used in Experiment II to estimate an aesthetic for each game.

Viable

The optional *viable* element indicates a game's estimated viability. This is measured during self-play and is based on the game's observed:

- Completion rate,
- Balance,
- First move advantage, and
- Average game length.

Score

The optional *score* element describes a game's predicted aesthetic score based on either its ranking (if it is a source game) or its weighted sum of aesthetic measurements (if it is an evolved game).

Note a given *score* element may refer to a game's end condition, advisor weighting or support element. Scoping will resolve any potential ambiguity.

Description and Aim

The *description* and *aim* elements may be used by the game author to include text descriptions of the game and its winning conditions, respectively. These are legacy items largely superseded by the Ludi player's "plain English" feature as described in Section 8.4.1, but may still be used if the author wishes to pass special notes on to the end user or phrase the rules more elegantly than the automated interpretation provided by the Ludi player.

5.4 Custom Ludemes

Provision is made in the Ludi GDL for some degree of game customisation through the use of ludeme substitution and metarules. These mechanisms are not implemented for the Ludi player but are mentioned for completeness.

By relaxing the expected format of the *ludeme* element, it is possible to define custom elements by name and instantiate those elements when that name is used. For instance, the Tic Tac Toe board and end conditions described earlier may be encapsulated in the following custom ludemes:

```
(ludeme TTT_Board
    (board
        (tiling square i-nbors)
        (shape square)
        (size %1)
    )
)

(ludeme TTT_End
    (end (All win (in-a-row %1) ) )
)
```

These custom ludemes encapsulate several subitems of information into more complex single items of information.

Such custom ludemes may be loaded by the Ludi player and stored in ludeme database, indexed by unique name. When instantiated, they expand their encapsulated information in-place, hence the game may now be described more succinctly as:

```
(ludeme TicTacToe 3
    (players White Black
    (TTT_Board)
    (TTT_End)
)
```

In fact, the entire game may be described by the following custom ludeme:

(TicTacToe 3)

Care would have to be taken when expanding such custom ludemes to ensure correct scoping, argument matching, and so on. Anti-collision rules would need to be specified, for instance if instantiation resulted in two board elements being defined.

A similar process is used during the crossover operation in game recombination. However, that process is rule-safe as elements are only overwritten with equivalent or compatible elements.

Metarules

Metarules may be incorporated into the language by defining rule modifications that are applied to subsumed elements. For instance, the following Misere metarule inverts the outcomes of the subsumed game:

(*metarule* Misere
    (*end* (*set win lose*) (*set lose win*))
)

Misere Tic Tac Toe may then be defined as follows:

(Misere (TicTacToe 3))

This game is lost by the first player to form a line of 3 pieces.

These custom ludemes and metarules were not implemented for two main reasons. Firstly, elaborate safeguards would be required to guarantee rule safety under random recombination, possibly constraining the use of the language itself. Secondly, it was deemed preferable that rules should be split open and remixed at the lowest level possible during evolution in order to encourage the emergence of truly novel combinations, and that remixing chunked information at higher levels, such as custom ludemes and metarules, might run the risk of producing a large number of similar variants.

5.5 Summary

The Ludi game description language (GDL) is presented, based upon a new model of combinatorial games incorporating *means*, *play* and *ends*. The Ludi GDL defines a general class of combinatorial games in terms of structured trees of elements corresponding to rule sets. The main elements are:

- *players*,
- *board*,
- *pieces*,
- *start*,
- *play*, and
- *end*.

In addition, a number of support elements can be embedded in the rule set to provide additional information regarding a game's recommended policy, evolution history, predicted preference score, and so on.

The Ludi GDL allows the definition of a variety of combinatorial games; the exact scope of games is implied by the language description given above in conjunction with the complete game grammar given in Appendix A. However, this particular GDL was designed primarily as a proof of concept for this thesis and not all aspects of it have been implemented; it is not a complete solution for the general description of games.

Rules sets written in the Ludi GDL are readily comprehensible to human readers, easily edited and automatically parsable for rule safety. The rule trees defined by the language are well suited to the task of genetic recombination and mutation as applied by evolutionary algorithms.

Chapter 6

The Ludi General Game Player

> Play the opening like a book, the middle game like a magician, and the endgame like a machine.
> – Spielmann
>
> Much computation brings triumph. Little computation brings defeat.
> – Sun Tzu, *The Art of War*

6.1 Introduction

The Ludi general game player parses rule sets defined in the Ludi game description language and facilitates play between a number of human and/or computer opponents. This chapter describes the structure, implementation and basic operation of the general game player.

Of necessity, the implementation closely follows the format of the game description language. Key design decisions are highlighted where appropriate.

6.2 Model

The Ludi general game player forms the core of the Ludi system, as outlined in Figure ii.1. Figure 6.1 reiterates the part of this overall model specific to the general game player.

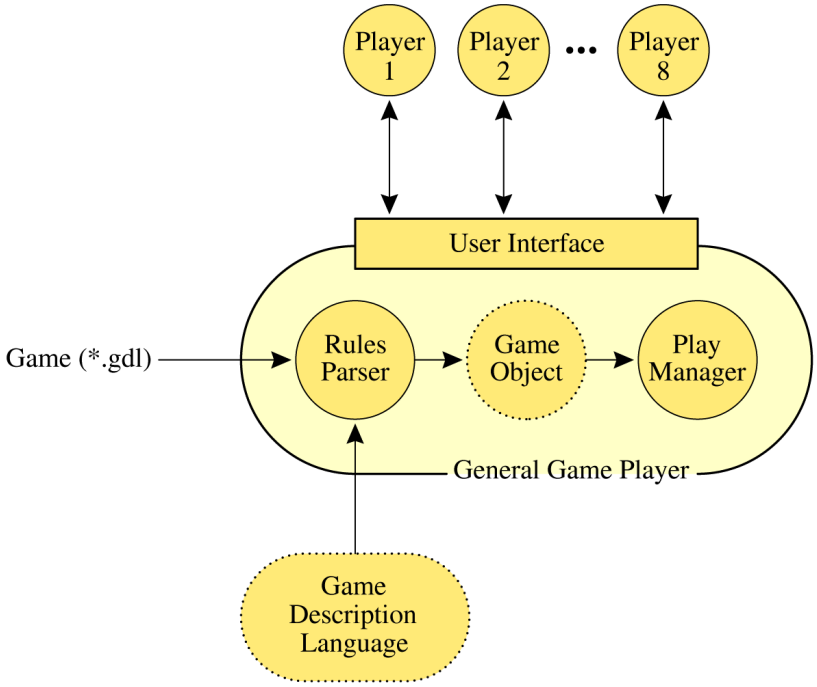


Figure 6.1 The general game player model.

The main components of the general game player are:

- Rules parser,
- Game object,
- User interface, and
- Play manager.

Program Design

The Ludi general game player was written in C++ using the Visual C++ 6.0 compiler and Microsoft Foundation Classes (MFC). The application follows the standard MFC *single document interface* design and includes a single game object; it can load and play one game at a time.

The class structure of the application and further implementation details are given in Appendix B.

6.3 Rules Parser

Each game is defined by a single plain text rules file with the extension *.gdl, which contains the rule tree of the game according to the Ludi game description language presented in the previous chapter.

Rules files are parsed for correctness in two steps as each game is loaded:

- Ludeme tree construction, and
- Game object construction.

Ludeme Tree Construction

Firstly, a ludeme tree is constructed according to the rules file. The load is aborted if the rules file is not well-formed, that is, if it does not describe a nested tree of elements (with optional attributes) and matching brackets.

Variables corresponding to optional arguments in the root element, if any, are expanded in the tree at this point.

Game Object Construction

Secondly, the master game object is constructed based upon the information contained in the ludeme tree; stringent error-checking is performed at this point. All elements and attributes are tested for correctness according to the GDL, their ranges checked and their contexts relative to parent and child elements tested as appropriate.

Separate routines are used to interpret each of the main element types:

- *players*,
- *board*,
- *pieces*,
- *start*,
- *play*,
- *end,* and
- *support elements*.

Details of the resulting game object are provided in the following section.

6.4 Game Object

The single game object contains the following main components:

- Ludeme tree,
- Player descriptions,
- Board object,
- Piece definitions,
- Start items,
- State object,
- End conditions, and
- Move history.

The game object governs most aspects of the currently loaded game and its play, apart from its visual display and message-based move handling.

Ludeme Tree

The game object contains a pointer to the root of its ludeme tree. This is typically used to display the rules to the user, or to manipulate the rules to modify the game, following which the game object is reconstructed.

Player Descriptions

The Ludi player supports between one and eight players. Each player is described by a player object which contains details such as:

- Name,
- Colour,
- Index (move order),
- Type (*human* or *computer*),
- Style (*balanced*, *selfish*, *obstructive*, etc),
- Base direction,
- Policy, and
- Search settings (depth, width, time limit).

The play styles are used for the game measurement as described in Section 8.3. The base direction is the default direction of play for that player. The player's policy is a vector of weights that instruct the board evaluation function.

Additional player details dependent on the current game state – such as the player's score, number of pieces in hand, number of pieces captured, and so on – are maintained separately in the game's state object.

Board Object

The single board object describes the topology and constraints of the game's playing area, and is described in terms of *sites* and *regions*. The board object is constructed once when the game is loaded but reinitialised for each new game.

Sites

A site object is created for each playable board cell. Data members indicate whether each site:

- Exists in a particular phase,

- Is a corner, edge or interior cell, or
- Is an implied playable point in a boardless game, that is, a potential board cell rather than an actual board cell.

A list of adjacent neighbours is created for each site based on the tiling and board shape. Neighbours are distinguished as being either *direct* (share an edge) or *indirect* (share a corner but not an edge). Indirect neighbours are only created for games that include the *i-nbors* keyword at any point in the rules file.

All possible *bridge* formations between cells are detected and stored at this point. Each bridge consists of two non-adjacent *terminal* cells T_0 and T_1 that share a pair of commonly adjacent non-intersecting *carrier* cells C_0 and C_1, as shown in Figure 6.2.

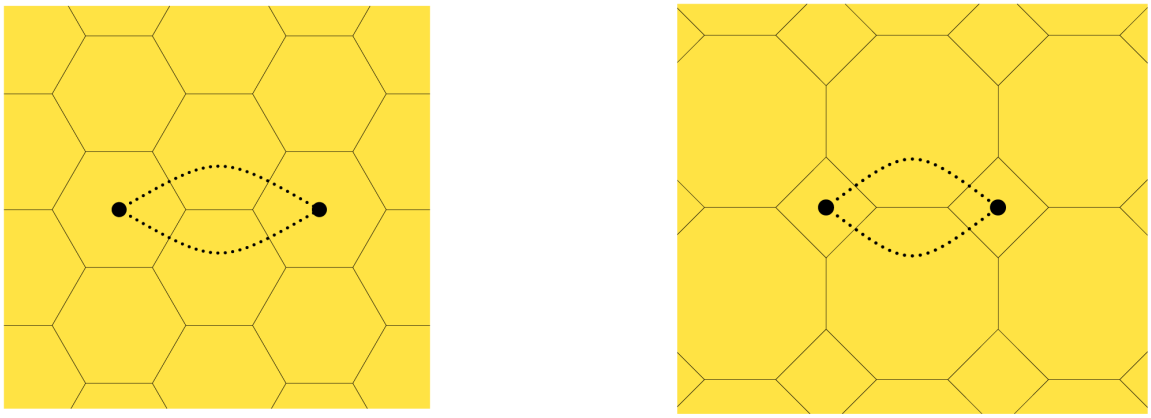


Figure 6.2 Bridges on the trivalent hexagonal and truncated square tilings.

The two carrier cells constitute alternate paths between the terminals that provide virtual connectivity between them; if the opponent occupies one carrier cell then the player can occupy the other carrier cell next turn to restore the connection. Bridges are especially useful in evaluating board positions in connection games.

Bridges on the trivalent hexagonal and truncated square tilings are unambiguous, as shown in Figure 6.2. However, bridges on the non-trivalent triangular and square tilings depend on whether indirect neighbours are specified or not.

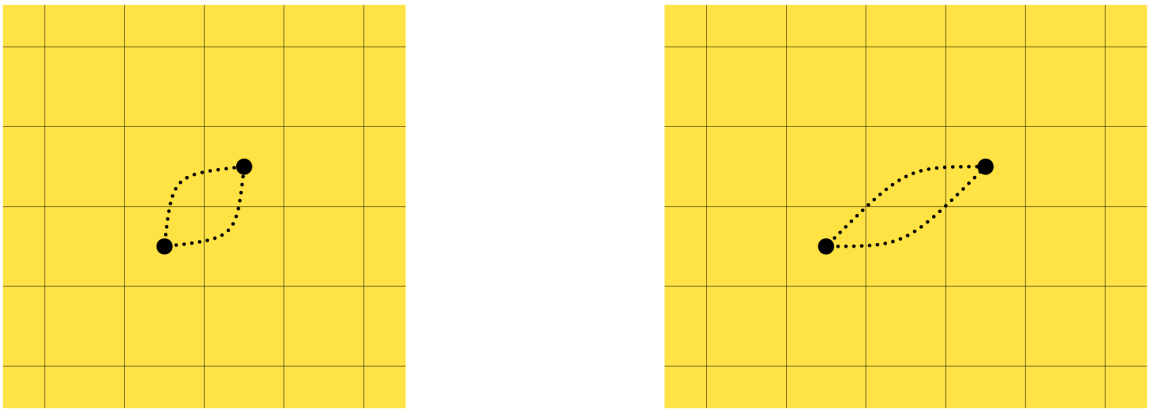


Figure 6.3 Bridges on the square grid with and without diagonal adjacencies.

For example, the bridge shown in Figure 6.3 (left) is only valid if indirect neighbours are not specified, as they would make the terminal cells adjacent. On the other hand, the bridge shown in Figure 6.3 (right) is only valid if indirect neighbours are specified.

For each bridge, the two terminal cells and the two carriers that form the bridge paths are stored.

Sites are created dynamically after each move of a boardless game and the relevant adjacency and bridge information updated.

Regions

A region object is created for each region defined in the game's ludeme tree. Note that some keywords, such as *alternating-sides*, automatically generate regions based on the current board shape.

Each region object contains the following information:

- Owner(s),
- Unique identifier tag (for move generation and end testing),
- List of references to member sites, and
- Cross references to contained bridge terminals.

Each board site also contains a return reference to all regions it belongs to. Sites may belong to any number of regions and regions may belong to any number of players.

Piece Definitions

A piece definition object is created for each piece definition in the game's ludeme tree. Each piece definition contains the following information:

- Name,
- Owner,
- Label,
- Initial value,
- Initial state,
- Initial flags, and
- Collection of move definition objects.

Each move definition contains the following information:

- Direction,
- Precondition tree,
- List of actions,
- Postcondition tree, and
- Cross reference back to the parent piece definition.

The precondition and postcondition trees are logical *condition trees* defined by the operators {*and*, *or*, *not*, *if*, *n-of*} in which all leaf nodes are either constant values or functions of the current board state. The *n-of* operator returns true if the specified number {INT, *half*, *majority*} of child nodes are true.

Start Items

The game's starting conditions are described by a collection of start item objects, each of which contains the following information:

- Type (*in-hand* or *place*),
- Piece type,

- Number of pieces,
- List of region identifiers,
- List of site indices, and
- List of direction or edge specifiers.

State Object

The single state object represents the current state of the game as it is played. Copies of the state object may be made for performing various tests without affecting the game's master state.

The state object contains the following information:

- Whose turn it is to play,
- Whether the game has been won,
- Winner (if any),
- Pieces remaining in hand for each player,
- Captured pieces,
- Players' scores, and
- Collection of site state objects.

Site States

Each site state object contains information relevant to the current state of a board cell. This includes:

- State,
- Reference to the site's defining object, and
- Piece stack object that describes the pieces currently stacked at this cell.

Pieces stacked at the cell may be accessed by index. The owner of the cell is deemed to be the owner of the topmost piece, unless otherwise specified. Each piece in the stack has a corresponding piece state object defining its current state.

Piece States

Each piece state object contains information relevant to the current state of a piece stacked at the current board cell. This includes:

- Colour,
- State,
- Flags, and
- Pointer to the piece's definition.

End Conditions

The game's end conditions are described by a collection of end tree objects, each containing the following information:

- Owner,
- Outcome, and
- Condition tree.

The condition tree in this case is a tree defined by logical operators {*and*, *or*, *not*, *if*, *n-of*} operators in which all leaf nodes are end condition objects, which contain relevant information including:

- Type (*n-in-a-row, reach, connect,* etc), and
- Options (target length, target colour, target number, target piece type, target regions, etc).

Move History

The move history of the current game is represented by a collection of move objects, each containing the following information:

- Mover,
- Piece colour,
- Pointer to the relevant move definition,
- List of actions {pop, push},
- List of postconditions (captures, conversions, promotions, etc),
- Evaluation details,
- Description of captures,
- Whether this move was made from pieces in hand, and
- Restore information for undoing the move.

6.5 User Interface

The user interface of the Ludi general game player, shown in Figure 6.4, coordinates play of the currently loaded game for up to eight players. An important design consideration was to present all games as uniformly as possible so that no game was more visually attractive than any other; any preference of one game over another should be based solely on the quality of the game itself.

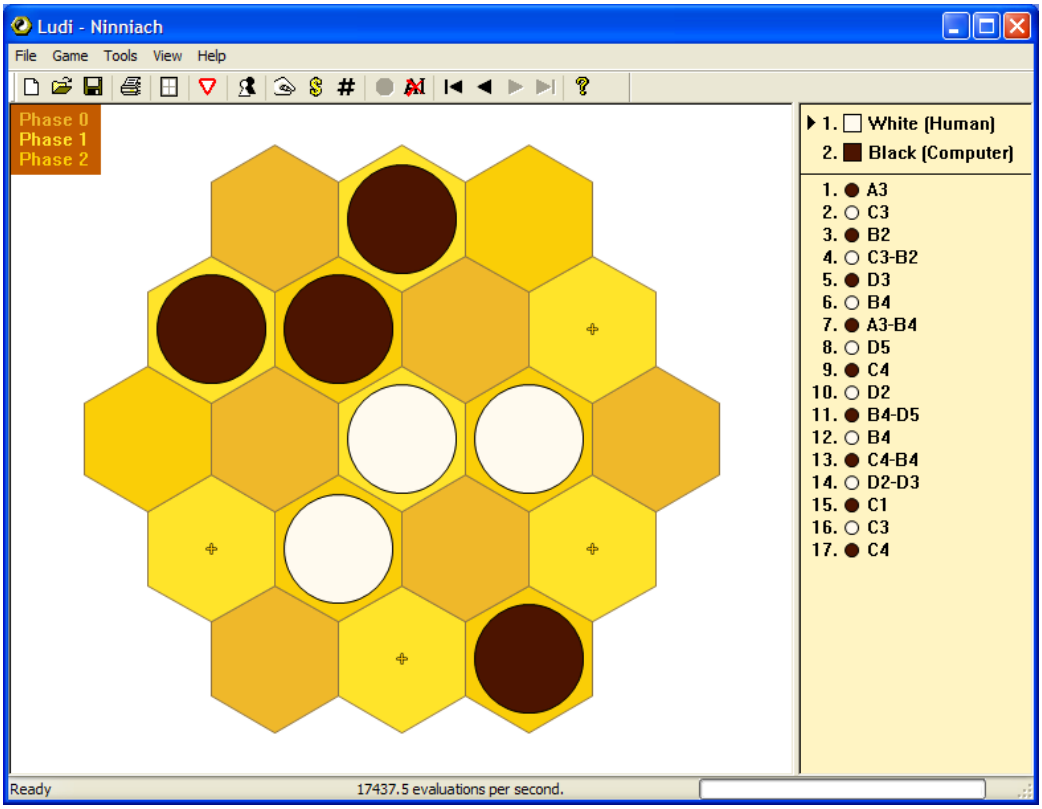


Figure 6.4 Ludi user interface in tutorial mode.

The user interface consists of a standard MFC main frame window with the following components:

- Menu system and toolbar (top),
- Main board view (left),
- Move list view (right), and
- Status panes and progress bar (bottom).

Menus and Toolbar

The menu system is a standard Windows implementation for allowing user control of all aspects of programme control and game management.

Similarly, the toolbar is a standard Windows implementation providing shortcuts for common user operations such as:

- Stepping forwards and backwards a move,
- Stepping to the start or end of the game,
- Forcing the computer player to stop the current search and make the best move they can,
- Evaluating the current board position, and so on.

Status and Progress

Status messages are displayed in standard Windows status panes (lower left and centre). These typically include helpful text messages such as whose turn it is, whether the game has been won, estimated board evaluation, number of nodes visited in the current search, and so on. The progress bar (lower right) indicates the status of pending operations, typically the search progress when planning the next computer move.

Main Board View

The main board view shows the current state of the currently loaded game. The user may specify optional display information such as board axis labels, cell coordinates, cell indices, move priorities, bridges, animation speed for piece movement, and so on.

The user indicates the move they wish to make by either:

- Clicking on a cell to place a piece,
- Clicking on an existing piece to remove it, or
- Dragging an existing piece to a different cell.

If there is any ambiguity in which exact move the user intends, for instance, if more than one type of piece may be placed at a specified cell, then the user is prompted to choose their exact move from a list of legal choices.

Move List View

The move list view includes a player status pane (upper right) and move list pane (lower right). The player status pane indicates whose turn it is to move, and allows access to player information and settings by clicking on the relevant player's name.

The move list pane shows the moves made so far in the current game, colour coded by player. The user may navigate through the game's moves by clicking on a given move to return the game to that point (later moves are greyed out).

Modes of Operation

The Ludi player operates in three basic modes:

- Tutorial mode,
- Manual mode, and
- Standard Mode.

In Tutorial mode, legal placements are marked '+', as shown in Figure 6.6, and when the user holds a mouse button down over a piece then valid destinations for a given piece are similarly marked. This was found to be extremely useful for users learning new games with arbitrarily complex rule definitions; seeing the legal placements each turn would frequently improve the user's understanding of the game and correct misunderstandings.

In Manual mode, piece placements and moves indicated by the user update the board state but do not trigger computer play. This is useful for manually setting up board positions for testing.

Standard mode is simply the absence of the other two modes. By default, the Ludi player operates in Tutorial mode.

Plain English Descriptions

In order to make a meaningful evaluation of a game, it is beneficial for players to at least understand its rules (if not its tactics and strategy) so that they can readily identify legal moves and anticipate replies from the opponent. Unfortunately, the rules in the ludeme tree form can be incomprehensible for users unfamiliar with the GDL.

For this reason, the Ludi player has a "plain English" feature that translates a given rule set into layman's terms. These descriptions were found to significantly aid the understanding of games for users not privy to the GDL definition, such as the survey participants in the Experiments.

For example, consider the compound movement rule previously given in Chapter 5:

```
(move
    (pre
        (and
            (owner from)
            (empty to)
            (line)
            (or
                (> (num-between empty) 0)
                (= (num-between enemy) 1)
             )
        )
    )
    (action (pop) (push) )
    (post
        (capture (if (>= (height current) 2) )  d-nbors)
        (inc-state)
    )
)
```

This compound rule generates the following plain English description:

> "Players may move a Stone if the source cell is owned by them, and the destination cell is empty, and the move is in a line and the number of empty cells jumped over is greater than 0 or the number of enemy pieces jumped over equals 1, following which pieces directly adjacent to the destination cell are captured if the stack height at the current cell is greater than or equal to 2 and the piece state is incremented."

It can be seen that although this description is grammatically correct both in terms of English grammar and game grammar, it is still quite daunting given that this is only one movement rule out of an entire

game description; to a user familiar with the GDL, the GDL description is often much clearer. For this reason, the user has the choice of viewing the game's rules either in plain English or ludeme tree form.

Additional Operations

The user interface also supports a number of functions to assist with game play and evaluation. These include functions to:

- Evaluate the current board position,
- Suggest the best move,
- Play a random game,
- Play an intelligent game,
- Time a number of random games, and so on.

Observing random and intelligent games (self-play games between two computer opponents) can sometimes give insight into the nature of the game. Timing random games can be useful for estimating game speed or a game's suitability for UCT move planning.

6.6 Play Manager

The play manager is the heart of the Ludi general game player and performs the primary tasks of:

- Move Handling,
- Legal Move Generation, and
- Move Planning.

6.6.1 Move Handling

Play is coordinated using a message-based approach through the application's standard Windows message queue. In response to a new game request, the application performs the following steps:

```
reset game object
reset state object
apply start items
set current player (starter)
post WM_SCHEDULE_MOVE message
```

Listing 6.1 Steps in the new game sequence.

The main frame then polls for WM_SCHEDULE_MOVE messages, and for each such move request performs the steps shown in Listing 6.2.

```
if (board display in manual mode)
{
     // manual placement – take no further action
}
else if (game over)
{
     report outcome to user
}
else if (all players passed in succession)
{
     declare a draw
}
else if (current player is computer)
{
     find the best move
     make the best move
}
else
{
     // human turn - take no further action
}
```

Listing 6.2 Pseudocode for move message handling.

Computer moves are triggered automatically, while user moves are indicated by mouse interactions with the main board view. The following actions are performed each time a move is made:

```
apply move to the state object
add move to the game's move history

if (end condition satisfied)
    set winner
else
    iterate current player to next in play order

update main board and move list views
post WM_SCHEDULE_MOVE message
```

Listing 6.3 Actions performed for each move.

The move scheduler therefore acts as a referee that arbitrates games and directs their play.

6.6.2 Legal Move Generation

The first requirement of move planning is the ability to generate a list of all legal moves for a given board position. Ludi achieves this by visiting each move definition of each piece definition, and creating a move object for each valid potential move found.

The method for identifying valid moves depends on the type of move, as follows.

- **Placement** (*action* (*push*))
  A move object is created for each board cell that satisfies the move's preconditions, typically that the cell is empty.

- **Removal** (*action* (*pop*))
  A move object is created for each piece that satisfies the move's preconditions, typically that the piece belongs to the current player.

- **Movement** (*action* (*pop*) (*push*))
  A move object is created for each board cell that satisfies the move's preconditions relative to each piece, typically that the piece belongs to the current player and the destination cell is empty.

The list of moves generated each turn is shuffled before move ordering is performed by 1-ply lookahead. This randomises play to some extent in the choice between equally valued moves.

As an aside, move objects are sizable and information-rich as currently implemented. Compacting or reorganising this information would be an obvious starting point for optimising the player for speed, as one such object is created for each of the many thousands of nodes visited when planning moves each turn.

Ko Moves

Figure 6.5 shows a situation that can be crippling for computer self-play. Consider that Black moves at position ***a*** (left) to surround and capture the White stone, resulting in the position on the right. White can then make a mirror move at ***b*** to capture the Black stone and return the board to its previous state. If the computer player values capturing moves above all others – which is very likely – then this sequence will continue until the game times out.

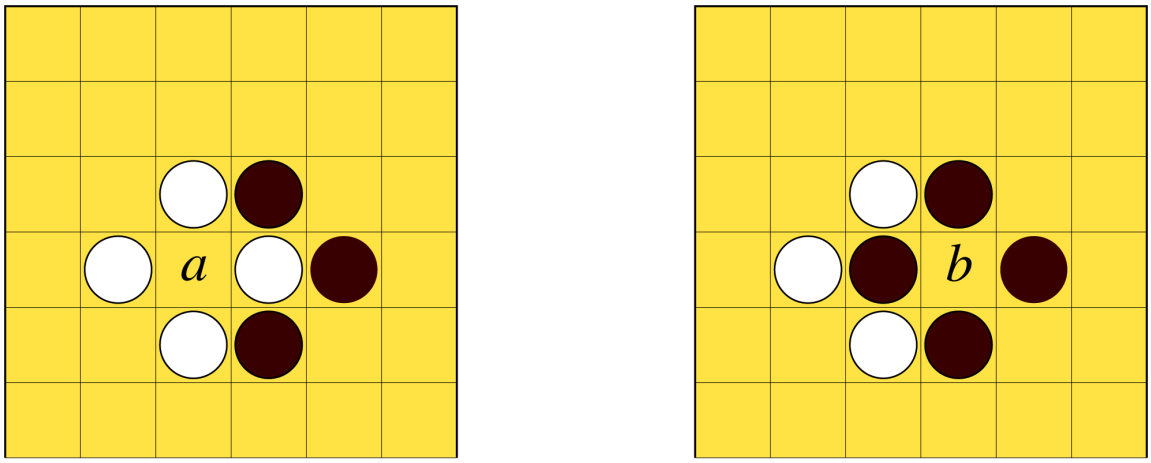


Figure 6.5 A ko situation.

Such repetitions are known as *ko* situations to Go players, and are so familiar that a specific ko rule has been created for dealing with them: players cannot make a move that would lead to a repetition of the previous board state. The Ludi player implements this rule by marking such ko moves and overlooking them during subsequent searches during self-play.

A more rigorous implementation might extend the ko rule to forbid the repetition of *any* previous board position, however this would have serious implications for speed and memory performance due to the need to compare each board position with all previous board positions. The simple ko rule described above proved sufficient for the practical purposes of this study.

6.6.3 Move Planning

Move planning is critical to the success of the general game player and to the project as a whole; if computer opponents do not make reasonable moves, then human users will not get a true feel for the game and automated self-play will not make realistic explorations of the move space.

The following steps are performed when the computer player receives a request to make a move:

```
generate legal moves

if (measuring)
    perform pre-search measurements

if (no legal moves)
    pass
else if (search depth < 0)
    choose random legal move
else if (search depth = 0)
    choose highest valued legal move
else if (search depth > 0)
    perform adversarial search

if (measuring)
    perform post-search measurements

return best move
```

Listing 6.4 Computer move planning.

Firstly, a list of legal moves available to the current player is generated. This list is shuffled, sorted and ko moves marked as described in the previous section.

If the current game is part of a quality measurement self-play trial, then a number of pre-search move measurements are performed at this point.

If no moves are available, then the computer player passes unless a *no-move* end condition indicates a win or loss. Otherwise, the computer chooses one of the legal moves to play based upon the current player's search settings.

If the current player's search depth $D < 0$ then a random legal move is returned, if $D = 0$ then the best move according to a shallow 1-ply lookahead is returned, otherwise an adversarial search is performed to depth D to determine the best move.

Adversarial Search

A standard minimax-based adversarial search is used including alpha-beta pruning, move ordering and beam width reduction, as described in Section 3.2.1. The maximum search depth and beam width are specified by the current player's search settings, and if a time limit is specified then iterative deepening is used.

The search can be interrupted by user at any point and the current best move returned. This is not guaranteed to produce a good result due to nature of minimax-style search; UCT search would be preferable in this respect, but was not implemented for reasons given below. Moves marked as ko moves are ignored during the search in order to avoid board repetitions during self-play.

For the purposes of this project, the preferred response time for computer move planning is between 10 and 30 seconds per move. This allows reasonable moves to be planned without trying the patience of human opponents and means that a large number of self-play games may be conducted in a reasonable amount of time. By way of comparison, Pell's METAGAMER [1993b] typically spent 45 minutes planning each move for Chess-like games at ply 5, using the machines of 15 years ago.

In order to detect forks (simultaneous non-overlapping threats) a search depth of at least ply 2 is required, and in order to detect potential fork threats from the opponent, a search depth of at least ply 3 is required. This is the point at which computer play generally starts to become intelligent and interesting, and provides a basic rule of thumb for implementation: *the general game player should support at least ply 3 searches for every game, and ideally higher plies for most games.*

Although not studied in this thesis, multiplayer games are supported by the Ludi player. Such games use a multiplayer minimax search that strives to maximise the current player's evaluation at each point. This method is unsophisticated but proved sufficient for testing purposes.

Throughout the search, leaf nodes are evaluated by a collection of advisors in accordance with the current player's policy, using the Strategy module. Chapter 7 gives further details of this process.

Suitability of UCT

Although the UCT method for move planning described in Section 3.2.4 has many attractive properties for general game play, it is limited by the need for fast random playouts.

The Ludi general game player achieves the notional target of 1,000 playouts per second for many games, but it is easy to devise games with complex movement rules and large branching factors which do not even come close; the general game player was designed with generality in mind and is not necessarily optimised for speed. The strength of computer play using the UCT algorithm could therefore vary widely depending on the complexity of each game.

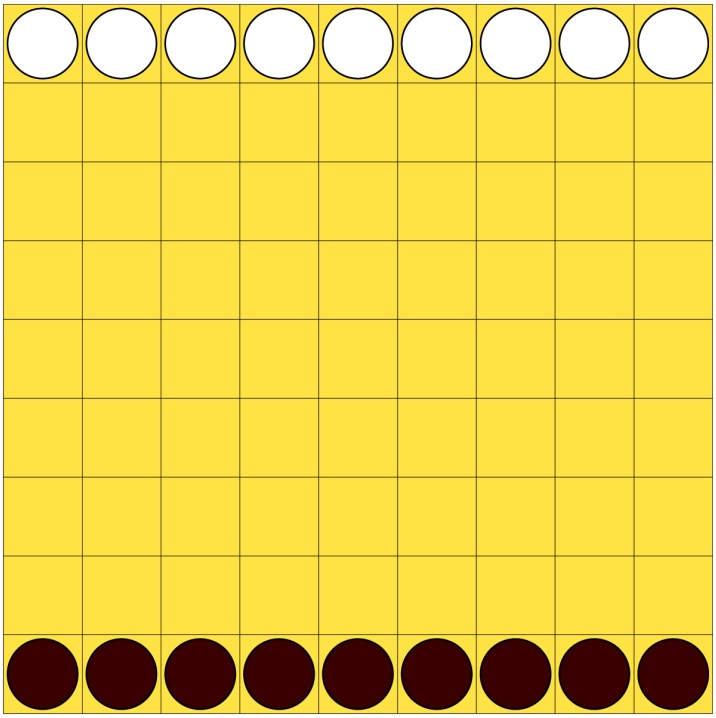


Figure 6.6 A simple game with a huge branching factor.

Consider the 9x9 game shown in Figure 6.6, in which each player starts with a line of pieces along their home row that may move to any other board cell. This game has the same board size as the 9x9 Go at which Gelly et al's UCT player excels [2006], however the branching factor in this case is an order of magnitude greater, possibly reaching 720 depending on whether capture or stacking may occur. The simple act of generating legal moves for each board position will significantly slow down random playouts in this case.

Another consideration is that the absence of domain knowledge for new games means that playouts would be entirely random, further reducing performance (although a general solution for this problem is presented in Appendix I).

UCT was therefore not implemented for this project, however it is noted as a topic of extreme interest for the development of future general game players.

6.7 Summary

The Ludi general game player interprets games defined in the Ludi GDL and allows the user to:

- Play the game against a computer opponent,
- Measure the game based on some aesthetic criteria, and
- Create new games by evolving an existing set of games.

When loading the rules for a game defined in a *.gdl file, the Ludi player expands the rules into an internal ludeme tree and initialises the appropriate data structures, most importantly the:

- Board object,
- Player objects,
- Piece definitions (including move definitions),
- Start items,
- End condition trees, and
- The board state object.

Stringent rule safety checks are performed at this point.

The Ludi player uses a standard Windows message-based approach to coordinate play. For each computer move, it invokes a standard adversarial search relying on the Strategy module to provide board evaluations. The play coordinator:

- Forces players to pass if they have no move,
- Avoids repeating the board position at the player's previous move, and
- Concludes the game if all players pass in succession.

In addition to a "plain English" feature for displaying a game's rule set in layman's terms, the understanding of new games is facilitated by a tutorial mode which indicates legal moves each turn.

Chapter 7

Strategy Module

> You have to learn the rules of the game. And then you have to play better than anyone else.
> – Albert Einstein
>
> If the student forces himself to examine all moves that smite, however absurd they may look at first glance, he is on the way to becoming a master of tactics.
> – Purdy

7.1 Introduction

The Strategy module handles the strategic aspects of game play, in particular the management of the board state advisors and the policies that decide how these advisors are applied. This chapter describes the operation and implementation of the advisors and policies. Some strategies that emerged during self-play policy optimisation are presented.

The key functions of the Strategy module include:

- Board state evaluation,
- Outcome detection,
- Providing default policies for games based upon their rules, and
- Policy optimisation by competitive learning.

7.2 Model

The core of the Strategy module is a bank of advisors, similar to those described by Epstein [1989] and Pell [1993b] but with the significant difference that they perform two functions:

- Board state evaluation, and
- Outcome detection.

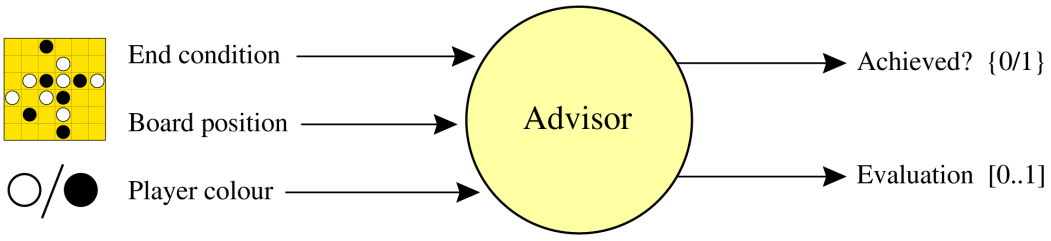


Figure 7.1 The advisor model.

Figure 7.1 shows the basic operation of each advisor. Given a player colour, board position and end condition, each advisor optionally returns a boolean value indicating whether the specified end condition has been achieved for that player, that is, whether the game has been won, lost or drawn. This test is only made for advisors known to be relevant to the specified end condition.

In addition, each advisor may also return a floating point value in the range [0..1] indicating its evaluation for the specified player for this board state relative to the given end condition. This value is

essentially a *distance to goal* (DTG) estimate indicating how close the mover is to achieving the end condition for this board position, except that 0 indicates a poor (losing) position and 1 indicates a good (winning) position.

If a loss is detected then 0 is returned and if a win is detected then 1 is returned. If a guaranteed win is detected, but not an actual win, then a value of 0.999 is returned so that guaranteed wins will have priority over other moves that are not actual wins.

The overall board evaluation is given by the weighted sum of all non-zero advisor evaluations, normalised to the range [0..1]. This set of weights $\{w_1, w_2, w_3 \ldots w_n\}$ constitutes the given player's policy for that game, as shown in Figure 7.2.

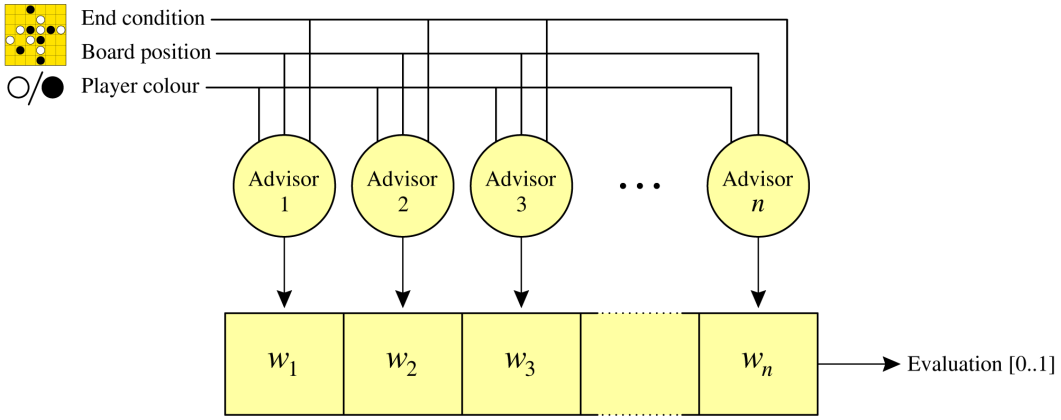


Figure 7.2 The policy model.

The following sections describe the implementation of the advisors and their related policies in detail.

7.3 Advisors

Ludi's advisor-based method of board evaluation is similar to methods outlined by other researchers such as Epstein [1989] and Pell [1993b]. However, the scope of games covered in this thesis is somewhat broader than those of previous studies, hence the advisors are not optimised to particular game types but rather have robust applicability to a wider range of possible games. Some compromises had to be made, and in most implementation choices between speed and generality, generality was preferred.

Each advisor is implemented as a child class derived from a virtual advisor base class. The doubling of duties as board evaluators and outcome detectors means that code common to both tasks may be stored in the same objects and used more efficiently.

Pell [1993b] distinguishes between *global* and *local* advisors, and between *static* and *dynamic* advisors. The distinction between local and global operation was not made in the Ludi advisors; each advisor simply operated in a local or global sense (or both) as appropriate. However, there is a clear distinction between static and dynamic advisors in the Ludi system:

- *Static*: Initialised once when the game is loaded.
- *Dynamic*: Recalculated for every board position.

7.3.1 Static Advisors

Static advisors involve records that are created once, immediately after a game's ludeme tree and board object are assembled. These records are used for quick lookup for subsequent board states.

The static advisors are generally concerned with the minimum or average distance between the pieces in the current board state and particular features of the board. A *manhattan* or *city-block* distance metric [Gonzalez & Woods, 2008] of one unit per adjacent step is used rather than Euclidean distance. For example, Figure 7.3 shows distances from the central cell for square grids with and without diagonal adjacencies.

3	3	3	3	3	3	3
3	2	2	2	2	2	3
3	2	1	1	1	2	3
3	2	1	0	1	2	3
3	2	1	1	1	2	3
3	2	2	2	2	2	3
3	3	3	3	3	3	3

6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	0	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6

Figure 7.3 Distance metrics for square grids with and without diagonal adjacencies.

Distance values are normalised to the range [0..1] based on the maximum distance found for that measurement. Note that the distances may differ significantly depending on whether indirect neighbours are specified or not.

The following distance tables are created upon board construction:

- Distance to corners ($dist_c$),
- Distance to sides ($dist_s$),
- Distance to middle ($dist_m$),
- Distance to home relative to each player ($dist_h$), and
- Distance to each end condition goal ($dist_g$).

The static advisors are listed below.

Proximity to Corners

For a given board position, this advisor returns 1 minus the average distance of each piece belonging to the mover from the closest board corner. This advisor encourages players to move towards the corners.

$$Adv_{proxc} = 1 - \sum_{p_m=1}^{P_m} dist_c[site(p_m)] / P_m$$

where P_m is the total number of pieces belonging to the mover and $site(p_m)$ returns the index of the cell containing mover's piece p_m.

This advisor's weighting is denoted by the element (*proxc* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Proximity to Sides

For a given board position, this advisor returns 1 minus the average distance of each piece belonging to the mover from the closest side of the board. This advisor encourages players to move outwards towards the sides.

$$Adv_{proxs} = 1 - \sum_{p_m=1}^{P_m} dist_s[site(p_m)] / P_m$$

This advisor's weighting is denoted by the element (*proxs* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Proximity to Middle

For a given board position, this advisor returns 1 minus the average distance of each piece belonging to the mover from the middle of the board. This advisor encourages players to move inwards towards the middle of the board.

$$Adv_{proxm} = 1 - \sum_{p_m=1}^{P_m} dist_m[site(p_m)] / P_m$$

This advisor is largely redundant due to the proximity to middle advisor, however it can impart subtly different information for some board shapes.

This advisor's weighting is denoted by the element (*proxm* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Proximity to Home

For a given board position, this advisor returns 1 minus the average distance of each piece belonging to the mover from their home edge. This advisor encourages players to stay near their home area; this may be useful for defending a home area, or may be negated to move pieces away.

$$Adv_{proxh} = 1 - \sum_{p_m=1}^{P_m} dist_h[site(p_m)] / P_m$$

This advisor's weighting is denoted by the element (*proxh* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Proximity to Goal

For a given board position, this advisor returns 1 minus the estimated distance between the specified player's pieces and the specified goal. This will be the average squared distance of the closest *N* closest pieces to the goal, where *N* is the number of pieces that that must reach the goal. This advisor encourages players to move towards the goal.

$$Adv_{proxg} = 1 - \sum_{p_m=1}^{N} dist_g[site(p_m)]^2 / N$$

where p_m iterates over the mover's N closest pieces to the goal. Squaring the distance encourages those pieces closest to the goal to accelerate towards it.

This advisor's weighting is denoted by the element (*proxg* FLOAT) in the Ludi GDL. It applies to the reach a goal end condition and indicates when the specified goal has been achieved.

7.3.2 Dynamic Advisors

Dynamic advisors are calculated on-the-fly for each given board position. These are generally more computationally expensive and slower than the static advisors.

Attack

For a given board position, this advisor returns the total value of enemy pieces directly threatened by the mover's pieces, divided by the total value of enemy pieces on the board. This advisor encourages players to attack enemy pieces.

$$Adv_{attack} = \frac{\sum_{m=1}^{M} site(push_m) = site(p_o) \begin{cases} value(p_o) \\ 0 \end{cases}}{\sum_{p_o}^{P_o} value(p_o)}$$

where M is the total number of moves available to the current mover, $site(push_m)$ is the site to which move m pushes a piece, p_o is an opponent's piece, $value(p_o)$ is the value of the opponent's piece and P_o is the total number of opponent's pieces on the board.

This advisor's weighting is denoted by the element (*attack* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Defence

For a given board position, this advisor returns the total value of friendly pieces defended by the mover's pieces, divided by the total value of friendly pieces on the board. This advisor encourages players to defend their own pieces.

$$Adv_{defen} = \frac{\sum_{m=1}^{M} site(push_m) = site(p_m) \begin{cases} value(p_m) \\ 0 \end{cases}}{\sum_{p_m=1}^{P_m} value(p_m)}$$

This advisor is denoted by the element (*defen* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Threat

For a given board position, this advisor returns the total value of friendly pieces directly threatened by enemy piece, divided by the total value of friendly pieces on the board. When negated, this advisor encourages players to avoid threats from enemy pieces.

$$Adv_{threat} = \frac{\sum_{p_m=1}^{P_m} threatens(p_o, p_m) \begin{cases} value(p_m) \\ 0 \end{cases}}{\sum_{p_m=1}^{P_m} value(p_m)}$$

where *threatens*(p_o, p_m) returns true if any legal move from p_o threatens p_m.

Note that this advisor is significantly slower than the attack and defence advisors as it requires additional move calculations beyond the mover's current set of moves; it effectively adds one ply to the search depth.

This advisor's weighting is denoted by the element (*threat* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Material

For a given board position, this advisor returns the total value of the mover's pieces on the board divided by the mover's total possible piece value. This advisor encourages players to capture enemy pieces, and can also be also useful in non-capture games for encouraging players to add new pieces rather than move existing ones.

$$Adv_{mater} = \frac{\sum_{p_m=1}^{P_m} value(p_m)}{\max\_total_m}$$

where max_total_m is the total possible value of pieces belonging to the specified mover, given by the sum of the total value of their pieces placed in the starting position plus the total value of their pieces that may potentially be placed on the board.

This advisor is denoted by the element (*mater* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Capture

For a given board position, this advisor returns 1 minus the total value of target pieces on the board divided by the mover's total possible piece value. The capture advisor is distinct from the material advisor as it only counts the target piece(s) specified in the end condition. This advisor encourages players to reduce the number of target pieces.

$$Adv_{capt} = 1 - \frac{\sum_{p_t=1}^{P_t} value(p_t)}{\max\_total_t}$$

where P_t is the total number of target pieces p_t on the board and max_total_t is the total value of target pieces.

Note that target pieces will generally belong to the each player's opponent rather than the players themselves.

This advisor's weighting is denoted by the element (*capt* FLOAT) in the Ludi GDL. It applies to the capture end condition and indicates when the target piece(s) have been captured.

Mobility

For a given board position, this advisor returns a measure of the number of legal moves M available to the specified mover. This advisor encourages players to maximise their movement options.

$$Adv_{mobil} = \min\left(1, \log_{10}\left(M + 1\right) / 2\right)$$

The $\log_{10}$ component of the calculation is divided by 2 to set a range limit of approximately 100 moves.

Ideally this advisor would return a measure of the number of *reasonable* moves available to the specified mover, however such a measure would be significantly more expensive to calculate.

This advisor is denoted by the element (*mobil* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Influence

For a given board position, this advisor returns the total number of board cells reachable by any move available to the specified mover divided by the total number of board cells (each reachable cell is only counted once). This advisor encourages players to maximise their movement options.

$$Adv_{infl} = \frac{count\left(\underset{1 \le m \le M}{site(push_m)}\right)}{total\_cells}$$

This advisor is denoted by the element (*infl* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Score

For a given board position, this advisor returns the specified mover's score divided by the total possible score. This advisor encourages players to maximise their score.

$$Adv_{score} = score(m) / \max\_score$$

This advisor's weighting is denoted by the element (*score* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none (the *score* end condition was not implemented).

Proximity to Enemy

For a given board position, this advisor returns 1 minus the average distance from each friendly piece to the closest enemy piece divided by the board size. This advisor encourages players to surround or otherwise engage with enemy pieces.

$$Adv_{proxe} = 1 - \frac{\sum_{p_m=1}^{P_m} dist\_to\_enemy(p_m)}{P_m \dim}$$

Unlike the static proximity advisors, proximity to enemy is entirely dependent on the specified board state and must be determined dynamically.

This advisor's weighting is denoted by the element (*proxe* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Stack

For a given board position, this advisor returns a measure of the specified player's potential for stack height. This advisor encourages players to either achieve a target height or to generally build higher stacks where possible. Alternately, this advisor can be negated to discourage detrimental stacking behaviour in a game.

For each stack belonging to the specified player, a value is stored depending on whether the specified end condition is of type *stack* or not:

- If so, then the stack value is the stack height divided by the target height.
- If not, then the stack value is $\log_{10}$ its height, which encourages quicker growth of shorter stacks and does not encourage growth after height 10 is reached.

The individual stack values are then combined as a union of probabilities to give a single resulting estimate:

$$Adv_{stack} = P(\bigcup_{i=1}^{n} S_i)$$

where S denotes the collection of stack probabilities and the union of probabilities is given by [Weisstein 1999]:

$$P(\bigcup_{i=1}^{n} S_i) = \sum_{i} P(S_i) - \sum_{ij}{}' P(S_i \cap S_j) + \sum_{ijk}{}'' P(S_i \cap S_j \cap S_k) - \ldots + (-1)^{n-1} P(\bigcap_{i=1}^{n} S_i)$$

This advisor's weighting is denoted by the element (*stack* FLOAT) in the Ludi GDL. It applies to every end condition and indicates when the target stack height has been achieved.

***N*-in-a-Row**

For a given board position, this advisor returns a measure of the specified player's potential for forming lines of pieces. This advisor encourages players to either achieve lines of a target length or to generally build longer lines where possible.

Firstly, a target line length is determined. If the end condition is of type *n-in-a-row* then the specified line length is used, else the target line length is set to half of the maximum board dimension.

For each piece belonging to the specified player, the total number of potential lines of the required length that pass through that piece and are not blocked by enemy pieces are found. For each such potential line, a value is stored indicating the total number of pieces in that line.

The potential line values indicate how close each potential line is to forming an actual line of the required length. As such, the value is halved if an opponent's piece blocks one end and halved again if another opponent's piece blocks the other end.

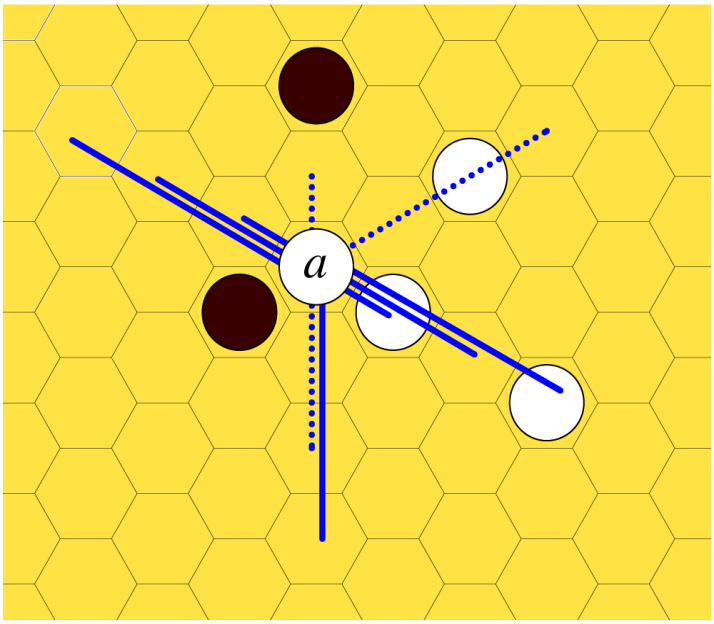


Figure 7.4 Potential 4-in-a-row lines through piece ***a***.

For example, Figure 7.4 shows the seven potential lines of length 4 through White piece ***a***. The two dotted lines indicate potential lines blocked by a Black piece at one end.

The potential line values for piece ***a*** in this example are $\{1/4, 2/4, 2/4, 3/4, 1/8, 1/4, 2/8\}$.

Potential line values are similarly calculated over all pieces belonging to the specified player, and the resulting value for the advisor is the union of these values calculated as a union of line probabilities L_i. Potential lines passing through more than one piece are only counted once.

$$Adv_{nina} = P(\bigcup_{i=1}^{n} L_i)$$

An alternative to combining these values as a union of probabilities is to simply take the average of the best two values, which takes into account the fact that the opponent will most likely block the player's highest-valued potential line next turn. This method proved reasonably successful but was found to lead to weaker play than the union of probabilities.

The *N*-in-a-row advisor is probably the most successful of all of the advisors. It is fast to calculate and allows strong computer play even at low search plies, and even in the absence of all other advisors.

It might appear that the evaluation could be further improved by only counting the maximum line through each playable cell. However, counting multiple lines through each cell gives a more accurate reflection of their relative importance.

This advisor's weighting is denoted by the element (*nina* FLOAT) in the Ludi GDL. It applies to every end condition and indicates when the target line length has been achieved.

Line Cap

For a given board position, this advisor returns an estimate of the specified mover's potential to cap a line of enemy pieces at both ends. This advisor encourages players to block enemy lines and increase the chance of linecap captures or conversions in the appropriate games.

For each friendly piece, adjacent runs of consecutive enemy pieces are counted and stored. This gives a value *cap_runs* which is the accumulated length of enemy runs capped at one end. The advisors resulting value is then calculated as follows:

$$Adv_{linecap} = \min\left(1, \log_{10}\left(cap\_runs + 1\right) / 1.4\right)$$

The $\log_{10}$ component of the calculation is divided by 1.4 to set a range limit of approximately 25 enemy pieces capped at one end.

This advisor's weighting is denoted by the element (*linecap* FLOAT) in the Ludi GDL. It applies to every end condition but achieves none.

Group

For a given board position, this advisor returns an estimate of the specified mover's potential for forming either a group of the specified size or a group of all friendly pieces. This advisor encourages players to gather their pieces into connected groups, and largely removes the need for a specific proximity to friend advisor.

Firstly, a target group size is determined. The target group size is assumed to be all pieces unless the specified end condition is of type *group* and specifies a different target group size.

All connected groups of friendly pieces are then found, the size of each determined and the maximum group size found. For each group, the distance to every non-group board cell is determined and these distance values squared and accumulated over all groups for each cell. The minimum total squared distance *min_total* is then found over all cells and a measure of the least accumulated distance given by:

$$closest = \sqrt{\min\_total / num\_groups}$$

where *closest* indicates the average distance to all groups from the cell that is on average closest to each. If the aim is to group all pieces together, then the advisor's resulting value is:

$$Adv_{group} = 1 - \min\left(1, \log_{10}\left(num\_groups + c \times closest\right)\right)$$

where *c* is a constant bias factor (*c*=0.1 provides good results). Otherwise, if the aim is to form a group of a specific size then the advisor's resulting value is:

$$Adv_{group} = \min\left(1, \log_{10}\left(\max\_group\_size + c \times closest\right)\right)$$

The first form rewards fewer groups and the second form rewards larger groups. Both forms reward disconnected groups being closer together due to the *least* component.

This advisor's weighting is denoted by the element (*group* FLOAT) in the Ludi GDL. It applies to every end condition and indicates when the mover has achieved a single group of their pieces.

Connection

For a given board position, this advisor returns an estimate of the potential for the specified mover to connect R target regions with a path of friendly pieces. This advisor encourages players to strengthen their connection in accordance with a golden rule of connection games: *your position is only as good as the weakest link in your best connection* [Browne, 2000].

Firstly, the maximum board dimension *dim* is used as a yardstick for normalising distance measures. If two regions r_0 and r_1 are to be connected, such as in the game of Hex, then the following algorithm is used.

Starting at region r_0, a forward distance sweep is made using a graph distance metric that marks 1 step between adjacent cells and bridge terminals with empty carriers. Steps to friendly pieces involve 0 distance (short) and steps to enemy pieces are forbidden (cut). At the same time, the current gap size (number of steps since the last piece) and the accumulated gap size squared from r_0 are stored for each cell.

All cells within destination region r_1 are then visited, and the best cell r_{1i} that minimises distance plus accumulated gap multiplied by a bias factor c is found (c = 0.25 gives good results). This best cell's value gives a measure of the step distance between the target regions, incorporating virtual bridge connections and the maximum gap size between pieces along the best path:

$$Adv_{conn} = 1 - \min\left(1, \min_{1 \le i \le size(r_1)} \left(dist(r_{1i}) + c \times gap\_acc(r_{1i})\right) / \dim\right)$$

This is a slight simplification, as steps between bridge terminals attract a gap penalty of 1.001 while steps between adjacent cells attract a gap penalty of 1. This has the effect of preferring adjacent steps over bridge steps, but not at the expense of connective distance.

This algorithm would not compete against a dedicated Hex algorithm such as one that constructs complete virtual connection trees between target regions [Anshelevich, 2002]. However, it is fast, makes reasonable moves in most cases, and the incorporation of bridges goes some way to imparting to the algorithm a small understanding of virtual connections. Detecting *edge templates* that guarantee a piece's connection to a target region [Browne, 2000] would improve the player's strength, however these would require special care as the board tiling and shape may be arbitrary in the case of general games.

If more than two regions are to be connected, such as in the game of Y, then a different approach must be taken. Firstly, the Queenbee distance (QBD) metric is defined; this is a modification of the standard graph distance metric which uses the second-best neighbour at each step rather than the best. This incorporates the knowledge that the best neighbours are not reliable as the opponent has the opportunity to block one on their next move [Van Rijswijck, 2002].

The algorithm for R>2 proceeds as follows: for each target region r_t the Queenbee distance is calculated to all other cells, these values are then squared and accumulated for each cell and the *least* valued cell found. The advisor's final result is then given by:

$$Adv_{conn} = 1 - \min\left(1, \log_{10}\left(least / \dim / R + 1\right)\right)$$

The connection calculations indicate high priority cells along players' best paths as a side-effect; it is a shame to discard this information. This was the main motivation behind attempting to prioritise cells during board evaluation for the purposes of move ordering.

This advisor's weighting is denoted by the element (*conn* FLOAT) in the Ludi GDL. It only applies to the *connect* end condition and indicates when the specified target regions are connected.

7.3.3 Potential Advisors

This set of advisors is by no means exhaustive, but proved sufficient for the games studied in this thesis to generally develop meaningful strategies. Other advisors might include the following.

Cycle

Detecting cycles of pieces is straightforward, however measuring distance to cycle completion is difficult problem to estimate efficiently. This is the reason for leaving out this advisor and not supporting games with a cycle end condition.

Prefer Direct Line

In some games, it may be preferable to place pieces along lines of direct adjacency rather than indirect adjacency, or vice versa. This advisor was not implemented, however some games learnt to incorporate this information in an unexpected way, as described shortly in Section 9.6.2.

Territory

A measure of a player's territory could be valuable information, but also difficult to implement efficiently.

Enclosure

Similar to territory, it might be useful to determine for each piece the size of the enclosed area in which it lies. This should give some indication of piece's potential for movement, but is again difficult to implement efficiently.

Move Into Space

It may be beneficial to encourage pieces to move towards empty board areas. This would have immediate application to pursuit games such as Fox & Geese, but may be too specialised and may discourage the engagement of enemy pieces.

7.4 Policies

A *policy* is a linear weight vector that indicates the relative importance of each advisor to the current. These are stored in the *advisors* element of each game's ludeme tree for convenience, for example:

```
(advisors
        (reach 10)
        (mobil 1)
        (stack -2)
)
```

Advisor weightings are floating point values but are generally represented by integers, for ease of comprehension and manual editing. Advisors not listed in the policy are assumed to have zero weight.

During board evaluation, each advisor with a non-zero weight is applied to the current board position for the current player. The overall value is given by the sum of advisors values multiplied by their relative contribution. For instance, the contribution of the advisors described by this policy would be $\{10/13, 1/13, -2/13\}$. Each advisor returns a value in the range [0..1] and the overall board evaluation will be in the range [0..1].

Advisors with zero weight are not applied during board evaluation, so in the interests of efficiency it is preferable that each policy contain only those advisors that are truly useful for that game. Policies are not used for win detection; this is handled by the Ludi player's move planning algorithm.

Policies use linear weight vectors rather than more elaborate nonlinear rating schemes, such as those proposed by Böhm et al [2005], as many of the advisors themselves apply nonlinear measurements. Linear weight vectors proved sufficient for competent play in most cases so the investigation of more complex weighting schemes was not warranted.

Similarly, the implementation of dynamic policies that change over the course of each game [Higashiuchi & Grimbergen, 2005] were not investigated. For instance, a Chess player with only a King and a pawn remaining may appreciate advice on mobility and promotion more than advice on attacking enemy pieces.

7.4.1 The Single Policy Policy

The Ludi player maintains a separate policy for each opponent to allow comparisons between different styles of play during game measurement. The policy is applied to each end condition specific to each opponent.

However, only one policy is stored in the ludeme tree for each game, which represents the best known policy for that game for all players. In the case of players with different goals, this single policy is applied in different ways depending on the goals specific to each player; policies for such games are overloaded with strategies for all players. This is an efficient way to handle policy information for a game (it halves the required information) but will obviously have some detrimental effect on play for games with unequal goals.

Policies were implemented this way in order to make automated policy optimisation as fast as possible for most games, so that large numbers of games could be evolved in a relatively short amount of time. The results proved sufficient for the specific tasks of this thesis in that the computer player generally learnt to make reasonable moves, however the implementation of distinct *advisors* elements for each player would be an obvious starting point for improving the standard of computer play in future.

If there were unlimited time available for policy optimisation through self-play, even stronger play may be obtained by going a step further and maintaining and specifying a separate policy for each end condition for each player. This would potentially produce dozens of policies for each player depending on rule complexity.

In addition, games with separate policies maintained for unequal goals run the risk of one policy significantly outperforming the other, incorrectly suggesting a bias in the game.

7.4.2 Default Policies

The Ludi player is able to generate a default policy for a given game, provided that at least one end condition is specified for at least one player (which should always be the case). This is useful if the game's *.gdl file does not include a recommended policy, for instance, if the game has just been automatically created.

Advisors are categorised by importance relative to the end conditions and weighted as follows:

- Critical (x 10)
- Useful (x 2)
- Peripheral (x 1)
- Irrelevant (x 0)

The classification of each advisor is based on the game's rules: end conditions, movement rules, possibility of capture, possibility of stacking, and so on. Values for advisors related to detrimental measurements, such as the threat advisor, are negated.

In the interests of speed, minimal default policies may be requested which do not include useful and peripheral advisors. All default policies include *proxe*, *mobil* and *infl* advisors with a weighting of 1 to encourage players to engage with enemy pieces while leaving themselves room to manoeuvre and escape. This constitutes a "fight or flight" response in the absence of any better policy.

A default anti-stacking mechanism (*stack* -2) is also included for games in which pieces may stack. The reasons for this are explained shortly.

7.4.3 The NULL Policy

If a player has no specified end conditions, then a NULL "fight or flight" policy is used by default:

```
(advisors
    (proxe 1)
    (mobil 1)
    (infl 1)
)
```

This may occur if the player's goals are implied by the opponent's conditions rather than stated explicitly, as is the case with the following Fox & Geese variant:

```
(end
    (White win (reach away) )
    (White lose (no-move) )
)
```

The starting position for this game is shown in Figure 7.5 (left) and the swarming behaviour resulting from Black's NULL policy is shown on the right.

It would be a simple matter in some cases to detect such implied wins and craft more appropriate policies. However, consider the following case of a typical connection game:

```
(end
    (White win (connect own-regions) )
)
```

If this game is played on a hexagonal (trivalent) tiling with alternating regions, then a Black win can be implied by the absence of a White win, as the game's goals are complementary. If the game is played on a square (non-trivalent) tiling, however, such a Black win cannot be implied.

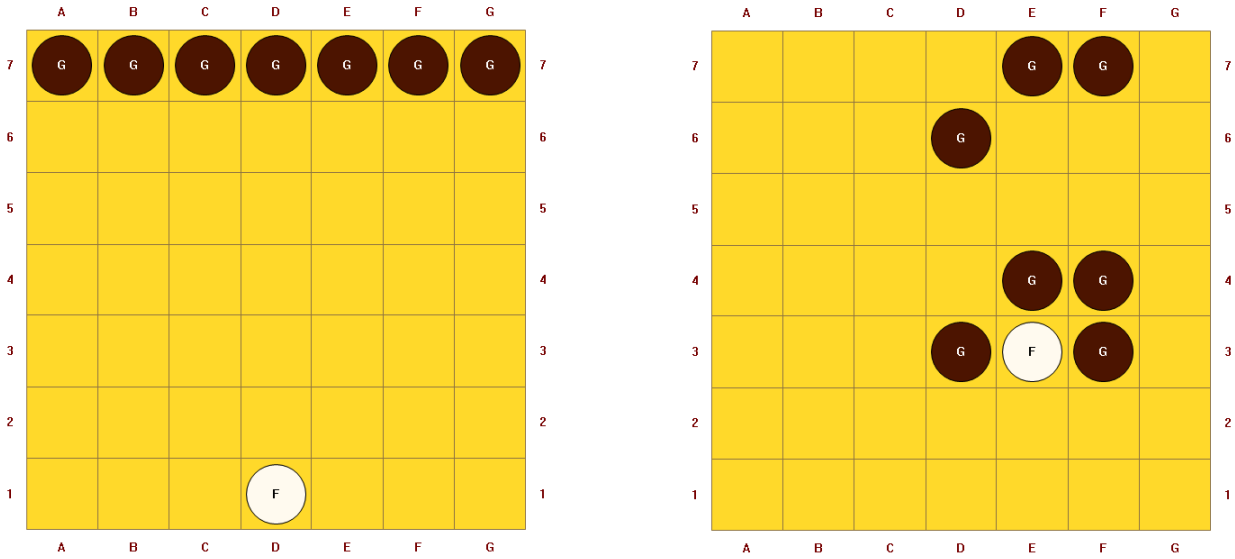


Figure 7.5 Starting position and end game showing swarming.

In addition, pathologies in the rules may prevent the conditions leading to an implied win ever being met. In general, it is not safe to assume that implied wins can be readily detected. As a rule of thumb, it is generally best to explicitly state player's end conditions and avoid use of the NULL policy where possible.

7.5 Policy Optimisation

Automatic policy optimisation is performed by self-play using a *two-membered evolutionary strategy* (1+1)-ES [Bäck et al, 1991] in which the current policy (the champion) is matched against a succession of alternative policies (the challengers). If any challenger significantly outperforms the champion by winning at least two thirds of the games between them, then the challenger becomes the new champion. The eventual champion of this tournament of policies becomes the game's new policy.

The number of matches (default 100) and number of games per trial (default 20) may be set by the user, as may search settings for the self-play tests. The optimisation process may be started from scratch using the game's default policy, or preferably started with a pair of policies known to be good.

For A_{nz} non-zero advisors in the champion policy, the first A_{nz} challengers are created by copying the champion policy and negating one advisor in turn, the assumption being that the presence of an advisor indicates that it should have at least some relevance to the game. The next A_{nz} challengers are created by copying the champion policy and removing each advisor in turn, as it is preferable to find smaller policies where possible.

The challengers for the remaining runs are created by copying the champion and randomly performing the following operations on each advisor in turn with the likelihood indicated:

- If the advisor has zero weight, activate it with a random weight (1%),
- Else remove the non-zero advisor by setting its weight to zero (2%),
- Else randomly adjust the advisor weight by a value drawn from a normal distribution.

This simple process led to policy improvement for most games within a reasonable amount of time (typically within 30 minutes on a standard Intel 1.8 GHz desktop machine). A number of games failed to find policies significantly stronger than their default polices; in such cases advisor weights were manually edited based on common sense and observation in order to improve play. Several of the default policies could not be improved upon, indicating their general robustness.

7.5.2 Emergent Strategies

Following policy optimisation by self-play, a number of automatically learnt tactics and strategies emerged. These took the form of unexpected non-zero weightings for advisors not obviously relevant to those games, but which improved play in some respect. A number of these emergent tactics and strategies are now discussed.

Anti-Stacking Safeguard

The most important strategy to emerge from the policy optimisation process was an anti-stacking mechanism that prevents the computer player from getting stuck in detrimental stack-building cycles. The need for this safeguard is demonstrated in the following game in which:

- Pieces may be placed on an empty cell or moved to an adjacent cell (possibly stacking), and
- The aim is to form a line of 3-in-a-row.

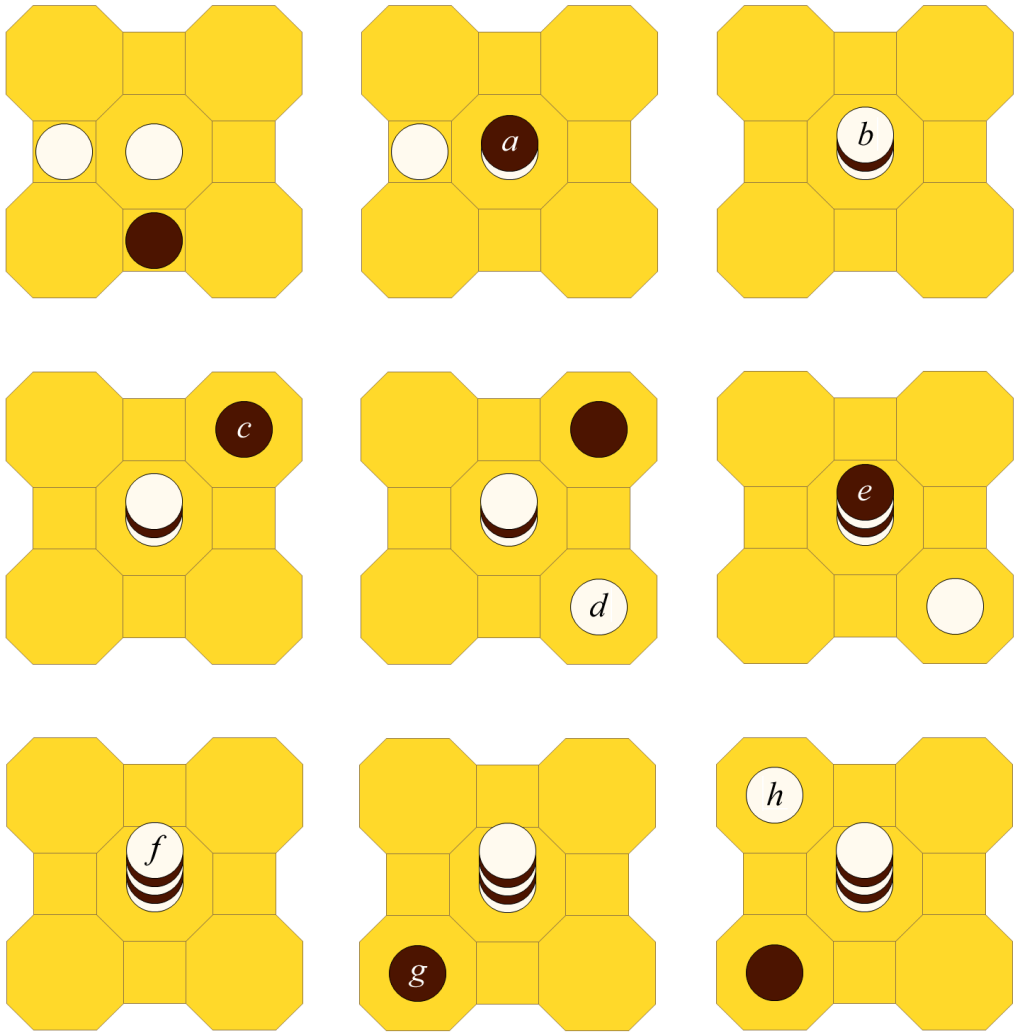


Figure 7.6 A case of detrimental stacking (Black to play).

Figure 7.6 (top left) shows a typical board position with Black to move. Black blocks the immediate loss with stacking move ***a*** then White replies with another stacking move ***b***. Move ***b*** is understandable as it removes all Black pieces from play while taking the crucial central cell through which pass the maximum number of potential lines. However, this tendency to stack soon proves to be overly defensive and rather short-sighted.

Moves ***c*** and ***d*** involve both players adding pieces to the board in optimal positions while replying moves ***e*** and ***f*** are again defensive stacking moves. The problem becomes evident with moves ***g*** and ***h*** in which both players once again add pieces to the board in optimal positions but which will be immediately followed by two more stacking moves; both players fail to develop their position while the central stack continues to grow.

The game has entered a cycle of knee-jerk defence which will continue until the move limit is reached and the game abandoned without producing a winner. Note that this does not constitute a ko cycle as the board position is unique for each turn (the central stack will continue to grow regardless of which rotation the surrounding pieces are played in) even though the same basic relationship is being repeated each cycle.

The anti-stacking mechanism takes the form small negative *stack* advisor weight for games in which stacking can occur but is not a winning condition, such as the game described above (if stacking is a winning condition then a large positive *stack* value is expected). This encourages the computer player to explore alternative lines of play that may not seem immediately beneficial but may ultimately prove more fruitful.

After its emergence, this anti-stacking mechanism was added by default to all games in which stacking could occur but was not a winning condition. It was found to be a crucial addition that stopped some games from being incorrectly discarded as overly drawish.

Central Cell Preference

A small positive weight for the *proxm* (proximity to middle) advisor was found to occur for Tic Tac Toe. This embodies a strategy specific to this particular game; in the absence of an impending fork threat, the optimum play is to take the central cell (Figure 7.7).

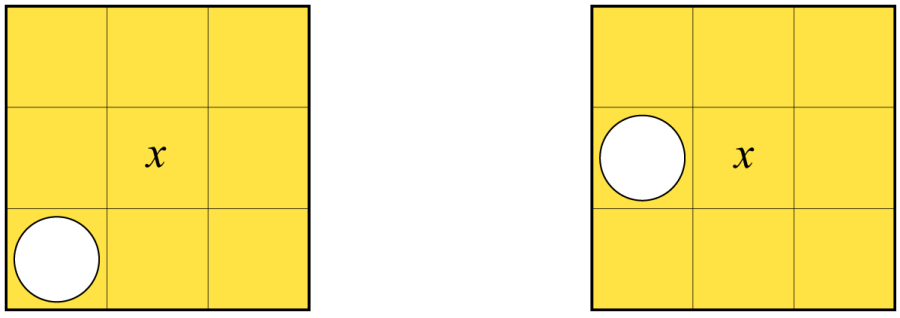


Figure 7.7 Central cell preference.

Backs Against the Wall

Similarly, a small positive weight for the *proxs* (proximity to sides) advisor was found to occur for one particular game in which pieces may leapfrog over single enemy pieces to capture them, as shown in Figure 7.8.

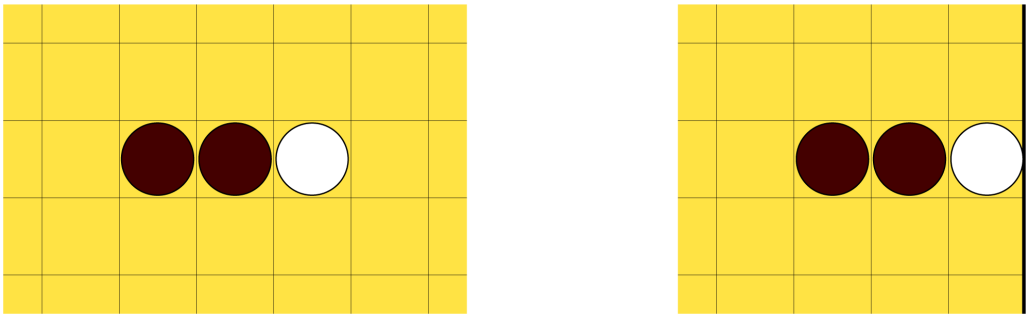


Figure 7.8 Board sides protect against jump capture.

The exposed White piece shown on the left is under threat of capture, while the White piece shown on the right is protected from capture by the side of the board. This is an example of a game with *asymmetric capture*, as one player is able to threaten the opponent without being threatened in turn.

Obviously, corner cells would offer even more protection in this game, however the *proxc* (proximity to corner) advisor was not activated for this particular game. It can be expected that such corner-seeking behaviour might eventually emerge given sufficient self-play, unless the limited mobility offered by the corners was an overriding factor.

Diagonal Preference

Although the default value for the influence advisor is (*infl* 1), larger values were found to occur for some games played on the square grid with diagonal neighbours. This did not appear to have any notable impact on play, however it is conjectured that this emphasis on piece influence encourages diagonal placements on such boards (Figure 7.9) that in some way benefit the games in question.

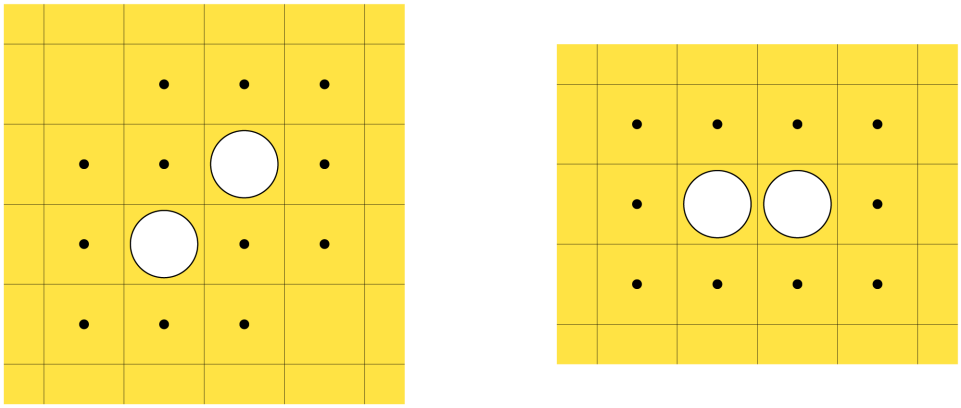


Figure 7.9 Diagonal moves influence more neighbours: 12 (left) and 10 (right).

For example, diagonal piece placement can lead to strong defensive formations in the game of Breakthrough, in which opposing sets of pieces must cross the board using Pawn-like movement and capture.

Similarly, connection games played on the square board with diagonal neighbours will generally benefit from diagonal play. Figure 7.10 shows two White formations at an equal distance to each of the White edges, however the formation on the left has more connective potential between the White edges than the single-file assault on the right.

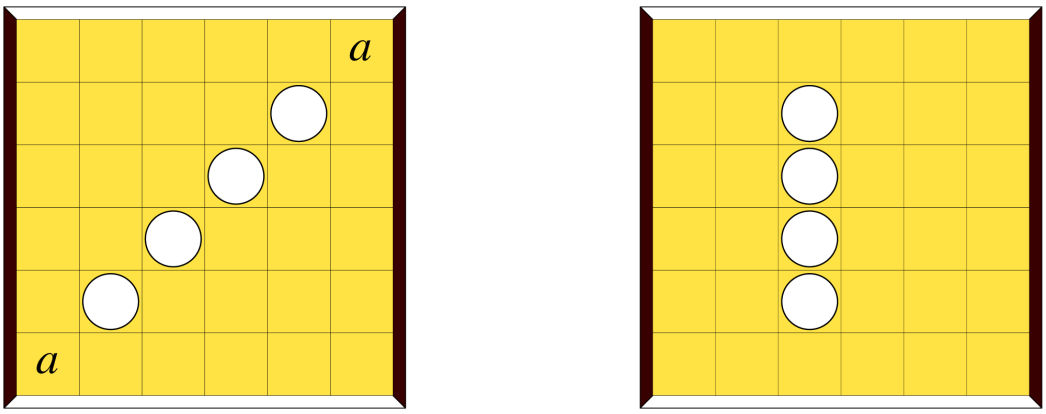


Figure 7.10 Diagonal connections are more dangerous.

Inexplicable *linecap* and *N*-in-a-row advisors would also occasionally occur in such connection games played on the square grid. Presumably the *linecap* advisor (which encourages or discourages pieces placed at the end of enemy lines) led to favourable local patterns, while the *N*-in-a-row advisor was useful as it encouraged diagonals to form and extend along the board's long diagonal (cells marked ***a***) as this is the line with the greatest potential for growth.

7.6 Move Priorities

Move ordering in the Ludi player's adversarial search was initially achieved through the use of priorities assigned to moves as a side-effect of the advisor evaluations. Each advisor therefore had three duties to perform with respect to a given board, player and end condition:

- To detect wins,
- To evaluate board positions, and
- To prioritise moves.

For example, the connection advisor marked moves to gaps in the best path of each player as high priority, the *N*-in-a-row advisor marked moves that extended lines (especially forking moves) as high priority, the material advisor marked moves for pieces under threat of capture as high priority, and so on.

The priority value for each move was then summed over all advisors for all players, indicating the relative urgency of each move. Opponent's priority values were added rather than subtracted because moves of urgency to both players are worth investigating.

The advantage of this approach was its utilisation of knowledge already gleaned during board evaluation that would otherwise be discarded, effectively saving the overhead cost of a 1-ply lookahead search for standard move ordering. Anshelevich [2002] and Van Rijswijck [2002] use similar methods for leveraging gained knowledge to good effect in their Hex playing programmes.

However, this idea was abandoned after it was found that such move priorities were not reliable in all cases, and performing move ordering using a standard 1-ply lookahead search provided a higher standard of play for a negligible loss of speed.

7.7 Summary

The strategy module handles board evaluation and policy optimisation tasks. Board evaluation is performed by a set of advisors, each of which return a value indicating their estimate of the value of a given board position to a given player, with respect to a given end condition. Advisors may be:

- *Static*: Initialised once when the game is loaded.
- *Dynamic*: Recalculated for every board position.

Advisors also perform the task of indicating when a particular end condition has been achieved for a particular player in a particular board position. This doubling up of duties makes this process more efficient.

The vector of weighting terms indicating the relative importance of each advisor constitutes the policy for that game. Each game has one recommended policy, however it is possible to assign different policies to different players in order to simulate different styles of play during game measurement.

The Ludi player is able to set a basic but functional default policy for new games and a NULL "fight or flight" policy for players without explicit end conditions. Policies may be optimised by self-play, although this is not guaranteed to improve upon a game's default policy within a reasonable amount of time.

Some innovative strategies emerged during self-play policy optimisation. The anti-stacking safeguard, in particular, proved so important that it was incorporated into the default policy of all relevant games.

Chapter 8

Criticism Module

> The best games are not those in which all goes smoothly and steadily toward a certain conclusion, but those in which the outcome is always in doubt.
> – George B. Leonard
>
> Where the pieces have different and bizarre motions, with various and variable values, what is only complex is mistaken (a not unusual error) for what is profound.
> – Edgar Allen Poe, *The Murders in the Rue Morgue*

8.1 Introduction

The Criticism module comprises the aesthetic system that measures each game and produces a corresponding aesthetic score in response. This is achieved by measuring trends in play over a number of self-play trials according to a collection of aesthetic criteria. This chapter describes the overall design of the aesthetic system based on a player-centric model, the categories of aesthetic criteria and their implementation.

8.2 Model

The general game model introduced in Section 5.2 is now expanded to incorporate a player model, then further expanded to define a complete aesthetic model.

8.2.1 Player Model

Given that many of the aesthetic criteria for game quality suggested in Section 4.3 are based on the players' understanding of the game and their experiences while playing it, it is useful to examine the relationship between the players and the game, as shown in Figure 8.1.

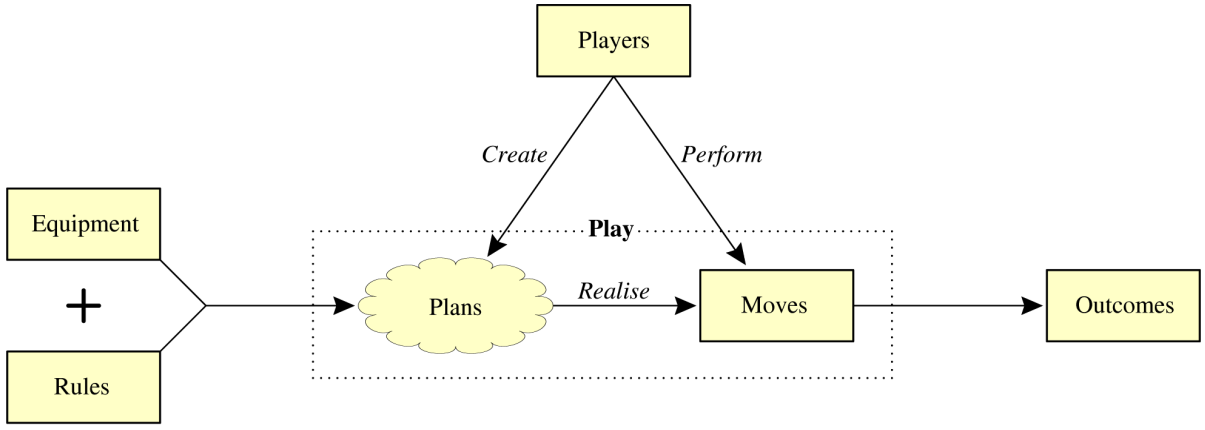


Figure 8.1 The player model.

In this context, *play* may be described as consisting of:

- *Plans* formulated by players in response to the equipment, rules and current state of play, and
- *Moves* that result from these plans.

Plans are the internal mindstates of the players that represent their understanding of the rules and their strategic planning; this is where the players' aesthetic appreciation of a game really lies. The resulting moves represent tangible realisations of these plans which may be measured. Analysing the moves of a game may therefore give some insight into the players' experience of that game.

8.2.2 Aesthetic Model

This player-centric model immediately suggests a way in which a game may be measured in terms of its:

- Rules and equipment,
- Play (or at least the moves), and
- Outcomes.

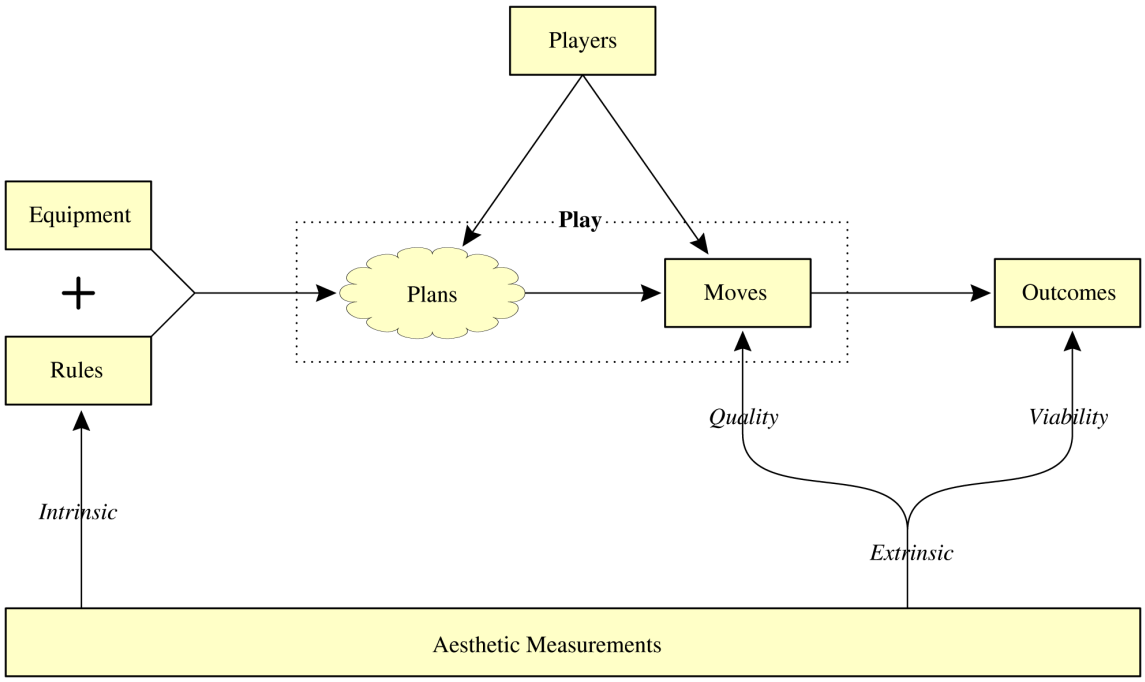


Figure 8.2 The aesthetic model.

This model, shown in Figure 8.2, distinguishes between the following aesthetic criteria:

- *Intrinsic:* Specific to the game's rules, and
- *Extrinsic:* Based upon the actual game play.

Intrinsic Criteria

Intrinsic criteria are those measurements that may be made directly from the game's rules and equipment, without the need to play out any actual games.

Note that "equipment" measurements in this context do not relate to the design or presentation of the board or components, but rather to their quality in an abstract sense as they relate to and constrain the game's rules and movements.

Extrinsic Criteria

Extrinsic criteria are those measurements based upon the actual game play, including the moves made by the players and the eventual outcomes. Extrinsic measurements are further divided into *quality* measurements based upon the moves and *viability* measurements based upon the outcomes.

Quality Criteria

Quality criteria are those measurements based upon the players' moves during self-play trials that aim to capture significant trends in play. While it is difficult to model the motivations behind each move, such measurements still give some indirect insight into the players' experience of the game.

Viability Criteria

Viability criteria are those measurements based on the eventual outcomes of self-play trials. These criteria indicate whether the game generally functions to produce a winner within a reasonable number of moves, and are more tangible and robust than the quality measurements.

8.2.3 Algorithmic Aesthetics Model

In terms of Stiny & Gips' [1978] criticism algorithm outlined in Figure 4.2, the Criticism module forms the aesthetic system. The input to this algorithm will be a text file containing a game's rule set and the description of the object will be the game's ludeme tree. A reference algorithm is not required in this case as any game successfully parsed and loaded by the Ludi general game player can be evaluated by self-play (subject to time constraints).

The interpretation of the object will be a set of aesthetic measurements and the algorithm's output will be a score predicting the quality of the game. A comparison algorithm is not required as the relationship between aesthetic scores is trivial: the higher the score, the more valuable the object.

Although it is tempting to assume that any aesthetic system for games would naturally follow the constructive mode of interpretation due to their rule-based nature, this is not the case in this study. Instead, an evocative approach is used based upon game preferences indicated by human players; the aesthetic system is trained on these player preferences to determine aesthetic criteria weightings. As it is not initially known which aesthetic criteria will prove important, a large number have been implemented to test a wide variety of game qualities.

8.3 Operation

Games are measured by conducting a number of self-play trials and performing the appropriate measurements before, during and after play. Each aesthetic criterion is represented by a single criterion object that produces a single value in response to the game, and the overall aesthetic score is the sum of each criteria value A_i multiplied by a corresponding weighting value w_i, plus a bias term w_c:

$$A_s = \left(\sum_{i=0}^{C-1} A_i w_i \right) + w_C$$

The weighting values w were obtained empirically by correlating the human player preference scores obtained in Experiment I with actual game measurements, as described in Chapter 11. The resulting aesthetic value for a given game is a prediction of the likelihood that human players will prefer that game over other games, and by implication find it interesting.

Game Trials

Most of the 57 aesthetic criteria implemented for this study are measured in a single batch of self-play trial games between balanced computer opponents. The trial size T is specified by the user, and has a default value of 20. This is quite a small value and reflects the need to process a large number of games in a reasonable time, however it proves sufficient in practice given that:

- Intrinsic criteria only involve a single measurement,
- Quality criteria generally involve multiple measurements per trial, and
- Viability criteria are generally more robust than other criteria.

The trial size T should be even so that games with perfect completion records have at least the chance to produce an equal number of wins for each player. Several of the criteria store multiple measurements per trial; their final scores will be the mean of all stored values.

Some criteria require additional trials to be run, typically with opponents pursuing unbalanced policies in order to expose flaws in the game. Such criteria are marked as "slow" and are not performed unless strictly necessary. The player modes are:

- Balanced,
- Skill level,
- Obstructive,
- Selfish, and
- Random.

Each trial consists of a maximum of M moves. Each game starts with a random sequence of R moves followed by intelligent computer moves until the game either concludes or is abandoned when the move limit is reached.

A default move limit M_{max} of 120 is used; halving this value gives an implied preferred game length M_{pref} of 60 moves:

$$M_{pref} = M_{\max} / 2$$

This default value was chosen due to the observation that games between human players who take around 30 seconds per move will take on average 30 minutes to complete (the computer will play much faster than this).

The default settings for the computer player are a search depth of 4, a beam width of 20 and random starting sequences of up to 10 moves in length; the exact number for each trial is chosen randomly. The user may adjust any of these settings as desired.

The board display is optionally updated with each move so the user can follow the self-play games. In addition, the current status of each measurement is displayed to the user in a results window during self-play. At the conclusion of each set of trials, the complete results including sample values from all criteria are stored to file and the aesthetic value returned.

All experiments were conducted on a standard Intel 1.8 GHz desktop machine. It is expected that similar results and levels of performance would be achieved on any similar machine.

Move Histories

In order to measure trends during play, it is useful to take a novel approach and examine the play histories of actual games. Figure 8.3 shows the *move history* of a game between two players, White and Black.

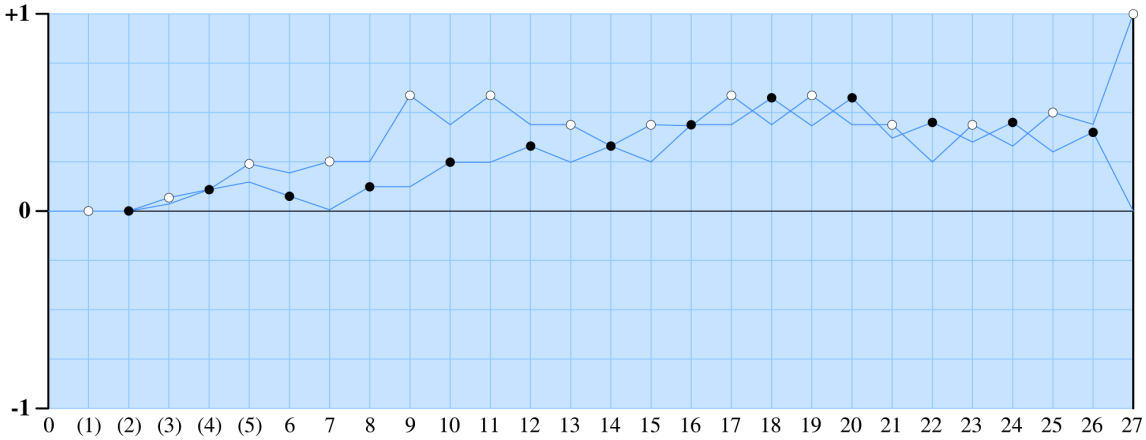


Figure 8.3 Move history of a game between White and Black.

The vertical axis shows estimated board evaluation in the range [0..1] and the horizontal axis shows the index of each move. The evaluation values are given by adversarial search informed by the current player's policy and indicate how close that player is to winning from the given board position. An evaluation of 1 implies a win (goal achieved) however evaluations of 0 do not necessarily imply a loss, rather they indicate zero progress towards the goal.

Note that even though the games in question are zero-sum in nature, this is not reflected in the graph (a loss would be indicated by a value of -1 in standard CGT descriptions). Thus this method of representing games is not limited to zero-sum games. This is not a typical game analysis technique, but has proved useful for the current project.

Move 0 in Figure 8.3 indicates the game's starting position before any moves are made. $E(m_0)$ the evaluation at move m_0 is 0 for both players in this case, but this is not necessarily the case as games that start with pieces on the board may begin with a base evaluation value higher than 0.

The last move (m_{27} in this case) indicates the end of the game. Note that the winner (White) has an evaluation value of 1 and the loser (Black) has an evaluation of 0 at this point.

The alternating white and black dots indicate moves by White and Black, respectively. It can be seen that White made the first move in this game. Moves with bracketed indices are random moves made to encourage novel opening positions. It can be seen that a player's board evaluation generally increases with their own moves and decreases with opponent moves, as expected.

Lead Histories

A game's move history becomes more useful when rephrased as a *lead history*. This simply involves plotting the difference in evaluation between the eventual winner and the eventual loser at each move, as shown in Figure 8.4.

The lead history gives an immediate indication of trends within the game. For instance, it can be readily seen that White establishes an early lead around moves m_9 to m_{11} but that Black fights back

into the game to take the lead at m_{18} before White eventually makes a strong move m_{27} to win the game.

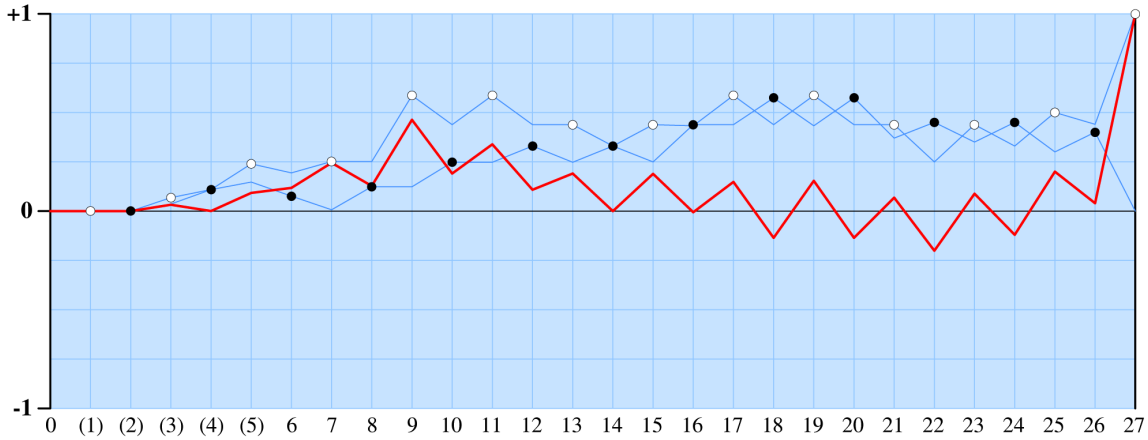


Figure 8.4 Lead history with respect to the winning player (White).

Note that move histories are restricted to the range [0..1] but lead histories have a range of [-1..1].

The lead history can reveal some useful information regarding trends of the play within a game. For example, the large number of zero-crossings in the latter half of the game shown in Figure 8.4 indicate a large number of lead changes and therefore a high degree of uncertainty as to the game's eventual outcome, while the fact that any fluctuations in the lead are small up until the winning move indicate that the game is relatively stable.

Opening Variety

Variation in play is a critical consideration in self-play trials between computer opponents. Computer players run the risk of following the same lines of play game after game if they are completely deterministic, in which case only a small part of the potential move space will be explored and games played will not be representative of actual games.

Variation in computer play is traditionally introduced by adding random variations to leaf node board evaluations, as noted in Section 3.2.3. However, this approach is not suitable for the current project as several of the proposed game measurements rely on comparisons between deep and shallow board evaluations, and the introduction of noise could be significantly detrimental to what are already somewhat imprecise measurements.

Instead, an alternative method is used to introduce variation into self-play: the first R moves of each game are randomly chosen from the available legal moves rather than determined by adversarial search. Moves made during these random sequences are described as *random moves* and subsequent moves made by the adversarial search as *intelligent moves*. Random starting sequences effectively constitute new starting positions that encourage novel lines of play to develop, and do not necessarily disqualify games from the definition of combinatorial. Care is taken to distinguish between random and intelligent moves in the various game measurements.

Note that this technique of opening variety only applies to self-play between computer opponents; it is assumed that human players will bring their own degree of random play to the game. Further variety is added on a per-move basis by shuffling the legal moves generated for each board position before move ordering by 1-ply lookahead, randomising the choice between moves of equal (shallow) value.

8.4 Aesthetic Criteria

The aesthetic criteria are measurements that quantify several of the intuitive aspects of game quality presented in Chapter 4, in order to inform the aesthetic system. A total of 57 criteria are implemented as follows:

- 16 × Intrinsic criteria
- 41 × Extrinsic criteria
  - 30 × Quality criteria
  - 11 × Viability criteria

Many of the measurements assume that a deep evaluation from a lookahead search will be more accurate than shallow evaluation. Sadikov et al [2005] discuss the problem of *minimax pathology*, that is, the fact that higher search depths can actually decrease the performance of computer players (although they found that this did not actually stop the computer players from playing well). For the low search plies used in this current task (no more than 4-ply) this is unlikely to be an issue [Matsubara et al, 1996]. It is assumed that deeper search depths lead to better moves, and experience with the computer player bears this assumption out.

Not surprisingly, depth – one of the most important indicators of game quality – is also the most difficult aspect to measure. Several depth-related criteria are provided, however a concrete measurement of depth is yet to be defined.

The measurements are described using the following terms:

G Total number of game trials for the current game.
There are typically 20 game trials per measurement run.

G_o Total number of game trials with clear {win / loss} outcomes ($G_o \leq G$).

M_g Total number of moves in game trial g.

M_{gr} Total number of random moves in game trial g.

M_{gi} Total number of intelligent (i.e. non-random) moves in game trial g ($M_{gr} + M_{gi} = M_g$).

m_n The n^{th} move of the current game trial, where:

- m_0 denotes the starting position before any moves are made,
- m_1 denotes the first move, and
- m_i denotes the first intelligent (i.e. non-random) move, and
- m_M denotes the last move.

m_{gn} The n^{th} move of game trial g.

m_{nr} The r^{th} reply available to the current player after the n^{th} move.

$E_W(m_n)$ Board evaluation for White at move m_n of the current game trial.

$E_B(m_n)$ Board evaluation for Black at move m_n of the current game trial.

$E_w(m_n)$ Board evaluation for the eventual winner of the current game trial at move m_n.

$E_l(m_n)$ Board evaluation for the eventual loser of the current game trial at move m_n.

$E_c(m_n)$ Board evaluation for the current mover at move m_n of the current game trial.
This is the player who has just made their move.

$E_o(m_n)$ Board evaluation for the opponent (non-mover or next player) at move m_n of the current game trial.

$E_{co}(m_n)$ Difference in board evaluations for the current player and their opponent at move m_n of the current game trial ($E_{co}(m_n) = E_c(m_n) - E_o(m_n)$). This is the lead at m_n.

$E_{oc}(m_n)$ Difference in board evaluations for the opponent (non-mover) and current player for move m_n of the current game trial ($E_{oc}(m_n) = E_o(m_n) - E_c(m_n)$).
This is the negative lead at m_n.

Many measurements do not take readings for random moves or the winning move. In such cases, random moves are not included as they do not reflect actual play, and the winning move is not included as the abrupt change in evaluation on the final move may unduly bias the results.

As an aside, it is conjectured that the UCT algorithm (Section 3.2.4) could provide a number of useful additional measurements based upon the pattern of exploration/exploitation choices over a large number of random playouts. This might have particular relevance to measurements regarding the clarity of depth of a game, as UCT adjusts its search depth asymmetrically as required. Again, however, playout speed was a limiting factor and such measurements were not suitable for the current implementation.

8.4.1 Intrinsic Criteria

Intrinsic criteria are those measured directly from the game's equipment and rules, without the need to play any games out.

16 intrinsic criteria are implemented. These include measurements of rule complexity, piece movement type and goal type.

Complexity

Complexity is a measure of game's rule complexity, based on the number of items related to piece movement and winning conditions in its ludeme tree. Only piece movement and winning condition items are counted because these are the aspects with which players will be primarily concerned over the course of the game; rules relating to starting position and board configuration are taken for granted once the game has started.

Rule complexity was one of the user-defined parameters in Pell's METAGAMER game generator [1993].

Assuming a function *count*(*element*) that recursively counts the total number of attributes and subelements contained in the specified element, complexity is measured as follows:

$$A_{comp} = \min(1, \log_{10}(count(pieces) + 4 * count(end) + 1) / 2)$$

The end condition count is multiplied by a factor of 4 as goal clauses tend to be more succinct and information rich than movement clauses, which are typically verbose sets of simple instructions. The $\log_{10}$ component of the calculation is divided by 2 to set a range limit of approximately 100 rules.

The criterion returns a floating point value in the range [0..1] where 0 indicates low rule complexity and 1 indicates high rule complexity. A single value is stored for each game.

Movement Type

This collection of seven features indicates whether the game involves certain types of piece movement. This information is extracted directly from the (*pieces*) element of the game's ludeme tree.

The following movement types are detected:

- Addition,
- Removal,
- Movement,
- Stacking,

- Capture,
- Conversion, and
- Promotion.

This is not an exhaustive list of possible movement styles allowed by the Ludi GDL, but it covers the main movement types involved in the 79 source games used for Experiment I. A complete list of the games is provided in Appendix C.

Each of these criteria return a boolean value {0 | 1} indicating the absence or presence of the respective movement type. A single value is stored for each type for each game.

Goal Type

This collection of eight features indicates whether the game involves certain types of end conditions. This information is extracted directly from the (*end*) element of the game's ludeme tree.

The following goal types are detected:
- Connect,
- Capture,
- *N*-in-a-Row,
- Race,
- Group,
- Stack,
- Score, and
- Block.

Again, this is not an exhaustive list of possible end conditions allowed by the Ludi GDL, but it covers the main goal types used in the 79 source games for Experiment I.

Each of these criteria return a boolean value {0|1} indicating the absence or presence of the respective goal type. A single value is stored for each type for each game.

8.4.2 Extrinsic Criteria

Extrinsic criteria are those measured during and after play. 41 extrinsic criteria are implemented.

8.4.2.1 Quality Criteria

Quality criteria are those measured during the course of play, typically from the game's move and lead histories.

30 quality criteria are implemented. These typically attempt to capture trends in play that may indicate players' experiences over the course of the trial games.

Branching Factor

A game's *branching factor* (number of move choices per turn) indicates the combinatorial complexity of the game, and the amount of information that the player must process each turn in order to make reasonable moves.

Branching factor may not be of overwhelming importance if the game has good clarity and it is easy to distinguish sound moves from unsound ones. For instance, Go is widely considered to be one of the

very best combinatorial board games, despite having one of the largest branching factors at around 235 choices per move [Coates, 2005].

If a game has poor clarity, on the other hand, then too much complexity can appear as noise [Koster, 2004] and should be avoided if possible

A game's average branching factor is calculated as follows:

$$A_{bf} = \sum_{g=1}^{G} \min\left(1, \log_{10}\left(\frac{\sum_{n=0}^{M_g-1} moves(m_n)}{M_g} + 1\right) / 2\right) / G$$

where $moves(m_n)$ is the total number of move choices available to the current mover at move m_n. The $\log_{10}$ component of the calculation is divided by 2 to set a range limit of approximately 100 moves.

For each game trial, a measurement is made for the starting position m_0 as well as every non-winning move (random moves still provide valid branching factors) and the logarithm of the average value for that game stored. The final result is the mean value over all game trials.

The criterion returns a floating point value in the range [0..1] where 0 indicates low branching factor and 1 indicates high branching factor.

Response Time

Response time measures the average amount of time required by computer player to formulate each move at its default settings. Like the branching factor criterion, this value indicates the amount of information the player must process each turn. However, response time also incorporates the complexity of the game's rules; games with larger branching factors and more complex rule sets will result in longer response times.

Response time is calculated as follows:

$$A_{rt} = \sum_{g=1}^{G} \min\left(1, \log_{10}\left(\frac{\sum_{n=1}^{M_g} time(m_n)}{M_g} + 1\right) / 2\right) / G$$

where $time(m_n)$ returns the total time in seconds required to formulate m_n. The $\log_{10}$ component of the calculation is divided by 2 to set a range limit of approximately 100 seconds.

For each game trial, a measurement is made for every move and the logarithm of the average value for that game stored. The final result is the mean value over all trials.

The criterion returns a floating point value in the range [0..1] where 0 indicates low (quick) response time and 1 indicates high (slow) response time.

Move Effort

The *move effort* of a game is the average number of lookahead moves visited in the computer's search for each move. Like the branching factor and response time criteria, this value indicates the amount of information the player must process each turn. It is somewhat more reliable than response time as its value will be independent of move complexity and the relative speeds of different advisor implementations.

Move effort is calculated as follows:

$$A_{me} = \sum_{g=1}^{G} \min\left(1, \log_{10}\left(\frac{\sum_{n=1}^{M_g} nodes(m_n)}{M_g} + 1\right) / 6\right) / G$$

where $nodes(m_n)$ is the total number of nodes visited in the computer search to formulate move m_n. The $\log_{10}$ component of the calculation is divided by 6 to set a range limit of approximately 1,000,000 nodes.

For each game trial, a measurement is made for every move and the logarithm of the average value for that game stored. The final result is the mean value over all trials.

The criterion returns a floating point value in the range [0..1] where 0 indicates small move effort and 1 indicates large move effort.

Search Penetration

Search penetration describes the average shape of the search tree created when formulating each move; a search is more penetrating if it is deep and narrow rather than shallow and wide.

For each non-random move of each game trial, the search ply of each node visited in the lookahead search leading up to that move is squared, accumulated then divided by the total number of search nodes.

$$A_{spen} = \sum_{g=1}^{G} \min\left(1, \log_{10}\left(\sum_{n=M_{gr}+1}^{M_g} \frac{\sum_{i=0}^{nodes(m_n)} depth(i)^2}{nodes(m_n)} / M_{gi} + 1\right) / 2\right) / G$$

where $depth(i)$ returns the depth (search ply) of node i in the search tree. The $\log_{10}$ component of the calculation is divided by 2 to set a range limit of including search plies (squared) up to 100.

The logarithm of the average value for each game trial is stored, and the final result is the mean value over all trials. This criterion theoretically returns a floating point value in the range [0..1] 0 indicates minimal penetration and 1 indicates maximal penetration at ply 10 search. In practical terms, however, the range is actually [0..0.277] for searches up to 4 ply.

Clarity (Variance)

The *variance* criterion measures the amount of variance in evaluations of the moves available to the current player at each turn. This is related to the concept of clarity as it models the amount of information the player must process in order to determine which moves are the most promising from the available choices.

It was initially assumed that high variance would equate with high clarity, as a distribution of move estimates with a wide spread of values might be more readily separated into "promising" and "unpromising" categories. However, some analysis soon revealed that this not the case; a distribution of move estimates with high clarity is more likely to have low variance but a few significant outliers that indicate the most promising moves.

Special cases worth considering are games in which all moves are equally good, for instance those in which players take turns placing a piece on an empty cell and the winner is the first player unable to place a piece [Allis 1991]. Despite having no outliers (all moves are equally good) such games will have maximum clarity and theoretically zero variance if the move estimates are accurate.

The average variance in evaluations for each potential non-random and non-winning move is stored for each game trial, and the final result is the mean value over all trials.

$$A_{\mathrm{var}\,n} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}}^{M_g - 1} VE_c(m_n)}{M_{gi} - 1} / G$$

where $VE_c(m_n)$ is the variance in the evaluations of moves available to the current player at move m_n.

This criterion returns a floating point value in the range [0..1] where 0 indicates poor clarity based on variance and 1 indicates high clarity based on variance.

Clarity (Narrowness)

The *narrowness* criterion measures the tendency for a small number of promising moves to stand out from the choices available at each turn. Like the variance measurement, this criterion is related to the game's clarity.

Allis et al [1991] describe a pathological game played on a 19x19 Go board in which players take turns placing a piece of their colour, no capturing takes place, and the player to place the last piece wins. Such a game would have minimal narrowness as any move is as good as any other, and would also be of minimal interest to the players.

For each non-random and non-winning move m_n, evaluations are made for all potential moves from the current board position. The average evaluation $\bar{E}_c(m_n)$ and maximum evaluation $E_c max(m_n)$ are determined for this set of potential moves and the number falling at least 75% above the difference between the maximum and the mean are counted.

The narrowness of a move m_n is given by:

$$narrowness(n) = \frac{\underset{[0<=r<moves(m_n)]}{count} \left(E_c(m_{nr}) > \overline{E}_c(m_n) + 0.75 \times \left(E_c \max(m_n) - \overline{E}_c(m_n) \right) \right)}{moves(m_n)}$$

and the average narrowness of the game is given by:

$$A_{narr} = \sum_{g=1}^{G} \left(1 - \frac{\sum_{n=M_{gr}}^{M_g - 1} narrowness(n)}{M_{gi} - 1} \right) / G$$

where $moves(m_n)$ is the number of potential moves available at the board position corresponding to move m_n.

The average value is stored for each game trial, and the final result is the mean of these values. This criterion returns a floating point value in the range [0..1] where 0 indicates poor clarity based on narrowness and 1 indicates high clarity based on narrowness.

Convergence

Convergence is the tendency for the number of moves available at each turn to decrease as the game progresses. Convergent games have the attractive property than an outcome is guaranteed after a certain number of moves.

If branching factor is plotted against game duration, then convergence may be measured by the slope of this plot. A convergent game will slope downwards while a divergent game, in which the number of moves increases over the game's length, will slope upwards. Simple linear regression is used:

$$slope(g) = \frac{\sum_{n=0}^{M_g - 1} (x_n - \bar{x})(y_n - \bar{y})^2}{\sum_{n=0}^{M_g - 1} (x_n - \bar{x})^2}$$

where the x terms denote non-winning move numbers n and the y values denote the branching factor at each non-winning move m_n. The convergence is then measured as follows:

$$A_{conv} = \sum_{g=1}^{G} \left(1 - \left(slope(g) / 2 + 0.5\right)\right) / G$$

A measurement is stored for each game, and the final result is the average over all games. Note that the slope is negated and normalised to the range [0..1] so that higher values indicate greater convergence.

This criterion returns a floating point value in the range [0..1] where 0 indicates maximum divergence and 1 indicates maximum convergence.

Uncertainty (Overall / Early / Late)

Uncertainty is the tendency for the outcome of the game to remain uncertain for as long as possible, ideally until a decisive lead is established in the last few moves. The sooner a game's outcome is known the less interesting it becomes (especially for the losing player).

Figure 8.5 (left) shows the lead history of an uncertain game in which neither player establishes a dominating lead and the eventual winner is not known with any certainty until the winning moves. On the other hand, Figure 8.5 (right) shows a game in which Black establishes a dominant lead on their second move and continues to extend this lead until winning the game. The outcome of this game is known with reasonable certainty from a very early stage.

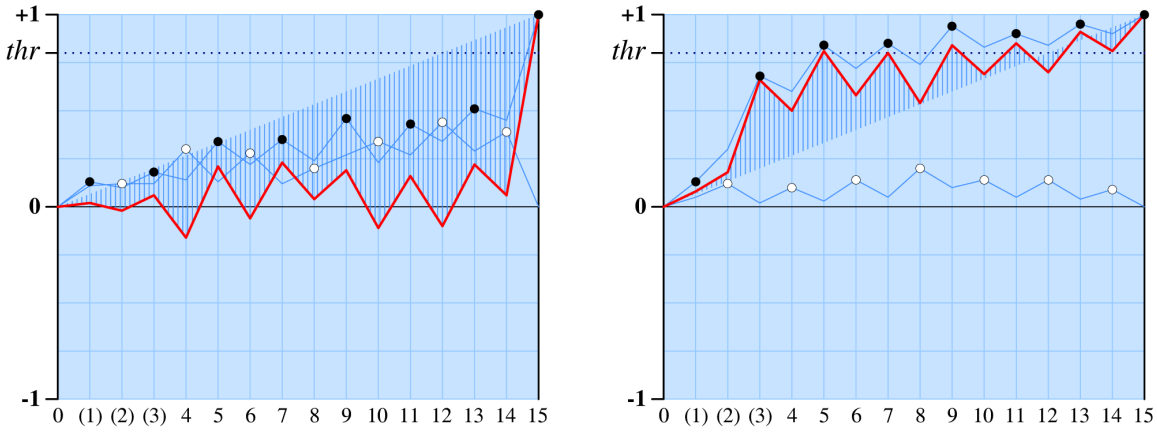


Figure 8.5 An uncertain game (left) and a certain game (right).

The amount of uncertainty is indicated by the area enclosed by the lead plot and an imaginary line from (0, 0) to (*M*, 1). This line describes a linear increase in lead over the course of the game, starting with a neutral lead of 0 and culminating with a winning lead of 1; this is the line of *average certainty* that a neutral game would be expected to follow. The game is uncertain if the majority of the enclosed area is below the line of average certainty (left) and certain if the majority of the enclosed area is above this line (right).

This area is approximated by taking *S*=100 samples at regular intervals *t* = [0..1] over the course of the game, and finding for each t the difference between the average certainty line and the interpolated value of the lead plot at this point. These differences are accumulated over all games and the average value for each *t* stored; the final result is the mean of these averages.

The uncertainty criteria are divided into overall, early and late types. The *uncertainty (overall)* criterion performs the calculation described above:

$$A_{unco} = \sum_{s=0}^{S-1} \left(\min\left(1, \left(t - \left(\sum_{g=1}^{G} E_{co}(m_{tM_g}) \right) / G \right) / t \right) \right) / S$$

where *t* is the time value *s*/(*S*-1) and $E_{co}(m_{tM_g})$ gives the interpolated lead at time *t*. Note that *t* also gives the average uncertainty at time *t*.

The *uncertainty (early)* criterion performs a similar calculation but emphasises earlier values towards the start of the game (*t*=0):

$$A_{unce} = \sum_{s=0}^{S-1} \left(\min\left(1, \left(t - \left(\sum_{g=1}^{G} E_{co}(m_{tM_g}) \right) / G \right) (1-t) / t \right) \right) / S$$

The *uncertainty (late)* criterion performs a similar calculation but emphasises later values towards the end of the game (*t*=1):

$$A_{uncl} = \sum_{s=0}^{S-1}\left(\min\left(1, t - \left(\sum_{g=1}^{G} E_{co}(m_{tM_g})\right)/G\right)\right)/S$$

Májek & Iida's [2004] more sophisticated method for measuring uncertainty based on the progressive entropy of the game was not used for two main reasons:

- The difficulty of providing a uniform game distance metric over arbitrary rule sets, and
- Preliminary tests revealed that insufficient data could be generated within the allotted number of trials (Májek & Iida [2004] typically ran 4,000 self-play games per measurement).

These criteria each return a floating point value in the range [-1..1] where -1 indicates maximum certainty and 1 indicates maximum uncertainty.

Drama (Average / Maximum)

Drama is the tendency for players to recover from seemingly bad positions. Drama is good for a game as it means that the leading player will not necessarily win the game and the eventual outcome remains uncertain until that lead becomes decisive.

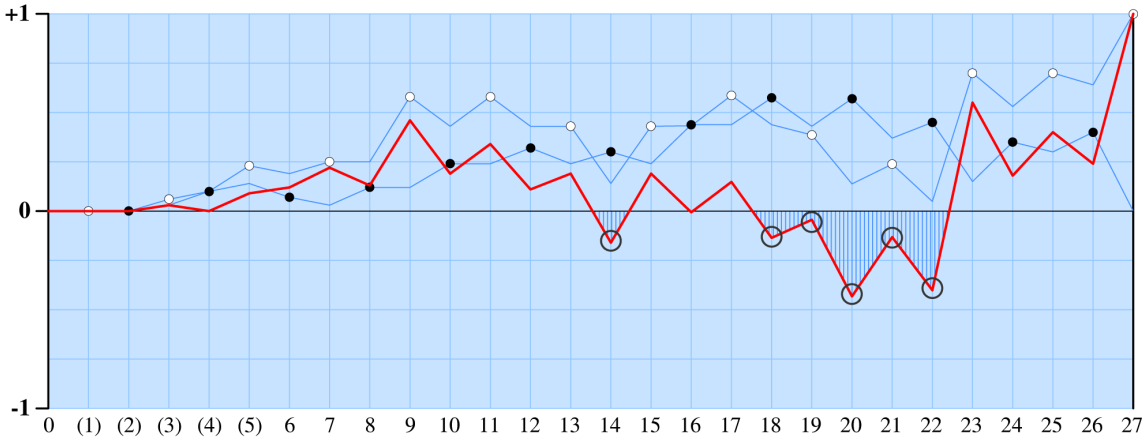


Figure 8.6 A dramatic recovery by White.

Drama is indicated by the number of moves that the eventual winner spends in a trailing position, and the severity of each such position. This corresponds to the shaded regions in Figure 8.6, which show time spent by lead plot below the zero line for six moves of interest.

For each game, the *drama (average)* criterion gives the sum of these lead differences for each non-random and non-winning move divided by the total number of such moves:

$$A_{drav} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g-1} E_w(m_n) < E_l(m_n) \begin{cases} \sqrt{E_l(m_n) - E_w(m_n)} \\ 0 \end{cases}}{\underset{[M_{gr}+1 \le n \le M_g-1]}{count}\left(E_w(m_n) < E_l(m_n)\right)} / G$$

The *drama (maximum)* criterion gives the maximum of these lead differences over all non-random and non-winning moves:

$$A_{drax} = \sum_{g=1}^{G} \left(\max_{[M_{gr}+1 \le n \le M_g - 1]} E_w(m_n) < E_l(m_n) \begin{cases} \sqrt{E_l(m_n) - E_w(m_n)} \\ 0 \end{cases} \right) / G$$

The square root of the lead difference is taken in each case to emphasise larger dramatic steps.

The average and maximum drama values are stored for each game, and the final results are the mean over all games. Each of these criteria return a floating point value in the range [0..1] where 0 indicates lack of drama and 1 indicates a high degree of drama.

Killer Moves

This measurement provides an estimate of the tendency of *killer moves* to occur in a game. A killer move is one that significantly improves the player's position and typically swings the outcome of the game. Killer moves are therefore closely related to drama.

Note that the term "killer move" has a different meaning here than its use in traditional game programming, where it refers to moves that help prune the game tree during adversarial search.

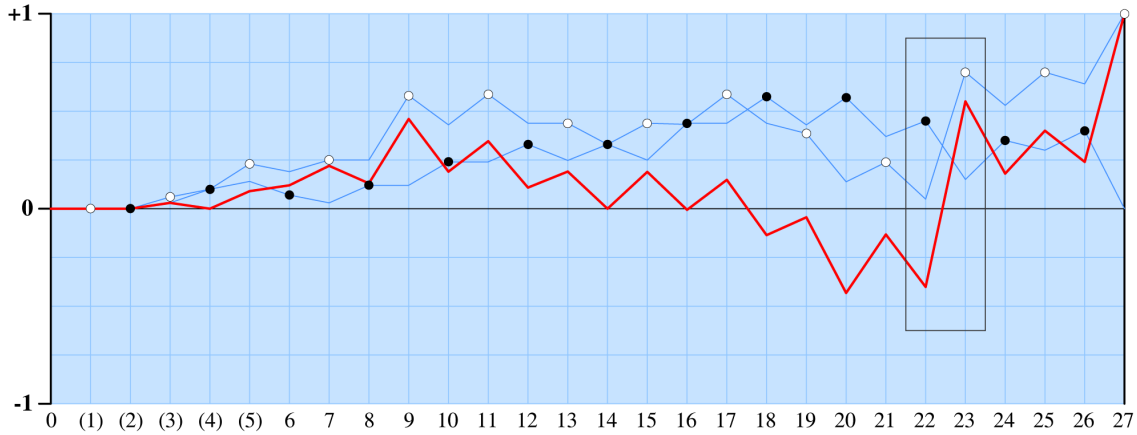


Figure 8.7 A killer move.

Figure 8.7 shows a killer move m_{23} by White which dramatically swings the lead in their favour. Note that neither the increase in White's evaluation nor the decrease in Black's evaluation is especially dramatic; the killer move is a combination of both.

Killer moves are measured by finding, for each game, the non-random and non-winning move that results in the greatest relative gain for the mover.

$$A_{km} = \frac{\sum_{g=1}^{G} \max_{[M_{gr} \le n \le M_g - 1]} \left(\left[E_c(m_n) - E_o(m_n) \right] - \left[E_o(m_{n-1}) - E_c(m_{n-1}) \right] \right)}{G}$$

One reading (the maximum value) is stored for each game trial, and the final result is the arithmetic mean of these values over all trials.

This criterion returns a floating point value in the range [0..1] where 0 indicates a lack of killer moves and 1 indicates a prevalence towards sizable killer moves.

Permanence

This criterion measures the tendency for players to immediately recover from the opponent's last move in terms of board position. Games are described as *permanent* if the player cannot immediately recover to negate the effect of the opponent's last move.

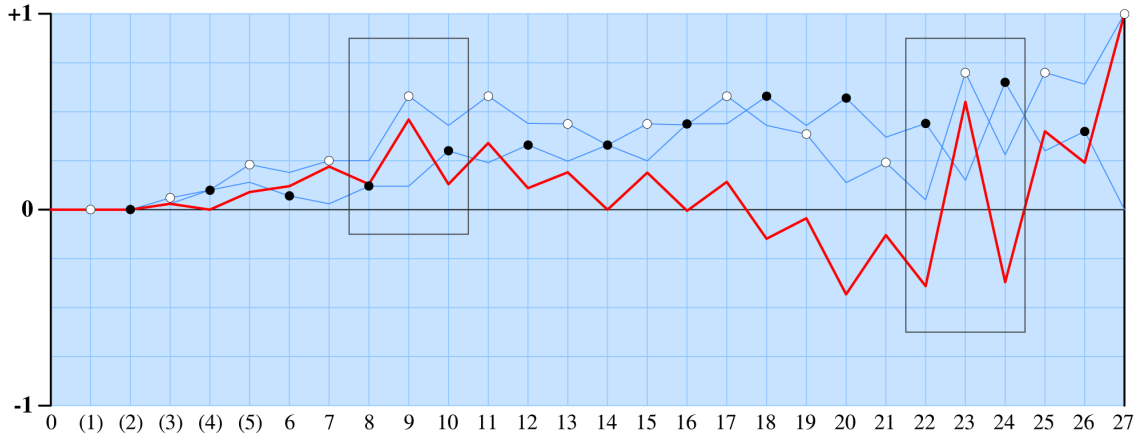


Figure 8.8 Two cases of move recoveries.

Figure 8.8 shows two instances of move recoveries. The leftmost example shows White's lead increase by a small amount at move m_9 and the rightmost example show White's lead increase by a much larger amount at move m_{23}. In both cases the opponent's reply reduces the lead again by similar amounts, however the second case involves a much larger overall increase and corresponding correction.

For each non-random and non-winning triplet of moves $\{m_{n-2}, m_{n-1}, m_n\}$, the decrease in relative board evaluation caused by the opponent's last move m_{n-1} is subtracted from the increase caused by the current player's replying move m_n. This is more meaningful than simply measuring the difference between the relative board evaluations at m_n and m_{n-2} as it incorporates the magnitudes of any decrease and subsequent correction.

$$A_{perm} = \sum_{g=1}^{G} \left(1 - \frac{\sum_{n=M_{gr}+1}^{M_g - 1} \left[\left(E_{co}(m_n) - E_{oc}(m_{n-1}) \right) - \left(E_{oc}(m_{n-1}) - E_{co}(m_{n-2}) \right) \right]}{M_{gi} - 2} \right) / G$$

The average value of all such triplets is stored per game trial, and the final result is the mean over all trials. This criterion returns a floating point value in the range [0..1] where 0 indicates a lack of permanence and 1 indicates a high degree of permanence.

Lead Change

The *lead change* criterion, as its name suggests, indicates the tendency for the lead to change over the course of a game.

For example, Figure 8.9 shows a game in which the lead has changed four times. This is indicated by the number of points at which the move history plots for both players cross each other, equivalent to the number of zero-crossings in the lead plot, for each non-random move of each game.

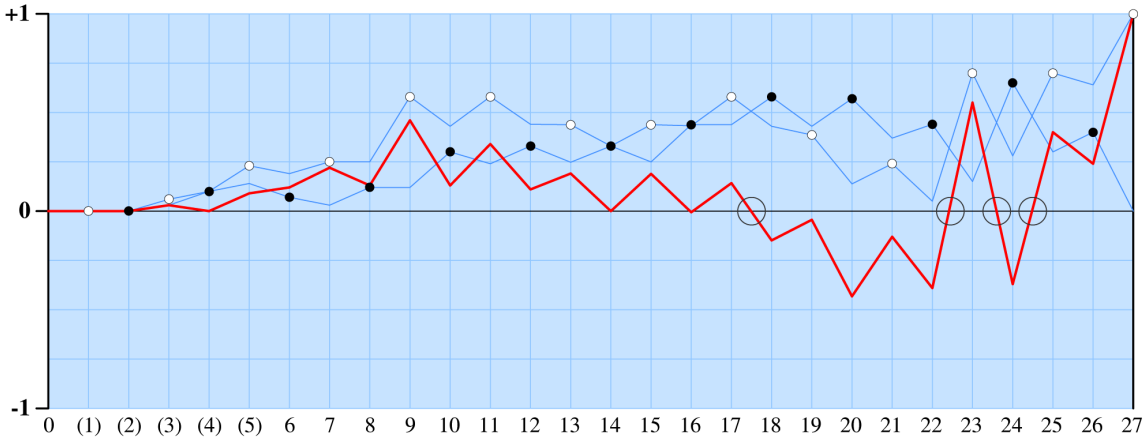


Figure 8.9 A game with four lead changes (indicated).

$$A_{lead} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g} leader(m_n) \neq leader(m_{n-1}) \begin{cases} 1 \\ 0 \end{cases}}{M_{gi} - 1} / G$$

where *leader*(m_n) is the leading player at move m_n, or if equal then the last player to have the lead.

One reading is stored for each game trial, and the final result is the mean of these values over all trials. This criterion returns a floating point value in the range [0..1] where 0 indicates no lead change and 1 indicates that every move is a lead change.

Stability

Stability is a measure of fluctuations in player's evaluations over the course of the game. This measure is closely related to the lead change and permanence measures. Games should generally have a moderate degree of stability; too much stability can indicate a lack of drama, whereas too little stability can indicate a random game with little cohesion.

$$A_{stab} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g - 1} \left| E_W(m_n) - E_W(m_{n-1}) \right| + \left| E_B(m_n) - E_B(m_{n-1}) \right|}{M_{gi} - 2} / G$$

For each non-random and non-winning move of each game, the absolute difference in player evaluation between consecutive moves is stored for each player. The final value is the overall mean over all games.

This criterion returns a floating point value in the range [0..1] where 0 indicates low stability and 1 indicates high stability.

Coolness

Coolness is a measure of the degree to which players are forced to moves that harm their position.

Figure 8.10 shows an example of a cold move m_{19} which actually worsens White's position. This is presumably the best move that White could have made under the circumstances.

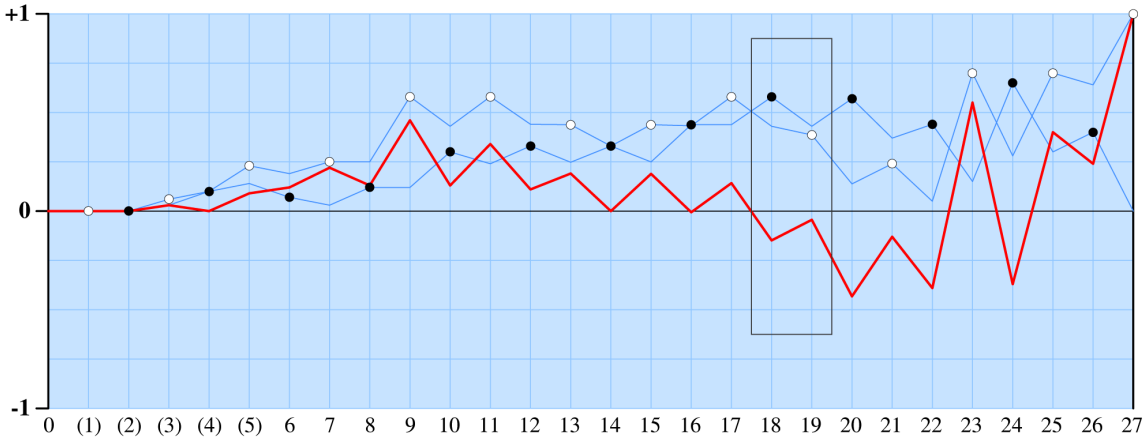


Figure 8.10 A cold move.

Cold situations are generally detrimental to a game as players seek to make the least-worst move rather than striving to make the most positive move each turn, which can be unsatisfying. Even worse, the game may degenerate into a *cold war* [Browne, 2005] in which players without positive move choices must continue to make inconsequential moves until forced to make a decisive losing move. Such games are especially unsatisfying if the players can deduce long beforehand who will be forced to make the losing move.

Coolness is simply measured as the average decrease in board evaluation for the current mover (if any) for each non-random and non-winning move.

$$A_{cool} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g-1} E_c(m_n) - E_o(m_{n-1}) \geq 0 \begin{cases} 0 \\ E_c(m_n) - E_o(m_{n-1}) \end{cases}}{M_g - 2} / G$$

More sophisticated coolness measurements might involve analysis of all replies in the current player's move tree, however the fact that the player makes a detrimental move is sufficient to indicate some degree of coolness.

One reading is stored for each valid move, and the final result is the mean of these values over all game trials. This criterion returns a floating point value in the range [0..1] where 0 indicates a lack of coolness and 1 indicates a high degree of coolness.

Decisiveness Threshold

The *decisiveness threshold* of a game indicates the point at which the leading player is generally known to have a winning advantage, that is, it can be stated that the leading player will go on to win the game with a reasonable degree of confidence.

For example, Figure 8.11 shows a decisiveness threshold of $th \approx 0.8$ for the shown game. White approaches this value with move m_{12} but does not actually reach it until move m_{20}, after which the game is soon won. Ideally games should reach a conclusion quickly once this threshold is reached. As an aside, this game is notable for the fact that White looks to be in a losing position around move m_{18} but then replies with the killer move m_{20} to steal a decisive lead and win the game.
The decisiveness threshold can be estimated by contradiction, by observing the maximum relative lead achieved by the loser over all histories for that game. In other words, it is the maximum lead that a player achieves without realising victory.

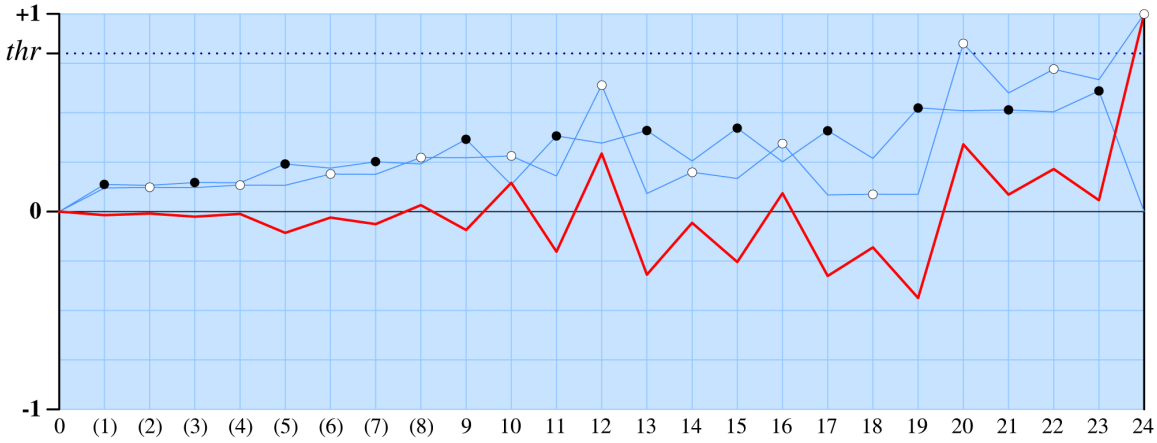


Figure 8.11 Decisiveness threshold.

$$A_{thr} = \max_{[1 \le g \le G, M_{gr} \le n \le M_g - 1]} \left(E_l(m_{n,g}) - E_w(m_{n,g}) \right)$$

This criterion stores a single value indicating the threshold value. It returns a floating point value in the range [0..1] where 0 indicates a low decisiveness threshold and 1 indicates a high decisiveness threshold.

Decisiveness

A game is *decisive* if a player wins quickly after reaching the decisiveness threshold for that game. Decisiveness is good as a game is of little interest once a player has established a commanding lead and is reasonably certain of victory.

For example, Figure 8.12 shows two instances of a game with a decisiveness threshold of $th \approx 0.8$. The instance on the left shows Black reaching this threshold value with move m_{11} then winning decisively a few moves later. On the other hand, the instance on the right shows Black reaching the threshold value early on with move m_5 (presumably due to some luck in the choice of the initial three random moves) but requiring several more moves to secure the win; this victory is far from decisive.

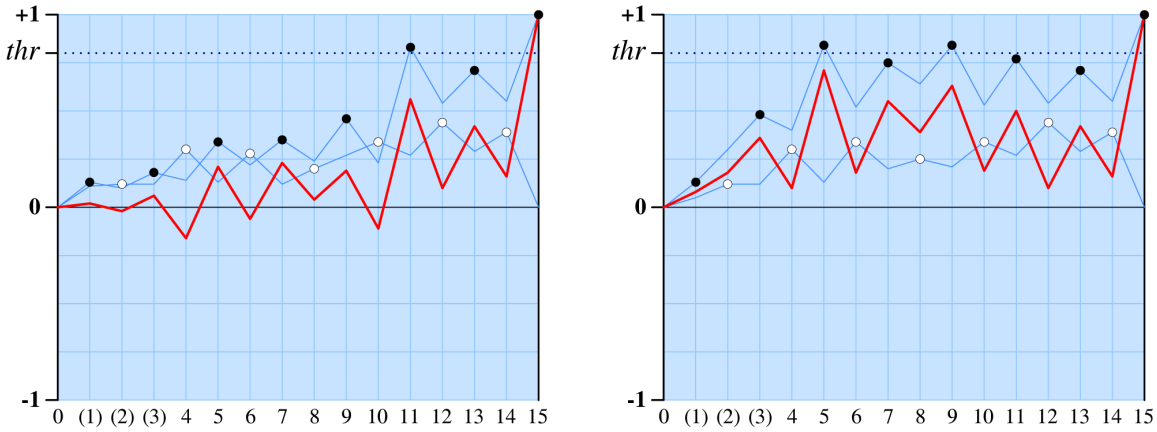


Figure 8.12 A decisiveness game (left) and an indecisive one (right).

The decisiveness value is the ratio of moves after the decisiveness threshold is reached to the total number of moves, averaged over all game trials.

$$A_{dec} = 1 - \sum_{g=1}^{G} \frac{M_{gd} - M_g}{M_g} / G$$

where M_{gd} is the move number at which the decisiveness threshold is reached for game g.

This criterion returns a floating point value in the range [0..1] where 0 indicates low decisiveness and 1 indicates high decisiveness.

Depth (Discrepancy)

Discrepancy is a measure of the average difference between each move's shallow and deep evaluations. A move's *shallow evaluation* is the immediate board evaluation made after making the move (ply 1 lookahead) while a move's *deep evaluation* is result returned by a full alpha-beta search (ply *N* lookahead).

Larger discrepancy values indicate that deeper (and theoretically more accurate) searches reveal information about a given board position that cannot be gleaned from immediate evaluations of the available moves. It can be argued that this hints at the depth of the game; experienced players with a deeper understanding of the game may prefer moves that novice players with a superficial understanding would overlook.

The absolute difference in shallow and deep evaluations is measured for each non-random and non-winning move of each game. The final value is the overall mean over all games.

$$A_{disc} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g - 1} \left| ED_c(m_n) - ES_c(m_n) \right|}{M_{gi} - 2} / G$$

where $ES_c(m_n)$ is the shallow (ply 1) evaluation for move m_n for the current player and $ED_c(m_n)$ is the deep (ply N) evaluation for move m_n for the current player.

This criterion returns a floating point value in the range [0..1] where 0 indicates no discrepancy and 1 indicates high discrepancy.

Board Coverage

This criterion measures the ratio of the board cells visited by any piece over the course of each game. Low board coverage indicates that the game's starting position or movement rules may not be suitable for its board topology; or perhaps the board is simply too big.

The ratio of visited cells to total cells is stored for each trial, and the final value is the mean over all game trials.

$$A_{b\,\mathrm{cov}} = \sum_{g=1}^{G} \frac{cells\_visited(g)}{TotalCells} / G$$

where *cells_visited*(g) returns the number of cells visited by any piece in game trial g.

This criterion returns a floating point value in the range [0..1] where 0 indicates low board coverage and 1 indicates high board coverage.

Momentum (1 / 2 / 3)

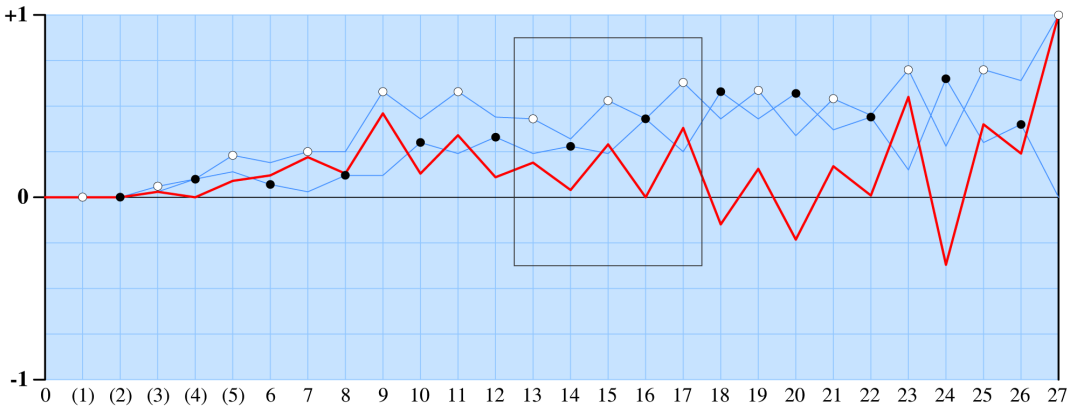


Figure 8.13 White carries the momentum.

Momentum is the tendency for the player in the lead to continue extending their lead with subsequent moves. For instance, Figure 8.13 shows White extending their lead with consecutive moves m_{13}, m_{15} and m_{17}, indicating that they carry the momentum over this period.

Within each game trial, all triplets of non-random and non-winning moves $\{m_{n-2}, m_n, m_{n+2}\}$ are found such that $m_{n-2} < m_n$, and the momentum calculated as the ratio of such triplets in which $m_n < m_{n+2}$. In other words, momentum is the ratio of consecutive lead increases for either player that are followed by a subsequent lead increase on their next turn.

$$A_{mom} = \sum_{g=1}^{G} \frac{count_{[M_{gr}+2 \le n \le .M_g-3]} \left(E_{co}(m_{n-2}) < E_{co}(m_n) \wedge E_{co}(m_n) < E_{co}(m_{n+2}) \right)}{count_{[M_{gr}+2 \le n \le .M_g-3]} \left(E_{co}(m_{n-2}) < E_{co}(m_n) \right)} / G$$

This category actually includes three related criteria. The momentum calculation for lead increases over two consecutive moves m_{n-2} and m_n is called the *momentum*(1) criterion. Criterion *momentum*(2) only performs this calculation for cases of lead increase over three consecutive moves m_{n-4}, m_{n-2} and m_n, and criterion *momentum*(3) only performs this calculation for cases of lead increase over four consecutive moves m_{n-6}, m_{n-4}, m_{n-2} and m_n. Longer lead increases exist with increasing rarity.

The average ratio over all such consecutive lead increases is stored per game trial, and the final result is the mean over all trials. These criteria return a floating point value in the range [0..1] where 0 indicates low tendency towards momentum and 1 indicates a high tendency towards momentum.

Correction (1 / 2 / 3)

Correction is the tendency for the lead to correct downwards following a consecutive lead increase by either player on their move. Correction is essentially the inverse of momentum.

For instance, Figure 8.14 shows White extending their lead with consecutive moves m_{13} and m_{15} and the subsequent downward lead correction at move m_{17}. Correction therefore describes the tendency for a game to normalise back towards a neutral position if either player starts to increase their lead.

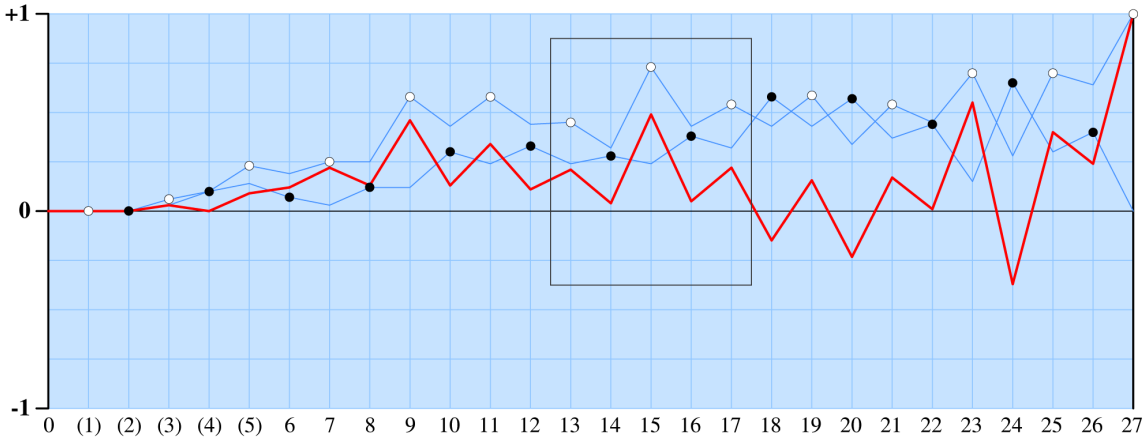


Figure 8.14 A lead increase by White and its subsequent correction.

Within each game trial, all triplets of non-random and non-winning moves $\{m_{n-2}, m_n, m_{n+2}\}$ are found such that $m_n > m_{n-2}$, and the correction calculated as the average lead decrease between m_n and m_{n+2}.

$$A_{corr} = -\sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+2}^{M_g-2} \left(E_{co}(m_n) - E_{co}(m_{n+2}) \right)}{count_{[M_{gr}+2 \le n \le M_g-3]} \left(E_{co}(m_n) > E_{co}(m_{n-2}) \right)} / G$$

This category actually includes three related criteria. The correction calculation for lead increases over two consecutive moves m_{n-2} and m_n is called the *correction*(1) criterion. Criterion *correction*(2) only performs this calculation for cases of lead increase over three consecutive moves m_{n-4}, m_{n-2} and m_n, and criterion *correction*(3) only performs this calculation for cases of lead increase over four consecutive moves m_{n-6}, m_{n-4}, m_{n-2} and m_n. Longer lead increases exist with increasing rarity.

Correction is measured differently to momentum (by value rather than by count) so that these two measurements provide different interpretations of the base concept rather than just negated or inverted values of each other.

The average correction over all such consecutive lead increases is stored per game trial, and the final result is the mean over all trials. These criteria return a floating point value in the range [-1..1] where -1 indicates maximum negative correction (i.e. momentum), 1 indicates maximum positive correction and 0 indicates neither.

Control

Control measures the degree to which the leading player controls play by limiting the move choices of the opponent. This is not based on the number of legal moves (branching factor) but rather the number of positive moves available to the opponent.

Control is generally seen as a positive quality that empowers the player [Kramer, 2000], maintaining their interest in the game by allowing them to direct its course. Control also rewards forward thinking through tactical sequences of moves that force the opponent into traps.

However too much control can be a bad thing, especially for the losing player, as it reduces the chance of a dramatic escape and means that the game is most likely headed for a predictable conclusion. Tavener [2007] points out that in this context it can be useful to consider each sequence of forcing

moves as a single composite move; players can then have the illusion of control on a sequence-by-sequence basis if not a move-by-move basis.

This measurement utilises the narrowness calculation to estimate the number of obviously good moves available to the opponent. For each non-random and non-winning move m_n at which the current player has a significant (0.1) lead, the difference between the narrowness in replies available to the opponent and the mean narrowness for that game is measured.

$$A_{ctrl} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g-1} \left(E_{co}(m_n) > 0.1 \begin{cases} narrowness(n) - (1 - A_{narr}) \\ 0 \end{cases} \right)}{count_{[M_{gr}+1 \le n \le M_g-1]} \left(E_{co}(m_n) > 0.1 \right)} / G$$

The average value over each game trial is stored, and the final result is the mean of all values. This criterion returns a floating point value in the range [-1..1] where -1 maximum indicates negative control and 1 indicates maximum positive control.

Foulup Factor

A game's *foulup factor* is the chance of a player making a serious mistake by overlooking a move [Tavener, 2007]. This is closely related to the clarity of a game, in addition to its obfuscation; this criterion measures the likelihood that the most obvious move is not the best one.

For each non-random and non-winning move, the average ratio of alternative move choices with a better shallow evaluation than the actual move made is determined. This indicates the likelihood that a player will be deceived by immediate evaluations into choosing a suboptimal move.

$$A_{foul} = \sum_{g=1}^{G} \frac{\sum_{n=M_{gr}+1}^{M_g-1} \frac{count\left(ES_c(m_n) \ge ED_c(m_n) \right)}{count(m_n)}}{M_{gi} - 1} / G$$

The average foulup estimate over all valid moves is stored for each game trial, and the final result is the mean over all trials. This criterion returns a floating point value in the range [0..1] where 0 indicates a low chance of foulup and 1 indicates a high chance of foulup.

Puzzle Quality

The *puzzle quality* is an estimate of the difficulty of puzzles created for the game. In keeping with Sam Loyd's observation that his goal was to compose puzzles whose solutions require a first move that is contrary to what 999 players out of 1000 would propose [Hayward & van Rijswijck, 2006], a puzzle's quality is measured as the counter-intuitiveness of its solution (the winning move).

The basic procedure used to create puzzles is as follows:

1. Play random game to 75% of average game length.
2. Continue playing with search depth ≥ 3 until game won.
3. Step back three moves to m_n.
4. Perform full game tree expansion to depth 3.
5. Mark leaf nodes:
   +1 = WIN
    0 = UNDECIDED
   -1 = LOSS
6. Propagate values back to m_n using minimax.

This gives the number of winning lines available to the current mover – the eventual winner – at move m_n. If there is a single winning line for the current mover at m_n then this position constitutes a "win in three" puzzle with a unique solution.

The puzzle quality is given by the number of moves with shallow evaluation better than the most highly valued winning line (essentially a discrepancy measure for puzzles):

$$A_{puzq} = \sum_{g=1}^{G} \frac{better(m_n)}{moves(m_n) - 1} / G$$

where $moves(m_n)$ returns the number of moves available to the current player at m_n and $better(m_n)$ returns the number of non-winning lines with shallow evaluation greater than or equal to the highest-valued winning line.

A measurement is made for each puzzle that the programme can construct (i.e. games of at least four moves with a clear winner) and the final result is the mean of these measurements. This criterion returns a floating point value in the range [0..1] where 0 indicates low puzzle quality and 1 indicates high puzzle quality.

This criterion is one of the slowest to measure; it requires a separate run of puzzle trials with a full game tree expansion to depth 3 in each.

8.4.2.2 Viability Criteria

Viability criteria indicate whether the game is essentially viable, that is, whether it is playable without any obviously serious flaws, and would provide a meaningful contest between competent players.

11 viability criteria are implemented. The measurements are typically made on the game outcomes, however game length is also included in this category as it is a function of the time at which any outcome occurs. Many are the same measurements as those described by Althöfer [2003].

These criteria are expected to be the most robust and reliable of those implemented. In order to describe them, it is useful to define some new terms:

$wins_W$ Total number of White wins over all trials for the current game.
$wins_B$ Total number of Black wins over all trials for the current game.
$wins_1$ Total number of first player wins over all trials for the current game.
$wins_2$ Total number of second player wins over all trials for the current game.
$wins_0$ Total number of draws or ties over all trials for the current game.
$wins_E$ Total number of wins for the expected player.

Completion

Completion is the tendency for games to reach a win or loss within a reasonable number of moves. Games that consistently fail to reach conclusive outcomes are generally unsatisfying.

Completion is measured as the ratio of games that reach completion within the prescribed maximum number of moves:

$$A_{comp} = (wins_W + wins_B) / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates lack of completion and 1 indicates a perfect completion record.

Balance

A game is *balanced* if players of each colour have an equal chance of winning. Unbalanced games can obviously be unsatisfying for the disadvantaged player.

Balance is measured as the absolute ratio of win difference between the two players, over all wins:

$$A_{bal} = 1 - \frac{|wins_W - wins_B|}{wins_W + wins_B}$$

This criterion returns a floating point value in the range [0..1] where 0 indicates lack of balance and 1 indicates perfect balance.

Advantage

This criterion estimates any inherent *advantage* in a game, that is, the tendency for the first player (or sometimes the second) to win more than their fair share of games.

Althöfer [2003] describes this criterion as *balanced chances*.

Advantage is measured as the absolute ratio of games that the first player wins above or below the expected 50% of games:

$$A_{adv} = \frac{|wins_1 - (wins_W + wins_B)/2|}{(wins_W + wins_B)/2}$$

This criterion returns a floating point value in the range [0..1] where 0 indicates no advantage (first and second mover each win 50% of games) and 1 indicates strong advantage (first or second mover always win).

Drawishness

Drawishness is the tendency for a game to end in draws. This includes both ties (more than one winner) and draws (no winner). Technically draws should be included in the completion criterion as a draw constitutes a completed game, however it is worth keeping this information separate.

Althöfer [2003] describes this criterion as the *drawing quota*.

Drawishness is given by the ratio of drawn games over all games:

$$A_{draw} = wins_0 / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates lack of drawishness and 1 indicates maximum drawishness.

Timeouts

The *timeout* criterion is a measure of the tendency for a game to fail to reach any sort of conclusive outcome (win, loss, draw) before the prescribed move limit is reached. An excessive number of timeouts indicate that a game may have serious flaws, such as goals that can never be reached.

The timeout value is the ratio of games that do not end in a draw or a win for either player:

$$A_{tout} = 1 - (wins_0 + wins_1 + wins_2) / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates no timeouts and 1 indicates that all games time out.

Duration

Duration is a measure of the average number of moves required to complete a game. Although the length of each game in a set of trials will vary depending on the number of killer moves, miscalculations, dramatic recoveries and so on, ideally each game's duration should not deviate too far from the user's preferred number of moves, which is assumed to be half of the maximum specified game length.

Game duration is useful for detecting pathological flaws in games in both direction; trivial games that end within a few moves, and excessively long games that are difficult to conclude. Game length is one of Althöfer's criteria [2003].

Duration is measured as the average absolute deviation in the number of moves from the preferred number of moves per game, over all games:

$$A_{durn} = \sum_{g=1}^{G} \frac{\left| M_{pref} - M_g \right|}{M_{pref}} / G$$

This criterion stores a measurement for each game, and the final result is the mean over all games. The name is somewhat counterintuitive as larger values do not necessarily indicate longer duration but greater deviation from the preferred game length.

It returns a floating point value in the range [0..1] where 0 indicates no deviation from the preferred number of moves and 1 indicates maximum deviation from the expected number of moves.

Depth (Skill Level)

Recall that a game is generally understood to be *deep* if human players are capable of playing it at many different skill levels [Thompson, 2000]. It can be assumed that a stronger player will generally beat a weaker player. The likelihood of skill levels emerging may be indicated by observing whether a deeper-searching computer player generally outperforms an opponent making a shallower search, the analogy being that the player who searches deeper in the game tree has a deeper understanding of the game.

The basic tenet of depth is that the more a player studies and understands a board position, the better the move they will make.

This measurement is somewhat dubious as it is generally known that deeper search improves the strength of computer play, a phenomenon known as *deepening positivity* [Althöfer, 2003]. However, the degree of deepening positivity may yield some insights.

Depth (skill level) is measured by reducing the search depth of one of the players by one ply, and it is expected that the player searching deeper should generally win more games. The reason that the search depth is not reduced by a more significant margin is to allow the shallower player to occasionally win a game if the random starting position proves favourable; it would not impart much information if the deeper player won every game. On the other hand, if there is little to choose between the deep and shallow players then the game is most certainly devoid of depth and of little interest.

The final value is the ratio of expected wins over all games:

$$A_{skil} = wins_E / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates the (unlikely) ascendancy of shallow play and 1 indicates the ascendancy of deeper play.

This criterion is slow to measure as it requires an additional set of trial games between unbalanced opponents, however the fact that one player searches to a shallower ply somewhat reduces this speed penalty.

Selfishness

A *selfish* player is one who concentrates exclusively on their own piece development each turn. If such a player wins any games against a balanced opponent, this may point towards a lack of interaction in the game.

Susceptibility to selfish play is measured by unbalancing one of the computer players to base their board evaluations exclusively on their own pieces; it is expected that this player should generally lose against a balanced opponent. The final value is the ratio of expected wins over all games:

$$A_{self} = wins_E / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates susceptibility to selfishness and 1 indicates immunity to selfishness. Note that these values are somewhat counterintuitive; the expected outcome is rated more highly, not the susceptibility to selfishness.

This criterion is slow to measure as it requires an additional set of trial games between unbalanced opponents.

Obstructive Play

An *obstructive* player is one who concentrates exclusively on their opponent's piece development each turn. If such a player wins any games against a balanced opponent, this may point towards a serious flaw in the game if a player can consistently win by impeding the opponent. Note that this measure is less important for zero-sum games with mutually exclusive goals, such as Hex, in which a bad position for one player implies a good position for the opponent.

Susceptibility to obstructive play is measured by unbalancing one of the computer players to base their board evaluations exclusively on the opponent's pieces; it is expected that this player should

generally lose against a balanced opponent. The final value is the ratio of expected wins over all games:

$$A_{obst} = wins_E / G$$

The Ludi player does not perform ko tests for move repetition for the obstructive player during game measurement trials, as it should be the obstructive player's intent to block the opponent.

This criterion returns a floating point value in the range [0..1] where 0 indicates susceptibility to obstructive play and 1 indicates immunity to obstructive play. Note that these values are somewhat counterintuitive; the expected outcome is rated more highly, not the susceptibility to obstructive play.

This criterion is slow to measure as it requires an additional set of trial games between unbalanced opponents.

Resilience

Resilience is the tendency for a game to withstand random moves from one of the players. If a random player wins *any* games against a balanced opponent, this may point to serious flaws either in the game or in the algorithm used to determine moves. For instance, random play would suit games that are so opaque that any move seems as good to any other, regardless of the player's skill. Susceptibility to random play means that players do not need to put much thought into each move and are therefore unlikely to engage intellectually with the game. This criterion measures the game's theoretical resilience to one of Borel's [1913] dactylographic monkeys.

Resilience to random play is measured by setting the search depth of one of the players to zero. It is expected that the zero depth player should always lose against a balanced opponent with search depth greater than zero. The final value is ratio of expected wins over all games:

$$A_{res} = wins_E / G$$

This criterion returns a floating point value in the range [0..1] where 0 indicates susceptibility to random play and 1 indicates immunity to random play.

This criterion is somewhat slow to measure as it requires an additional set of trial games between unbalanced opponents, however the fact that one of the player makes random moves makes this the fastest of the slow criteria.

Puzzle Potential

The *puzzle potential* is an estimate of the ease with which puzzles may be created for the game. An inability to create puzzles may point to a flawed game [Thompson, 2000].

"Win in three" puzzles are created using the algorithm described above for puzzle potential (both criteria are in fact measured at the same time) and the puzzle potential is given by:

$$A_{puzp} = \sum_{g=1}^{G} \frac{\min(1, \log_{10}(moves(m_n)))}{2^{lines(m_n)-1}} / G$$

where $moves(m_n)$ returns the number of moves available to the current player at m_n and $lines(m_n)$ returns the number of these moves that are winning lines. This equation returns a value of 1 for a single winning line, and this value is successively halved for each alternative winning line. The

logarithm component has the effect of reducing this value further if there are less than 10 moves to choose from, as this makes the puzzle increasingly easier.

A measurement is made each time that the computer player can successfully play a game to a win for either player in at least four moves, and the final result is the mean of these measurements. This criterion returns a floating point value in the range [0..1] where 0 indicates low puzzle potential and 1 indicates high puzzle potential.

This criterion is one of the slowest to measure; it requires a separate run of trials with a full game tree expansion to depth 3 in each trial.

8.4.3 Other Criteria

Although a large number of game measurements have been implemented, these are only indicative of the type of measurements that may be made and demonstrate how such measurements may be applied; the above list is by no means exhaustive.

If the general game player were adapted for UCT move planning, then it is speculated that the statistics embodied in the UCT tree during the search for each move could give valuable additional aesthetic information about a game. In particular, reliable measurements of a game's clarity and perhaps even its depth might be deduced by the "shape" of the search over a large number of playouts from each board position.

8.5 Summary

The Criticism module handles the task of measuring games according to a set of criteria and providing an aesthetic response to each game. Each aesthetic score is a prediction of the likelihood that human players will find the game interesting.

The basic game model is expanded to a player-centric model which allows the definition of intrinsic and extrinsic categories of aesthetic criteria, and further subcategories of quality and viability criteria. Move and lead histories are introduced as methods for analysing games in this context.

A total of 57 aesthetic criteria are implemented, as follows:

- 16 × Intrinsic criteria that directly measure aspects of the rules,
- 30 × Quality criteria that measure trends in play, and
- 11 × Viability criteria that measure game outcomes.

Aesthetic measurements are made over several self-play trials. Most measurements are performed during an initial set of trials played between balanced computer opponents (fast), then additional sets of trials are performed for measurements requiring mismatched opponents (slow). The final aesthetic value is the sum of aesthetic measurements weighted according to human player preference score correlations obtained experimentally.

Chapter 9

Synthesis Module

> Some part of a mistake is always correct.
> – Savielly Tartakover
>
> Optimisation hinders evolution.
> – Alan J. Perlis

9.1 Introduction

The Synthesis module is responsible for creating new games through the evolution of existing rule sets using genetic programming techniques. This chapter describes the overall game creation process with a worked example, and highlights key implementation decisions specific to this task.

9.2 Model

The game synthesis process follows a modified evolutionary algorithm, as illustrated in Figure 9.1 and described in pseudocode in Listing 9.1.

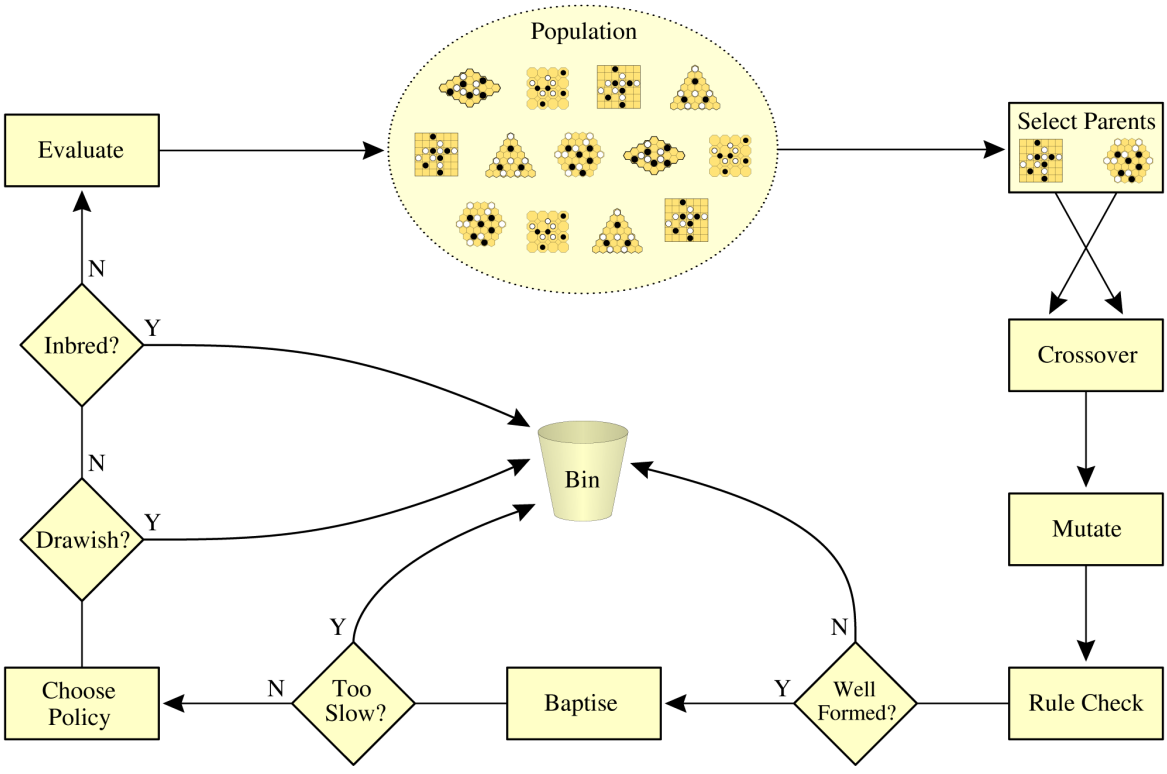


Figure 9.1 The game life cycle.

```
set initial population
repeat (until N children created)
{
      select parents {A, B}
      repeat (until valid child or T attempts)
      {
            recombine {A, B} -> child C
            mutate C
            check C for rule safety

            if (C not well formed)
                  cull C

            baptise C

            if (C too slow)
                  cull C

            choose policy

            if (C drawish)
                  cull C

            if (C inbred)
                  cull C
      }
      measure C for fitness
      add C to population
}
remeasure viable children
```

Listing 9.1 The game evolution process.

Starting with a population of known games, pairs of parents are selected and bred to create child games, which are then checked for rule safety, speed, drawishness and inbreeding, measured for fitness and added to the population. This process is repeated until the specified number of child games *N* is created. A default value of *N*=100 is used; this allows the user to create a large number of child games while monitoring the evolutionary progress in stages.

This process is similar to the standard evolutionary algorithm as described in Section 3.3, except that all valid offspring are added to the population; the fitness function is used to determine their place within the population rather than their survival. This encourages greater genetic diversity and means that even low quality children will occasionally contribute to future generations, in case they harbour partial rule combinations that prove fruitful in other contexts. The importance of this approach is borne out in the experimental results.

It is worth noting that the task of game synthesis is different to most applications of evolutionary algorithms as the aim is to produce a wide number of diverse solutions (games) rather than producing a single optimal solution. The emphasis is therefore on optimising the evolutionary process for speed in order to produce the widest range of games possible within a given amount of time.

This module constitutes the synthesis algorithm of Stiny & Gips' [1978] design system in their algorithmic aesthetics model. Unlike previous systems for creating games, such as Pell's METAGAMER [1993b], this process operates in an *evocative* (directed by desired response) rather than *constructive* (rule-driven) mode. This makes it more likely that unexpected rule combinations and behaviours will emerge.

In terms of the evolutionary analogy, a game's *genotype* is the structured set of ludemes comprising its rule set, and its *phenotype* is the game itself, manifested as the move space defined by this rule set and the resulting play.

The task of game synthesis is essentially the search for novel and harmonious rule combinations. Like Dawkins' memes, ludemes are susceptible to variation or distortion and will have to compete with themselves, as well as others, for attention [Hofstadter, 1985].

The steps in this process are now described in detail, with a worked example.

9.3 Initial Population

In order to increase the chance of meaningful games being created, it is useful to seed the initial population with games that either:

- Are playable, in order to propagate good structure, or
- Contain interesting rule combinations, in order to propagate move mechanics.

The initial population consists of the 79 games used for the Experiment I survey, as listed in Appendix C. These games all have functional policies and are generally viable (playable) but differ in quality.

The members of the initial population are sorted by user preference score (obtained in Experiment I) and constitute the ancestors from which all child games are derived. New games, as they are created, are inserted into the population according to their predicted preference score; the population remains sorted at all times.

9.4 Parent Selection

For each iteration of the game synthesis cycle, two parents are chosen from the population using a *stochastic universal sampling* method [Baker, 1987]. This sampling method prefers higher-valued individuals but still considers lower-valued individuals with decreasing probability, hence encourages genetic diversity while maintaining some standard of fitness. This approach reduces the likelihood that the evolutionary process will become elitist or the population too inbred.

Each pair of parents have up to T attempts (default value T=10) to produce a valid child. If no valid child is produced by this point, then another pair of parents is selected and the attempt counter reset. This process is repeated until N valid children have been created and the evolutionary process terminates.

For example, consider that the two parent games selected by stochastic universal sampling are Tic Tac Toe and Y. Figure 9.2 shows the rule sets and corresponding ludeme trees for these two games.

```
(ludeme TicTacToe
    (players White Black
    (board
        (tiling square i-nbors)
        (shape square)
        (size 3 3)
    )
    (end  (All win (in-a-row 3) ) )
)
```

```
(ludeme Y
    (players White Black)
    (board
        (tiling hex)
        (shape tri)
        (size 11)
        (regions all-sides)
     )
    (end  (All win (connect all-sides) ) )
)
```

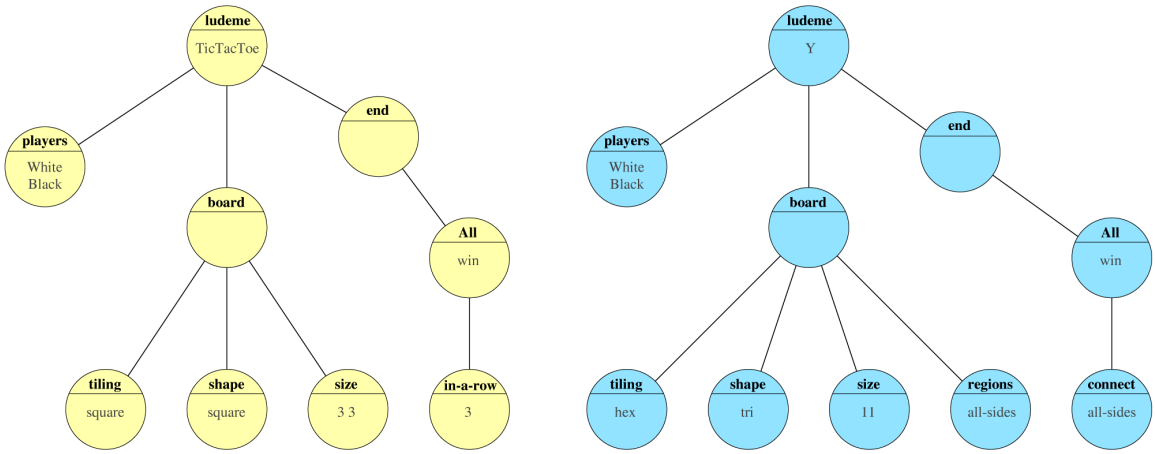


Figure 9.2 Selected pair of parent games: Tic Tac Toe and Y.

9.5 Recombination

The first stage in generating a child game is to recombine the ludeme trees of the two parents into a single ludeme tree.

One of the parents is chosen at random to form the template of the child, as is the case with most genetic programming applications [Langdon, 1996]. Each element of this template is then visited and marked for crossover with a specified likelihood (default value 10%). For each element marked for crossover, a compatible element is found randomly from either parent and the crossover made; elements of a parent can therefore cross over with other elements of the same parent.

Special care is taken when crossing over the key elements *players, board, pieces, start, play* and *end* to ensure that the essential structure of the child rule set is correct.

Figure 9.3 shows a typical recombination of the two example games, Tic Tac Toe and Y, into a single offspring called Child.

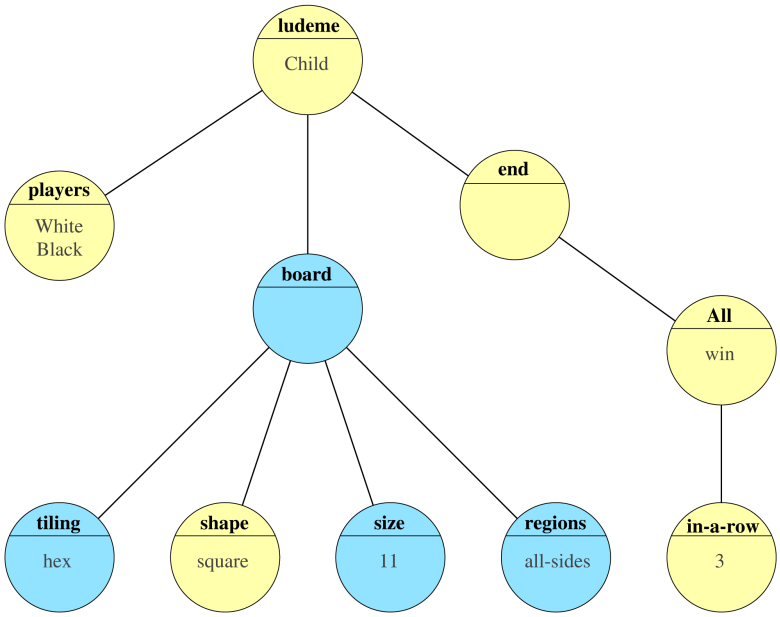


Figure 9.3 Recombining the parents into a single Child.

It can be seen that Tic Tac Toe was selected as the child's template, its *board* element replaced by the corresponding *board* element from the other parent Y, and the *shape* element of this modified board replaced with the equivalent *shape* element from the original parent Tic Tac Toe.

9.6 Mutation

Each element of the child is then visited and mutated with a specified likelihood (default value 10%). Each mutation may involve:

- Changing the element type,
- Removing a subelement,
- Adding a subelement,
- Changing an attribute value,
- Removing an attribute, and/or
- Adding an attribute.

Some subelement and attribute mutations are constrained by the context of the current element, and some element type changes are constrained by the context of the parent element. In particular, there is a separate mutation routine for each of the main element types (*players, board, pieces, start, play* and *end*) in order to maximise the chance of mutations producing well-formed rule sets. Mutations are otherwise unconstrained, for the sake of genetic diversity.

When changing an element type, the new type is selected using stochastic universal sampling with frequencies based on predefined distribution tables for each element type. Element frequencies are generally evenly distributed apart from:

- Trapeziums are less frequent than other hexagonal board shapes,
- The ratio of *win*/*lose*/*draw* outcome qualifiers is 8:4:1,
- The no-move end type is ¼ as likely to occur as other types, and
- The ratio of *and*/*or*/*if*/*not*/*xor* logical operators is 4:4:1:1:1.

Special care is taken to maintain the correct tree structure when mutations involve any of the logical operators {*and/or/if/not/xor*}. For instance, the removal of an *or* rule will remove all of its arguments, and the addition of an *and* rule will randomly create extra rule arguments as necessary.

If an element is changed to a logical operator, then that element becomes the operator's first argument. For instance (*group*) might mutate to (*not* (*group*)) or (*xor* (*group*) (*no-move*)).

Starting positions are handled separately as games generally benefit from symmetrical, balanced starting positions which are unlikely to occur at random. Most *place* mutations are therefore symmetrical and include predefined patterns such as opposed edges with a random number of rows or opposed corners (first and last cell), although some random placements are also made for the sake of diversity.

End condition mutation is constrained to generally favour balanced end conditions, although some unbalance in end conditions is also allowed.

Figure 9.4 shows two typical mutations applied to the example child game. Firstly, the board *size* attribute has been changed from 11 to 7. Secondly, the complete *in-a-row* element has been replaced with a *group* element with an attribute of 12.

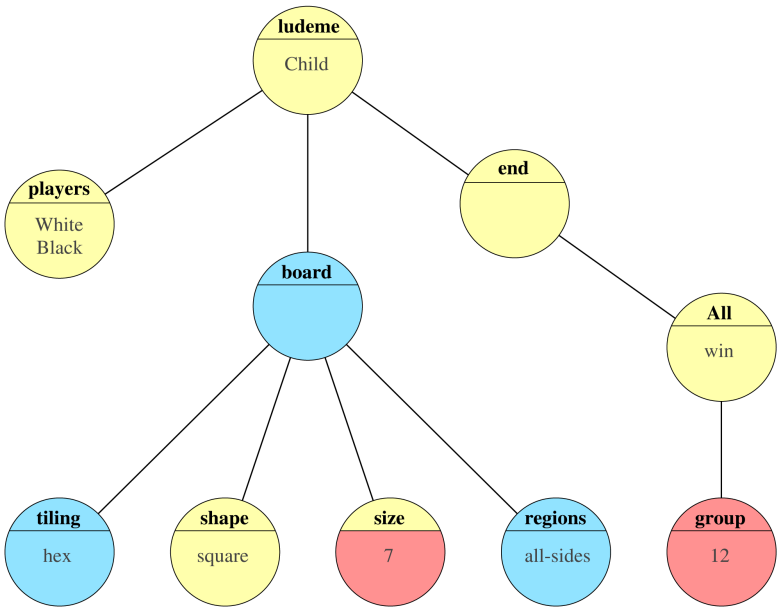


Figure 9.4 Mutating the child.

9.7 Rule Safety

Given that the recombination and mutation processes are essentially random (within the specified constraints) it is possible that the child game will contain malformed rules that violate the GDL. The child's ludeme tree therefore undergoes some preliminary safety checks before being subjected to the ultimate test for rule safety; whether it can be successfully instantiated as a game object.

Rule Clashes

Each of the key elements *players, board, pieces, start, play*, and *end* are parsed and tested for rule safety within their own context, and any immediate errors corrected. For example, suitable values are randomly generated for end conditions missing required attributes.

Consider the example child game generated so far:

```
(ludeme Child
    (players White Black
    (board
     (tiling hex)
              (shape square)
              (size 7)
              (regions  all-sides)
    )
    (end (All win (group 12) ) )
)
```

Although this rule set appears well-formed according to the GDL, it actually contains a *rule clash* in that *square* boards are not defined for *hex* tilings (Figure 9.5 left).

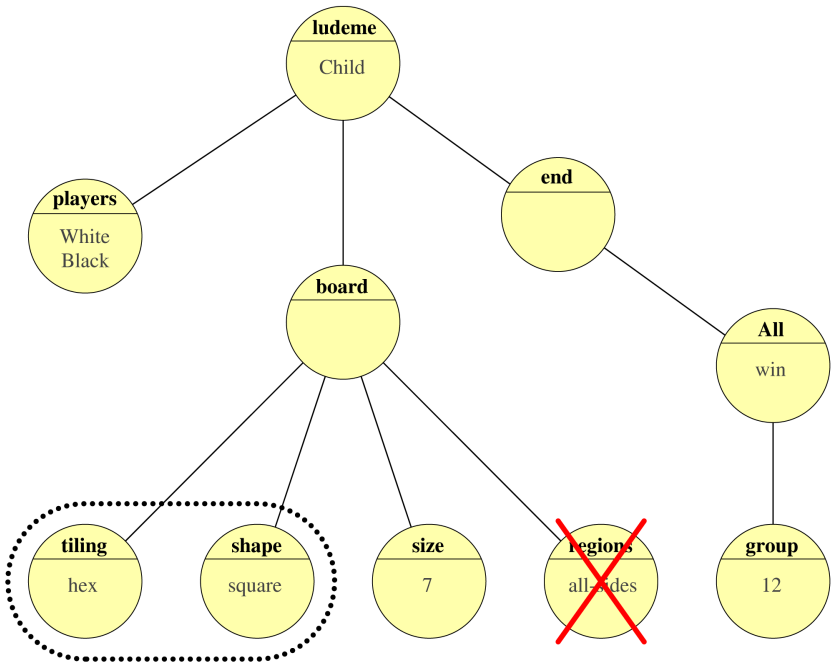


Figure 9.5 A rule clash (left) and a vestigial rule (right).

This example highlights how fragile rule sets are to change. However, it is a simple matter to correct this rule clash by changing the board shape attribute to a suitable value such as *rhombus*, as follows:

```
(ludeme Child
    (players White Black
    (board
     (tiling hex)
          (shape rhombus)
          (size 7)
          (regions  all-sides)
    )
    (end (All win (group 12) ) )
)
```

Suboptimal Rules

Suboptimal rules are those rules of the child game that are correct according to the GDL but that contain aspects which are superfluous or irrelevant to the game. By implication, suboptimal rules can be optimised to more concise and elegant forms or removed altogether without affecting the game.

The example child game contains a *vestigial rule* in the form of the (*regions all-sides*) element (Figure 9.5, right). This element is a throwback to the target regions of the parent game Y that are no longer relevant to the child game. Vestigial rules typically involve:

- Board regions that are no longer relevant,
- Superseded end conditions,
- Unplayable pieces,
- Specific piece movements no longer relevant to a new board topology, etc.

Other types of suboptimal rules include *superfluous rules* that may result in duplication of effort in legal move calculation or win detection. For instance, the superfluous rule (*or* (*true*) (*true*)) could be safely optimised to the rule (*true*).

Similarly, *impossible rules* that describe conditions that can never possibly occur may result in unnecessary computation. For instance, the impossible rule (*and* (*empty to*) (*not* (*empty to*))) may be removed without affecting the game. Cazenave describes the use of *impossibility metarules* that automatically detect and remove rules describing conditions that can never be met [1999].

Rule Optimisation

It is tempting to handle such suboptimal rules at this point in order to simplify and optimise the child games' rule sets. Minimal rule sets lead to "refined, tight, classy little games" [Costikyan, 2005] and are generally more elegant and easier for human readers to interpret.

However, preliminary tests revealed the importance of suboptimal rules to the success of the evolutionary process; they often act as dormant rules which pass unused through generations of games, only to reach their full potential in combination with other rules in later generations. It is unlikely that truly innovative rule combinations will emerge, fully formed, from the crossover and mutation of minimal rule sets.

In addition, it can be difficult to gauge the impact that culling an apparently superfluous rule will have on a game. For instance, consider the vestigial piece definition (*King*) which contains no movement rules. This is most likely a superfluous rule that can be safely culled, unless the game starts with a King on the board in a crucial position in which case its removal would significantly alter the game. Similarly, some rules that do not directly result in any legal moves may instead act as deterrents that have an indirect effect on play. Complete tests for rule importance are complex and potentially expensive, and such optimisations are best left as a post-processing step when greater attention may be paid to a small number of the most promising games.

Suboptimal rules are therefore treated as partial rule stubs that encourage genetic diversity and retained if they do not actually break the game. This design choice was validated by the number of serendipitous rule combinations that emerged during evolution which would have been extremely unlikely to occur with stricter rule optimisation; Yavalath, game #2 in Section 12.4.1, provides a striking example.

Size Limit

Results from Experiment II suggest that board size is not a significant indicator of game quality. Given that board size is not a critical factor and that smaller boards significantly reduce self-play times for policy optimisation and game measurement, the breeding process aggressively prefers games with smaller board sizes.

Any board found to contain more than 64 cells is automatically reduced in size. This allows the following maximum board sizes:

- 8x8 square,
- 11x11 rhombus,
- 11-per-side triangular,
- 5-per-side hexagonal, and
- 6-per-side trapezium.

These board sizes are more than sufficient for meaningful play.

Game Assembly

A game object is then assembled from the child ludeme. If this assembly succeeds then the game is rule safe and the process continues, else the child is discarded and the mating process repeated to create a new child. Using the rule parsing capabilities of the game object in this way saves a great deal of tedious rule validation.

Note that there has been no rule culling or optimisation to this point; any rule adjustments made to this point have been purely to make the game rule safe.

As an aside, it was interesting to note the effectiveness of randomly generated rule sets in detecting obscure errors in the code. Although robustness to arbitrary rule combinations was a high priority during implementation, it was inevitable that some assumptions based on well-formed games should be made in a system of this complexity. Subjecting an application to randomly generated input is a useful method of stress testing.

9.8 Baptism

If a game survives to this point, it is given a short name unique to the current population. Each name is generated using a Markov chain algorithm which, starting with a random character, assembles a string character-by-character based upon letter combination frequencies to be found in an input list of source names [Kernighan & Pike, 1999].

Given a list of source words, letter combinations of up to three characters in length are stored and their frequency noted. The input list used for the evolved games was a list of Tolkien-style names distributed with a public domain computer game [Greene, 2007]. This list proved a good choice as it only contains 601 entries yet results in short, well-formed and interesting names that are unlikely to be shared by any existing games. Other word lists were tried, including dictionaries of classical Latin and Greek words, but these proved orders of magnitude larger and generally provided less satisfactory results.

Typical names for games generated by process include:

- Oroth,
- Galdal,
- Ethorond,
- Kemeneth,
- Valindor,
- Bered, and
- Mor.

In the event of name clashes when merging populations from evolutionary runs on different machines, it is a simple matter to rename the offending games. Ludi provides a “Generate Name” feature that generates new names unique to the currently loaded population.

9.9 Speed Test

Games are tested for speed and discarded if too slow. This is to stop the process of evolution from getting stalled for hours on a single individual at the expense of many other individuals that may have been created in that time.

Using the child game’s default policy, a small number of moves are made at search plies 1, 2, 3 and 4. If any of these moves exceed a user defined time limit then the game is deemed too slow and

discarded. A default time limit of 15 seconds per move is used; this is rather short, but did not stop the emergence of several interesting games in Experiment III.

A downside of the speed test is that it may eliminate complex games with large boards and large branching factors, which may be the preferred type of game for many players. Obvious improvements to this somewhat draconian measure might involve iteratively reducing the board size and piece count downwards until an acceptable speed is reached.

9.10 Policy Choice

The game then undergoes perfunctory policy optimisation to find a workable policy within a short amount of time.

Parental Advice

As full policy optimisation is time consuming and not always successful, it is preferable to instead leverage the policy information available from the parents; these are a good starting point given that the parent policies were optimised for the rule sets from which the child was derived.

There are therefore three readily available policies likely to be relevant to the child game:

- The child's default policy,
- Parent A's policy, and
- Parent B's policy.

These policies may be combined by adding the component weights to produce *hybrid policies*, giving a total of seven policy combinations likely to be relevant to the child:

- Default,
- Parent A,
- Parent B,
- Parent A + parent B,
- Parent A + default,
- Parent B + default, and
- Parent A + parent B + default.

The weighting values of two policies can be safely added as they are relative; it is their ratio that is important, not their absolute values.

A mini-tournament of policies is conducted to determine which of these choices is best. Each of the 7 candidate policies plays each other twice (one start each) in a round robin format, giving 42 games in total. The Strategist module is used to coordinate the self-play tournament.

	D	A	B	A+B	A+D	B+D	A+B+D
D	x						
A		x					
B			x				
A+B				x			
A+D					x		
B+D						x	
A+B+D							x

Table 9.1 Mini-tournament for default, parent and hybrid policies.

The chosen policy is unlikely to be optimal for the child game. However, its function at this stage of the evolutionary process is to allow the completion of legal games for measurement purposes rather than achieving a high standard of play.

Completion Test

If more than half of the match games fail to produce a winner before the move limit is reached then the game is deemed to be *drawish*. This indicates a serious flaw in either the rule set or the game's chosen policy and the child is discarded.

Speed tests also continue to be made throughout the policy tournament. If any move exceeds twice the time limit then the game is deemed too slow and discarded. The time limit is doubled to take into account the fact that more complex board positions generally occur after the first few moves of a game.

9.11 Inbreeding

The distance of the child game from each member of the population is then measured, and the child game culled if it is not sufficiently distinct. This method of incest prevention [Eshelman & Schaffer, 1991] reduces the likelihood of inbreeding that may otherwise flood the population with similar games, reducing the potential for genetic diversity. Uncertainty and surprise – two key aspects of game play – are just as important in game creation, as players want new and interesting challenges.

The first step in measuring the distance between two games is to weight each component rule according to its relative importance:

- Elements are more important than attributes,
- Higher level elements are more important than lower level elements, and
- End condition elements are more important than other types of elements.

Specifically, elements and attributes are weighted according to their depth d in the ludeme tree as follows:

- Attributes: $1/d$.
- Elements: $2/d$.

These values are doubled for end condition elements and attributes to emphasise their relative importance. Support elements (*advisors*, *ancestry, ranking*, *score, description* and *aim*) and support attributes (game name, piece names, arguments, variable placeholders, etc) are ignored.

The total discrepancies in elements ($discrep_e$) and attributes ($discrep_a$) are then calculated by accumulating for each ludeme tree the values of those elements and attributes that cannot be matched in the other tree, and the overall distance between the two games given by:

$$dist = \min\left(1, \log_{10}\left(discrep_e + discrep_a\right)/2\right)$$

Suboptimal rules may falsely exaggerate this distance by including superfluous differences, so it is good to standardise two rule sets as much as possible before comparing them. Missing *pieces* elements are therefore expanded to their default values in all cases, and empty *start* elements excluded from the discrepancy measurements.

The earlier decision to retain suboptimal but legal rules in child games means that most distance measurements involving evolved games will be exaggerated, hence it does not make sense to be overly zealous in culling inbred games. Child games are therefore only culled if the distance to the

closest member of the population is zero; in other words, if the child is a duplicate of an existing game.

9.12 Aesthetic Fitness

The child game has at this point survived all critical tests, and is measured for fitness and inserted at the appropriate place in the population. The fitness value is the game's predicted aesthetic score, as determined by the Criticism module using the set of 16 best predictors (see Section 11.4). Pareto scoring is not used as it is based on the individual's component elements (genotype) whereas it is more important to measure the rule set as a functioning whole (phenotype).

The aesthetic measurements made during this calculation are stored to disk for future use, although they should only be considered approximate as the child game's policy is unlikely to be optimal, leading to suboptimal and possibly uncharacteristic trends in play. These preliminary estimates will be superseded by more accurate measurements for the most promising evolved games.

Viability measurements, however, are more robust and provide useful results even in the presence of suboptimal policies. In particular, the following four criteria were found to be of particular importance in identifying games of interest in Experiment III (Section 12.4):

- *Completion*: Most games reach a conclusion.
- *Balance*: No significant advantage to either player.
- *Advantage*: No significant first move advantage.
- *Duration*: Games end in a reasonable number of moves.

The duration measurement was found to be especially useful; it distinguishes games that provide a meaningful competition between players over a number of moves rather than trivial games that finish after the second or third move.

This information is available from the stored game measurements, and a simple viability test is performed on each game as follows:

```
if
(
      completion > 0.5
      &&
      balance    > 0.5
      &&
      advantage  < 0.5
      &&
      duration   < 0.5
)
{
      // game is viable
}
```

Listing 9.2 Pseudocode for viability testing.

Games that pass this test are marked for future reference by inserting a support ludeme (*viable* 1) in their rule set. This viability test is equivalent to the reference component of Stiny & Gips's [1978] algorithmic aesthetics system and returns a boolean value indicating whether a rule set actually constitutes a valid game, that is, whether the game is complete, correct and provides a meaningful contest.

A small "baby bonus" (default value 0.1) was initially added to the child game scores to promote newer games within the population in the interest of genetic diversity. However, this soon proved

unnecessary as child games regularly outscored their ancestors due to the approximate nature of the preliminary aesthetic measurements.

9.13 Final Population

After the completion of the evolutionary process, the final population consists of the initial set of source games plus a number of evolved child games with valid rule sets but mostly suboptimal policies, sorted by approximate aesthetic score. Most of the evolved games will in fact be barely playable – if at all – as might be expected from what is essentially random manipulation of fragile rule sets.

Further attention is then paid to those games identified as viable; their policies are further optimised and their aesthetic scores remeasured to give a final, more accurate result. This small set of viable games (typically less than 2% of the population) constitutes the end product of the evolutionary process, and the remainder may be kept as genetic material for further evolutionary runs or safely discarded.

Further details of the actual evolutionary run performed for Experiment III are given in Chapter 12, including validation and analysis of the resulting evolved games.

9.14 Summary

The Synthesis module is responsible for creating new games through the evolution of existing rule sets. Starting with an initial population of known games, pairs of parents are selected (with a bias towards fitter individuals) and standard genetic programming techniques applied to recombine them to produce child games.

Child games are checked for rule safety, given unique names, tested for speed, and basic policies determined. Rule clashes in child games are corrected but superfluous legal rules deliberately retained in order to encourage the formation of innovative but functional rules in future generations.

Child games are tested against the current population to for inbreeding, measured for aesthetic fitness and viability, and inserted into the population. Games marked as viable are then further optimised and remeasured to produce the final set of evolved games.

Part III

Results

One measurement is worth fifty expert opinions.
– Howard Sutherland

Overview

Part III presents the results of three experiments conducted to test the proposed system.

Experiment I takes the form of a survey to determine human player preferences for a data set of existing games. The induced ranking scheme provides a reference for game quality.

Experiment II involves the correlation of aesthetic measurements of this data set of games with their player preference scores, in order to provide a predictor set of aesthetic criteria with which preference scores may be predicted for new games. This experiment tests the first hypothesis: *that there exist fundamental (and measurable) indicators of quality for combinatorial game design*.

Experiment III involves the evolution of new games from the initial data set. A small number of viable evolved games are identified and a follow-up survey conducted to determine whether the predicted scores for these games reflect actual player preferences. This experiment tests the second hypothesis: *that these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games*.

Chapter 10

Experiment I: Game Ranking

> The people have spoken. I am their leader. I must follow them.
> – Jim Hacker, PM, *Yes, Minister*

10.1 Introduction

Experiment I is designed to establish a reliable scale of game quality, which can then be used to gauge the effectiveness of the aesthetic measurements conducted by the Criticism module.

This experiment consists of two parts:

- A survey to gather user preferences between pairs of games, and
- The induction of game rankings from these paired comparisons over a set of games.

10.2 Aim of the Experiment

In order to determine which aesthetic criteria are relevant to game quality, it is first necessary to distinguish good games from bad ones. Good games in this context are deemed to be those that interest human players, and bad games those that fail to engage the interest of human players.

The purpose of this experiment is to rank a number of test games from least preferred to most preferred based on the preferences of human players, in order to provide a concrete reference point for perceived game quality. A number of game measurements will then be made and correlations sought between game measurements and human player preferences; the stronger the correlation, the more useful the game measurements will be as predictors of the perceived quality of a game.

10.3 Method

Experiment I consists of a survey conducted in order to determine human rankings of a data set of games.

Ethical clearance to conduct this survey and a brief follow-up survey was obtained from QUT's University Human Research Ethics Committee (QUT Ref 3999H). The survey is classified as a "Level 1 (Low Risk)" anonymous questionnaire suitable for the general public, provided that participants are at least 18 years of age.

Data Set

The data set consists of the 79 games listed in Appendix C. Each game is referred to by index (000..078) rather than name to discourage survey participants from forming preconceptions about games based on their names. Indices are zero-based, so the description “game 017” refers to the 18th game in this set.

Most of the source games were taken from the collection of test games developed during the implementation of the Ludi player. Most were viable (workable) games that played reasonably well,

although several rule sets with pathological flaws were added to broaden the source set's coverage of the aesthetic design space.

11 of the source games were popularly known games, or at least close approximations of them:

- Breakthrough,
- Chameleon (11x11),
- Chess*,
- Fox & Geese*,
- Gomoku (5x5)*,
- Gomoku (11x11)*,
- Gomoku (19x19)*,
- Hex (17x17),
- Tic Tac Toe,
- Unlur*, and
- Y (11 per side).

The games marked with asterisks are not standard versions of the known games. For instance, the version of Chess does not include castling moves, en passant moves or stalemate tests, the version of Fox & Geese is played on a square board, the versions of Gomoku are played on different board sizes, and the version of Unlur does not support the contract phase. It is worth noting that these changes result in games that may look familiar to some players but may in fact be significantly different in nature. These examples indicate limitations imposed by the scope of the Ludi GDL and highlight a disadvantage of the GDL approach in general; in that it is difficult to foresee, define and implement all rules that will be required.

The policies for the source games were optimised at this point to ensure that the computer player could play each game at a level sufficient to at least challenge human opponents, otherwise they may not engage with the game and fully understand it. It was during this stage that the need for the NULL policy and ko cycle testing emerged. Many of the games failed to improve beyond their default policies, and the computer player was noticeably stronger at some types of games (the *N*-in-a-row games in particular) than in others.

Subjects

Survey participants were recruited via email by an invitation posted to online board game communities, relevant newsgroups, and QUT game design and AI courses. The majority of subjects (approximately 80%) were recruited through the Gamerz.net online board game community [Rognlie, 1996].

Each individual who replied to the invitation was asked an initial set of preliminary questions:

- Are you at least 18 years of age?
- Are you male or female?
- Approximately how often do you play board games?
- Approximately how often do you play video or computer games?
- Approximately how many different board games are you familiar with?
- Thinking of your favourite board game, how would you describe your level of play?
- How would you describe yourself as a board game player in general?

The first question was compulsory but all other questions were optional. This personal information was recorded in case further analysis of the results was warranted; it was not used and remains confidential.

Of the 106 individuals who answered these questions and indicated that they were at least 18 years of age, 57 went on to actually participate in the survey.

Game Comparison Survey

The survey itself consisted of a computer programme individually emailed to participants, specifically a cut-down version of the Ludi game player with most features hidden except for those required for playing the survey games. The *.gdl files describing each of the 79 data set of games were embedded in the programme as a resource.

Each time the survey application was launched the following actions took place:

- An introductory screen briefly explained the purpose and operation of the survey.
- Two games (A and B) were loaded at random from the 79 embedded games.
- Users were shown the rules to game A and required to play at least one game.
- Users were shown the rules to game B and required to play at least one game.
- Users were then asked to nominate whether they preferred game A or B.

Participants were encouraged to play each game at least once, and to keep playing each game as many times as they felt was necessary to fully understand its rules.

When it came to choosing between the two games, participants were specifically asked:

> "Which is the better game? Base your choice on which game you found most interesting and would most like to play again. Or if both games are flawed, please indicate the least flawed."

Participants were invited to enter comments about the games that they had just played. These comments made for interesting reading and proved quite useful in some cases.

Each user preference was then emailed back to the author as a message with a subject line of the form:

[Ludi-1] 1 4 -10 2 1 4 [19a0]

Each such subject line consists of the following items:

- A header item "[Ludi-1]" identifying this data as originating from the first survey.
- The index of game A [0..78].
- The index of game B [0..78].
- The user's preference (-10 = game A, +10 = game B).
- The total number of games played for this pair.
- The total time in minutes spent playing this pair of games.
- A checksum value verifying that the line of data is not corrupt.

The preference values of -10 and +10 hark back to an earlier incarnation of the survey in which users could choose from a range of integer choices between -10 and +10 denoting not only which game was preferred, but by how much. However, it was soon found that participants simply specified a preference of 0 in a large number of cases, especially when both games were flawed; it was necessary to force participants into a simple choice in order to coax out the desired information. Moreover, discrete boolean preferences are sufficient for paired-comparison analysis.

The number of games played and time spent on each pair were recorded in case this information proved relevant to user preference, although this information was not used. Players were also invited to make anonymous comments on each pair of games.

Induction of Rankings

A total of 628 data items were submitted by participants, each describing a paired comparison of two games. Analysis was then performed to test whether this information would provide a reliable ranking of games based on human player preferences using a *cross-entropy* (CE) method.

The CE method is a general algorithm for approximately solving global optimisation tasks of the form:

$$w^* = \arg\max_{w} S(w)$$

where S is a general objective function [Szita & Lörincz, 2006]. The CE method proceeds iteratively and instead of maintaining a single candidate solution as per optimisation algorithms, it maintains a distribution of possible solutions and updates this distribution accordingly with each step. De Boer et al provide a more complete discussion of the CE method [2004].

The CE method was applied to the classification task "is game A preferred over game B" over the set of all paired comparisons. Firstly, a value function was induced over the set of ordered pairs then all games were ranked by their corresponding value. Further details of the exact method used are given in Appendix D and a Matlab implementation given in Appendix E.

10.4 Results

Figure 10.1 shows the results of training on the 628 paired comparisons over 100 epochs.

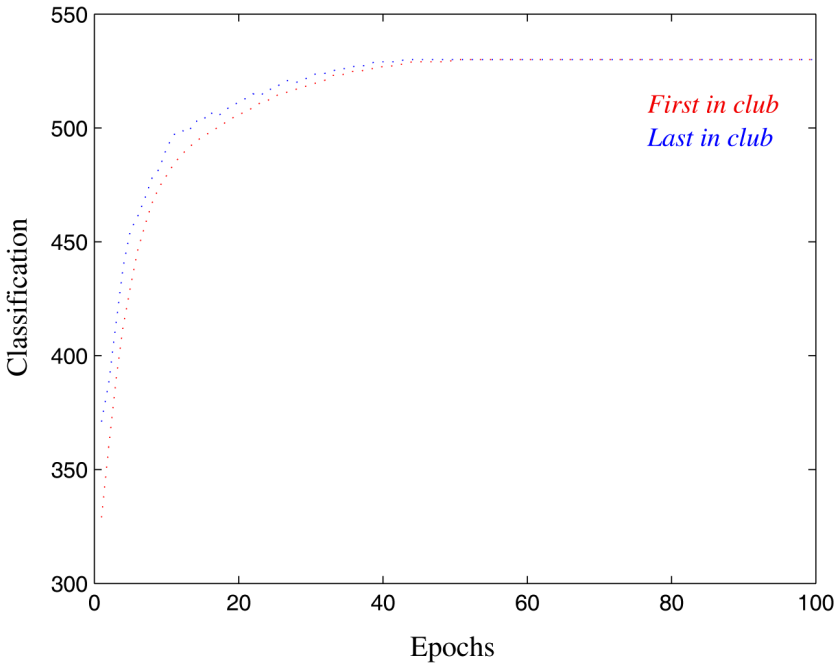


Figure 10.1 Classification success rate over 100 epochs.

For each epoch, an *elite* set of potential solutions are maintained which describe the current best set of E solutions, where the success of each candidate classification scheme is judged by the number of correct classifications.

It can be seen from the figure that the classification converges to a single elite solution at around epoch 50 and does not improve any further. This is the final classification scheme used.

Figure 10.2 shows the success of this classification scheme over all 628 comparisons. The mean classification rate for this training set was 0.899685 (variance = 0.000317653) after 100 epochs. This means that the classification scheme is able to specify whether a given game is preferred over another given or not with almost 90% accuracy.

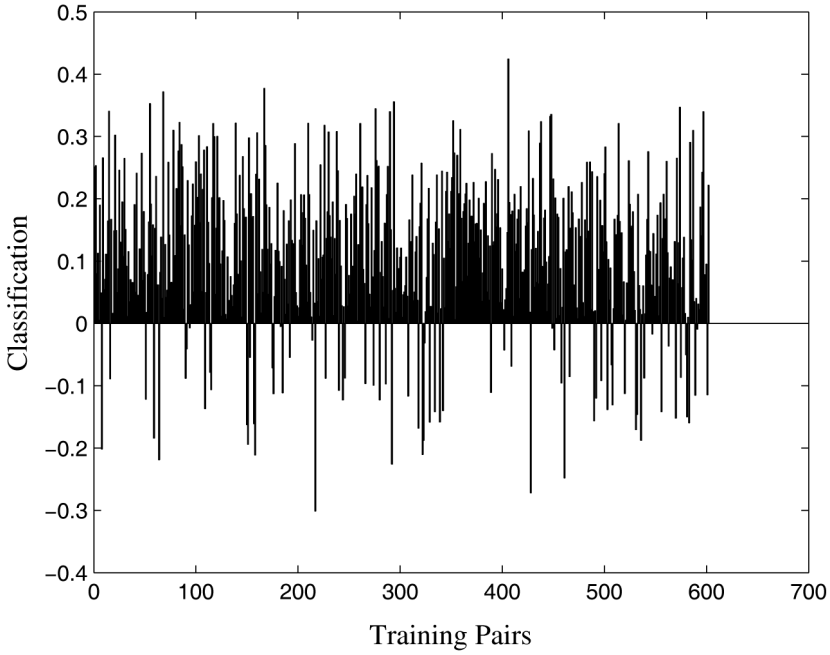


Figure 10.2 Final classification results over all comparisons.

Game Rankings

Each game was assigned a corresponding preference score in the range -0.189892 to 0.233477 and overall game rankings obtained by sorting the games based on this score. The preference scores and ranking values for each of the 79 games are listed in Appendix C. The top ten games are listed below:

#1. 004 - Gomoku (hexhex board, connected move, capture)
#2. 076 - Y (11 per side)
#3. 057 - Capture 5 (jump capture)
#4. 075 - Unlur
#5. 037 - Hex (move, 8 in hand)
#6. 061 - Form a Group (knight moves, capture)
#7. 049 - Stack 5 (line convert)
#8. 046 - Gomoku (move if freedom, surround capture)
#9. 000 - Breakthrough
#10. 035 - Hex (hexhex board)

Complete rule sets for the top ten games are given in Appendix F.

10.5 Discussion

The almost 90% success rate of the classification scheme indicates that the data set of games can be ranked according to human preference with reasonable accuracy. Given two games from this data set,

the one with the better ranking (or preference score) is the game more likely to be preferred by the survey participants. It will be assumed that the preference scores thus obtained indicate the quality of a game, at least within the context of this set of games and in the opinion of this group of participants.

Interestingly, only three of the top ten games – Y, Unlur and Breakthrough – are popularly known games, when it could be expected that known games would feature more prominently as their popularity has been proven over time. However, several variants of known games and even some randomly created games outrank many of the known games, perhaps indicating some curiosity on behalf of the participants to explore new rule combinations. This bodes well for the creation of new games in Experiment III.

10.6 Summary

A survey was conducted to determine human player preferences between pairs of games drawn randomly from a data set of 79 games. The data set of games consisted of mostly viable (playable) designs although a number of pathologically flawed games were added to cover more of the aesthetic design space.

A cross-entropy method was used to rank the games based on human player preferences, phrasing the problem as a task of minimising classification error between rankings. The resulting classification success rate of almost 90% provided a sufficiently reliable ranking scheme, giving reference point for measuring perceived game quality.

Chapter 11

Experiment II: Game Measurement

> Not everything that counts can be counted; not everything that can be counted counts.
> – Albert Einstein

11.1 Introduction

Experiment II is designed to determine whether the aesthetic measurements made by the Criticism module can be correlated with the player preference scores obtained in Experiment I. This experiment involves the aesthetic measurement of the 79 survey games, followed by correlation tests to determine which aesthetic criteria perform as the best predictors of perceived game quality.

11.2 Aim of the Experiment

Given the set of preference scores obtained in the previous experiment, it is now necessary to determine whether these scores may be correlated with aesthetic measurements performed during self-play. If such a correlation can be found, then these aesthetic measurements may be used to predict preference scores for new games indicating the likelihood that human players – in particular, the survey participants of Experiment I – would prefer those games over others.

11.3 Method

Experiment II consists primarily of a number of self-play measurement trials performed on each game, during which a score is calculated for each aesthetic criteria. These scores are then correlated with the human player preference scores obtained Experiment I.

Data Set

The data set consists of the 79 source games used in Experiment I and listed in Appendix C, ranked by human player preference score.

Aesthetic Measurement

Each game then underwent aesthetic measurement consisting of a number of self-play trials during which the 57 aesthetic measurements described in Chapter 10 were performed. Default settings were used and all measurements and the results stored to file for later use.

The measurements were performed on two standard Windows desktop machines over a period of two weeks. Games were batched and measured automatically where possible, however the process had to be supervised to some degree and occasionally interrupted if a game proved troublesome; some games took minutes to measure (e.g. game 065 TicTacToe) while some games took over eight hours (e.g. game 034 Hex-17x17).

In the case of excessively slow games with large branching factors, the measurement process was usually stopped and restarted with a narrower beam search. This generally has less impact on play

than reducing the search depth, and the two aesthetic criteria most affected by this adjustment – move effort and response time – will in any event be unusually high for such games.

Criteria	Min	Max	Mean	C.I. (95%)
0. Complexity	0.615	1	0.7953	-
1. Goal (connect)	0	1	0.2025	-
2. Goal (capture)	0	1	0.1392	-
3. Goal (N-in-a-row)	0	0	0	-
4. Goal (race)	0	1	0.2911	-
5. Goal (group)	0	1	0.0506	-
6. Goal (stack)	0	1	0.0253	-
7. Goal (score)	0	0	0	-
8. Goal (block)	0	1	0.1646	-
9. Piece addition	0	1	0.5696	-
10. Piece removal	0	0	0	-
11. Piece movement	0	1	0.7089	-
12. Piece stacking	0	1	0.7342	-
13. Piece capture	0	1	0.3038	-
14. Piece conversion	0	1	0.0506	-
15. Piece promotion	0	1	0.0253	-
16. Branching factor	0.263	1	0.7358	0.0099
17. Convergence	0	1	0.6098	0.0431
18. Response time	0	0.724	0.1874	0.0210
19. Move effort	0	0.761	0.5309	0.0333
20. Search penetration	0	0.581	0.4358	0.0271
21. Clarity (variance)	0	0.176	0.0131	0.0036
22. Clarity (narrowness)	0.0611	1	0.8610	0.0350
23. Uncertainty (overall)	-0.109	0.99	0.7813	0.1001
24. Uncertainty (early)	-0.14	0.576	0.3918	0.1190
25. Uncertainty (late)	-0.0323	0.567	0.4107	0.0807
26. Drama (average)	0	0.625	0.0954	0.0519
27. Drama (maximum)	0	1	0.2273	-
28. Killer moves	-0.0005	0.422	0.0610	0.0158
29. Permanence	0	1	0.7926	0.0160
30. Lead change	0.0017	0.769	0.3052	0.1177
31. Stability	0	0.996	0.7121	0.0318
32. Decisiveness threshold	0	0.797	0.1970	0.0830
33. Decisiveness	0	0.98	0.4464	0.1810
34. Depth (discrepancy)	0	0.7	0.1580	0.0698
35. Foulup factor	0	0.926	0.1475	0.0430
36. Momentum (1)	0	0.618	0.1454	0.0799
37. Momentum (2)	0	0.591	0.0692	0.0519
38. Momentum (3)	0	0.539	0.0421	0.0371
39. Correction (1)	-0.124	0.821	0.0242	0.0294
40. Correction (2)	-0.15	0.394	0.0213	0.0307
41. Correction (3)	-0.0757	0.108	0.0119	0.0260
42. Control	-0.338	0.142	-0.0229	0.0319
43. Coolness	0	0.229	0.0542	0.0224
44. Board coverage	0	0.921	0.5082	0.0586
45. Puzzle quality	0	0.428	0.0430	0.0631
46. Completion	0	1	0.8260	-
47. Balance	0	1	0.6971	-
48. First move advantage	0	1	0.2822	-
49. Drawishness	0	1	0.0405	-
50. Timeouts	0	1	0.1399	-
51. Duration	0.142	1	0.6121	0.0997
52. Depth (skill level)	0.3	1	0.6722	0.3458
53. Selfishness	0.2	1	0.6310	0.3666
54. Obstructive play	0.3	1	0.6639	0.3565
55. Resilience	0.5	1	0.9310	0.0936
56. Puzzle potential	0	1	0.3807	0.3888

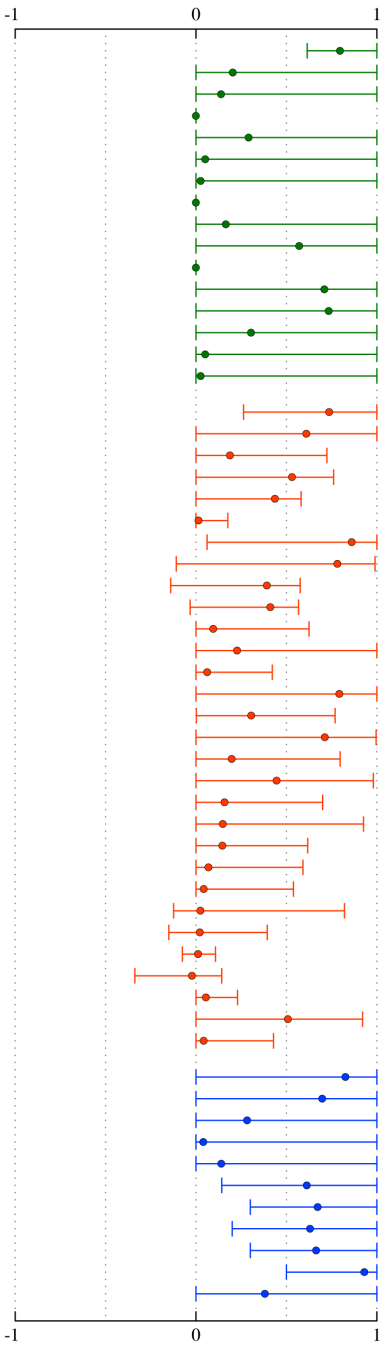


Figure 11.1 Aesthetic criteria scores over all games.

11.4 Results

Figure 11.1 shows the mean and range for each aesthetic criterion over all games, along with the average 95% confidence interval of each measurement. The criteria are grouped into three main categories: Intrinsic, Quality, and Viability.

The average 95% confidence interval is 0 for those criteria consisting of a single value rather than a range of values. This includes all of the intrinsic criteria, one quality criterion and five of the viability criteria.

Preference Correlation

The correlation between these aesthetic scores and the previously obtained human player preference scores was determined using linear regression and a standard *leave-one-out* method of cross-validation. This produces two outputs:

- $corr_s$ the correlation between human player preference scores and the aesthetic criteria values, and
- $corr_r$ the correlation between game rankings by human players and game rankings by aesthetic criteria value.

The ranking correlation $corr_r$ is slightly better than the score correlation $corr_s$ in most cases as rankings are discrete and hence more robust. Correlations will be discussed in terms of $corr_s$ in an effort not to overstate the results.

Table 11.1 lists the correlation results for the aesthetic criteria considered as a whole and by category.

Criteria	$corr_s$	$corr_r$
All (57)	0.4172	0.4256
Intrinsic (16)	0.0941	0.1145
Quality (30)	0.4264	0.4366
Viability (11)	0.5932	0.6086
Best set (17)	0.8208	0.8276
Best set (16)	0.7942	0.7986

Table 11.1 Aesthetic score correlations by criteria type.

It can be seen that the 16 intrinsic criteria are in themselves poor predictors of game preference score ($corr_s$ = 0.0941) while the 11 viability criteria are reasonably good predictors that outperform the overall set ($corr_s$ = 0.5932). The complete results for each category are given in Appendix G.

Best 17 Predictor Set

The best 17 predictor set (second last row of Table 11.1) contains criteria from all three categories. These were determined using a CE method to find combinations of criteria with high correlations, then performing manual improvements as follows:

- Starting with the most highly weighted criterion, add each other highly-weighted criterion in turn.
- If any addition improves performance then add that criterion to the set.
- For each new criterion added to the set, remove each other criterion in the set in turn, checking for improvement.

Figure 11.2 shows plots of predicted versus actual preference scores (left) and predicted versus actual rankings (right) for the 79 survey games using the best 17 predictors. This set shows a strong correlation of $corr_s$ = 0.8208 with a 95% confidence interval of 0.371 [0.562..0.933].

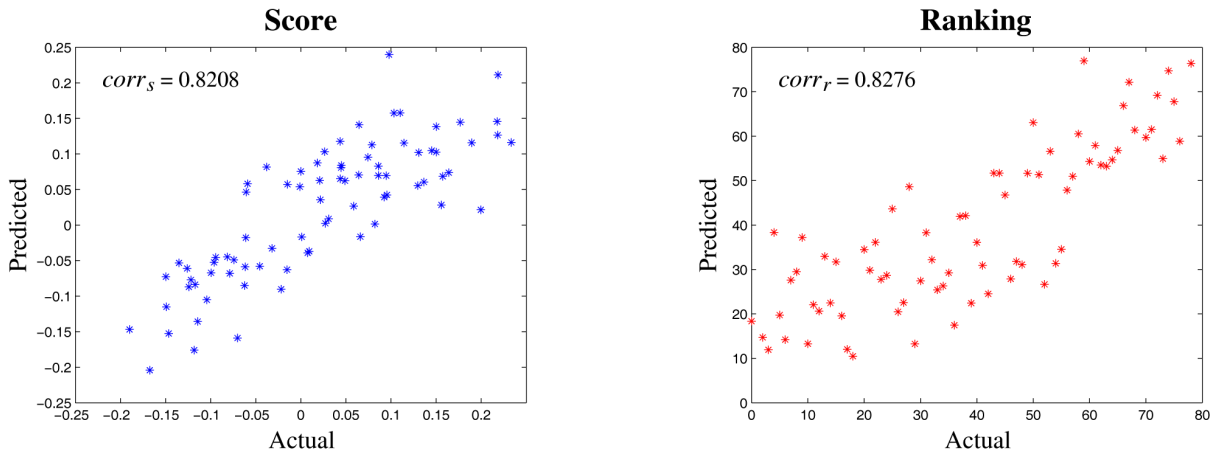


Figure 11.2 Correlation of the best 17 aesthetic criteria with game scores and rankings.

Figure 11.3 lists the actual criteria that make up the best 17 predictor set and their specific correlation weightings. This is not guaranteed to be the absolute best predictor set, but was the best combination found after considerable search.

Criteria	Weight
5. Goal (group)	0.0818
6. Goal (stack)	-0.0804
8. Goal (block)	-0.1055
13. Piece capture	0.0585
17. Convergence	-0.0221
21. Clarity (variance)	-0.3763
25. Uncertainty (late)	0.2023
26. Drama (average)	0.2167
28. Killer moves	0.3585
29. Permanence	0.1027
30. Lead change	-0.2769
32. Decisiveness threshold	0.1311
36. Momentum (1)	0.1244
39. Correction (1)	0.1622
45. Puzzle quality	0.0826
46. Completion	0.0788
51. Duration	-0.0907
Bias	-0.2576

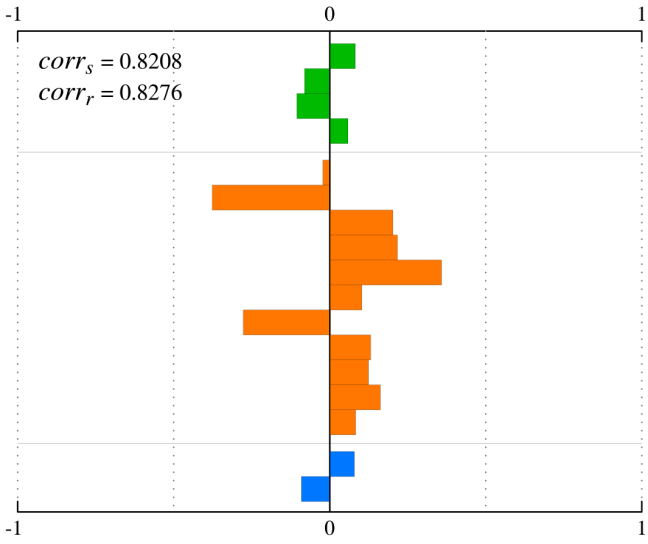


Figure 11.3 The best 17 predictors and their correlation weightings.

Further insight into the relative importance of the criteria within this set can be achieved by removing each in turn and observing the corresponding increase in error in preference score prediction. The greater the error, the more important the criterion, as shown in Figure 11.4.

The bottom row of Figure 11.4 shows a baseline error of 13.28% in preference score prediction over all games using all best 17 predictors (greyed). The remaining rows show the increase in error above this baseline value as each criterion is removed from the predictor set in turn; several criteria stand out as more important than others.

Criteria	Error
5. Goal (group)	14.10%
6. Goal (stack)	13.83%
8. Goal (block)	16.98%
13. Piece capture	15.16%
17. Convergence	14.11%
21. Clarity (variance)	19.16%
25. Uncertainty (late)	48.79%
26. Drama (average)	17.06%
28. Killer moves	23.09%
29. Permanence	28.09%
30. Lead change	36.45%
32. Decisiveness threshold	16.55%
36. Momentum (1)	16.42%
39. Correction (1)	15.10%
45. Puzzle quality	13.68%
46. Completion	23.10%
51. Duration	20.88%
Baseline (all 17 criteria)	13.28%

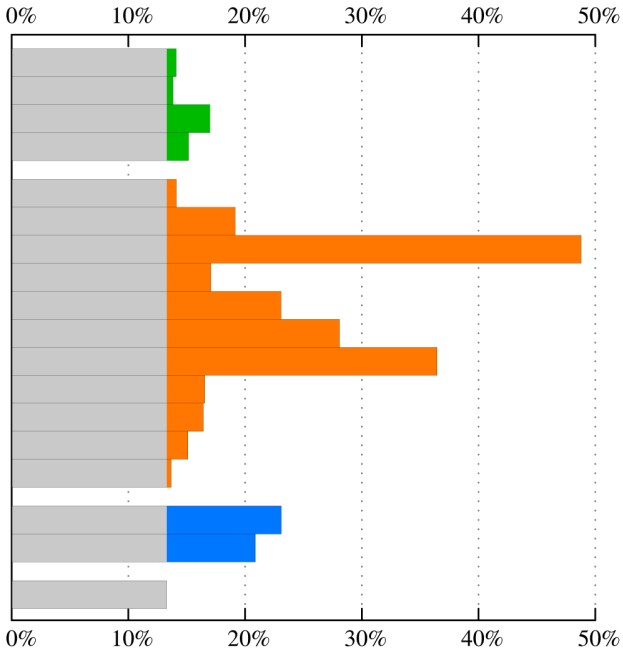


Figure 11.4 Increase in error following the removal of each criterion.

Validation of Results

To test the accuracy of these results, the difference between the actual preference score for each game and the preference score predicted by the best 17 criteria was calculated. The average error in score prediction was 14.1% of the total score range. The minimum error was 0.6% and the maximum error was 31.2% in one instance.

When ranking games by predicted score using the best 17 criteria, the average ranking error was 8.9 positions per game (11.2%) with 0 error for several games and a maximum error of 30 positions for one game.

Best 16 Predictor Set

The best 16 predictor set (last row of Table 11.1) consists of the best 17 predictor set with the "Puzzle quality" criterion removed. This criterion was the only slow measurement of the set and its removal dramatically improves the speed of game measurements with a negligible drop in accuracy of 0.029% (it is in fact the least significant of the 17 predictors).

The best 16 predictor set is therefore used for practical game measurements in which speed is critical, for instance during the evolution of new games.

11.5 Discussion

These results support the first hypothesis of this thesis: *that there exist fundamental (and measurable) indicators of quality for combinatorial game design*. It has been shown that a small number of aesthetic measurements may be used to predict human player preference scores for combinatorial games with a considerable degree of accuracy.

The poor performance of the 16 intrinsic criteria as predictors is not surprising given their simplistic nature; they do not convey much useful information and probably act more as player preference indicators rather than universal quality indicators. The superior performance of the 11 viability criteria

is also no surprise as these criteria are generally well-defined and robust, being based primarily on game outcomes.

The makeup of the best predictor sets is interesting. Six criteria stand out as most important overall:

- Uncertainty (late),
- Killer moves,
- Permanence,
- Lead change (negative),
- Completion, and
- Duration (negative).

This combination suggests that the survey participants generally prefer games with uncertain outcomes that end within a reasonable number of moves, and in which strong moves are reasonably permanent. Uncertainty (late) stands out as the single most important criterion for predicting perceived game quality.

It would be dubious to assume that these results reveal universal truths about combinatorial game design in general. The survey participants who made the initial preferences constitute a small part of the population and the games involved represent a small subset of all combinatorial games. In addition, the relative contribution of each criterion to prediction accuracy may vary depending on the other criteria with which it is combined.

The absence of any criteria from the best predictor sets should not be taken to imply that the associated concepts are not relevant to game design quality, but rather that different methods may be required to measure them, if they are in fact measurable at all.

11.6 Summary

Each of the 57 aesthetic criteria were measured for each of the 79 source games during a number of self-play trials, and correlations sought between these aesthetic measurements and the user preference scores obtained in the previous experiment.

Some correlation was found over all 57 aesthetic measurements. No significant correlation was found over the 16 intrinsic criteria, some correlation was found over the 30 quality criteria, and a better correlation was found over the 11 viability criteria.

Using a combination of the cross-entropy method and systematic elimination, a much stronger correlation was found to exist for a particular combination of 17 aesthetic criteria from all three categories. This set was streamlined to a faster best 16 predictor set for practical use.

An analysis of the most significant criteria in the best predictor sets suggests that the survey participants generally prefer games with uncertain outcomes that end within a reasonable number of moves, and in which strong moves are reasonably permanent.

These results support the first hypothesis of this thesis: *that there exist fundamental (and measurable) indicators of quality for combinatorial game design*.

Chapter 12

Experiment III: Game Synthesis

> There's nothing like millions of years of really frustrating trial and error to give a species moral fibre and, in some cases, backbone.
> – Terry Pratchett
>
> Creative destruction.
> – J. A. Schumpeter on innovation

12.1 Introduction

Experiment III investigates whether the knowledge obtained from the previous two experiments may be used to inform the evolutionary search for new high quality games. Specifically, Experiment III involves the creation of new games through the evolution of existing rule sets and determining whether those evolved games deemed to be viable may then be reliably measured for quality. The resulting evolved games are presented and the more interesting emergent rule combinations highlighted.

12.2 Aim of the Experiment

The first aim of the experiment is determine whether new, viable games may be evolved from an initial population of known rule sets. The second aim of the experiment is to determine whether these newly evolved games may be reliably measured for quality based on the game preference correlations obtained in Experiment II.

The ultimate aim of this experiment is to produce new games of high quality.

12.3 Method

New games were created by the Ludi system's Synthesis module, as outlined in Chapter 9. Those new games deemed to be viable were then set aside for further refinement and analysis, and their predicted quality scores validated in a short follow-up survey.

12.3.1 Game Synthesis

Data Set

The initial population consisted of the 79 source games listed in Appendix C.

Game Synthesis

A number of evolutionary runs were then conducted on three standard Windows desktop machines on two continents over a period of a week. Each run consisted of 100 generations using the default user settings, producing up to 100 new games per run.

The fitness function used for the evolutionary process was the game preference score predicted by the best 16 predictor set determined in Experiment II. Recall that all child games are retained for the reasons given in Sections 3.3.3 and 9.7, and that the fitness function is used at this stage to rank new games within the population rather than for culling unfit children.

Following each run, new child games and their corresponding aesthetic criteria measurement files were moved to a central location and merged with the existing games (some name clashes had to be manually rectified when merging populations). This created an ever-growing population of increasing diversity which was used as the initial population of each successive evolutionary run.

The aggressive size and speed tests made during the evolutionary process ensured that a large number of games were created in a reasonable amount of time. There is no reason that the process could not be run successfully for many thousands of generations; it was run in batches of 100 mostly for the sake of regularly collating the populations from different machines and generally monitoring overall progress.

After the final population had been generated, all new games deemed to be viable were then put aside for further consideration. The policies of the viable games were optimised (some manually in the interests of speed) and the games remeasured to produce a more accurate final predicted aesthetic score.

Viability Test

Table 12.1 shows the effect of testing for viability at different thresholds (the completion and balance thresholds increase from 0.0 while the advantage and duration thresholds decrease from 1.0). The values shown are the numbers of evolved games that pass the test at the given thresholds.

	>0.0 <1.0	**>0.1 <0.9**	**>0.2 <0.8**	**>0.3 <0.7**	**>0.4 <0.6**	**>0.5 <0.5**	**>0.6 <0.4**	**>0.7 <0.3**	**>0.8 <0.2**	**>0.9 <0.1**
Completion	1389	1389	1389	1388	1386	1378	1374	1353	1340	1309
Balance	999	999	994	986	977	935	932	700	685	196
Advantage	800	800	789	788	750	700	609	483	292	104
Duration	1388	436	225	162	119	67	32	24	21	1
C+B+A	416	461	400	390	349	283	228	122	53	3
C+B+A+D	416	288	147	96	65	**19**	6	1	0	0

Table 12.1 Viability testing at different thresholds.

It can be seen that completion on its own is surprisingly poor for distinguishing viable games, presumably due to the large number of pathological games with trivial wins on the first or second move. Duration is by far the most effective single viability measure, and the combination of completion, balance, advantage and duration is the most effective test.

The importance of duration is not surprising, as a well-behaved game that provides a meaningful contest between players will last for at least a reasonable number of moves and conclude within a reasonable number of moves. A good duration score implies a high completion rate, unless there are an inordinate number of draws.

It is worth reiterating that the four viability criteria are not based on human player preferences. They are universal measurements based on game outcomes with general applicability to all games.

12.3.2 Follow-Up Survey

The final predicted game preference scores were then validated by a small follow-up survey. The follow-up survey followed the same format as the initial survey conducted in Experiment I, except that users were asked to choose between game pairs drawn from the 19 viable evolved games rather than the 79 original survey games. This survey was covered by the same QUT ethical clearance as the initial survey.

A total of 27 individuals participated in the follow-up survey, recruited mostly from the original 57 survey participants. The small number who were not participants of the original survey were required to confirm that they were at least 18 years of age; the newcomers were notionally regarded as an extension of the original population as they were recruited through the same online board game community [Rognlie, 1996] as the vast majority of the original survey participants.

Each time a participant ran the survey, a paired comparison was made and emailed back to the author as a message with a subject line of the form:

[Ludi-2] 11 5 10 2 3 4 [a32f]

The signifier "[Ludi-2]" identifies this paired comparison as belonging to the follow-up survey. All other values are as for the initial survey except that the game indices are in the range [0..18] for the 19 evolved games.

A total of 127 paired comparisons were submitted by participants. Analysis was then performed to test whether this information would provide a reliable ranking of the 19 evolved games based on human player preferences.

12.4 Results

Starting with an initial population of 79 source games, a total of 1,389 new games were evolved of which 19 were deemed to be viable, an attrition rate of 98.7%.

The 19 viable evolved games were detected using 0.5/0.5 thresholds for the viability test. The final predicted aesthetic scores for these games ranged from -0.233 to 0.104.

12.4.1 Induction of Rankings

Actual game preference scores and rankings based on the results of the follow-up survey were determined using the CE method for paired comparison outlined in Experiment I.

Figure 12.1 shows the results of training on the 127 paired comparisons over 100 epochs. Note that convergence occurs much more quickly – within 15 epochs – for this smaller data set.

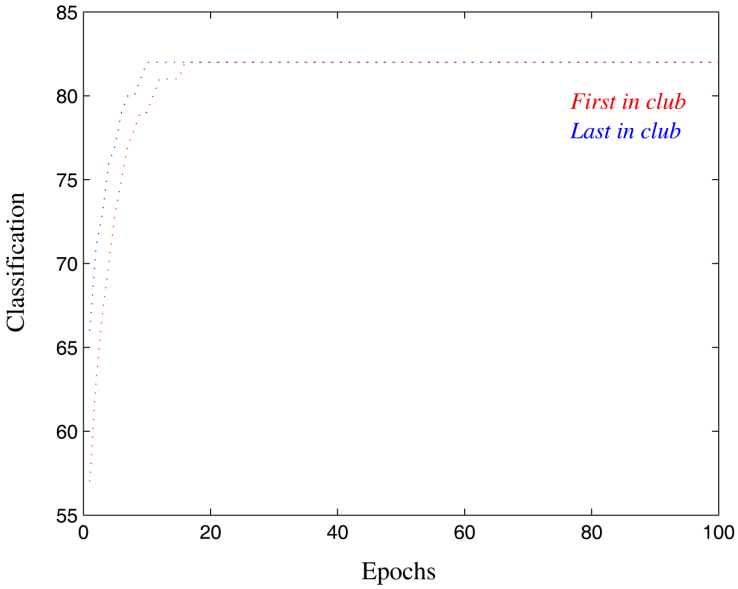


Figure 12.1 Classification success rate over 100 epochs.

Figure 12.2 shows the classification results after 100 epochs over all comparisons. The mean classification rate for this set of data was 0.828283 after 100 epochs.

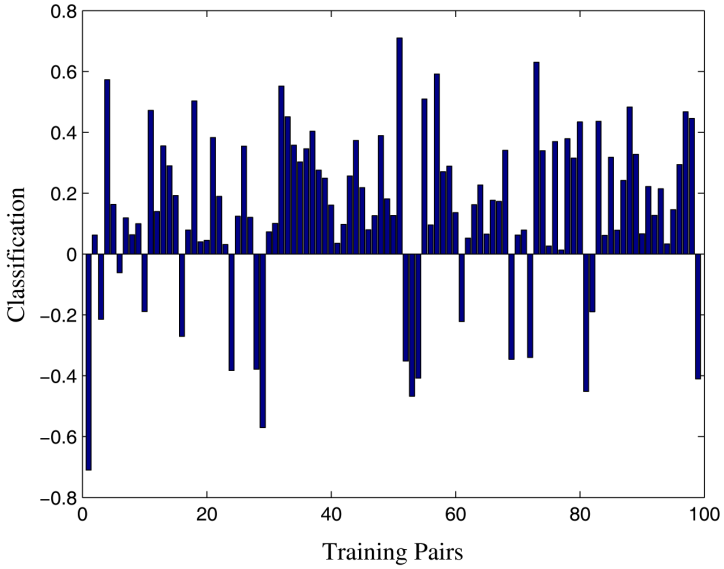


Figure 12.2 Final classification results over all comparisons.

Games were then ranked from most to least preferred based on human player preference:

Rank	Name	Preference	Predicted Aesthetic Score
#1.	Ndengrod	0.397170	0.104189
#2.	Yavalath	0.378889	0.057961
#3.	Rhunthil	0.318287	0.081392
#4.	Teiglith	0.278130	-0.007691
#5.	Elrostir	0.251483	-0.043764
#6.	Lammothm	0.215914	0.031938
#7.	Gorodrui	0.184889	-0.018419

#8.	Pelot	0.151842	0.096196
#9.	Hale	0.089290	-0.066060
#10.	Quelon	0.076292	-0.030909
#11.	Duath	0.024024	-0.044716
#12.	Elrog	-0.021202	-0.110200
#13.	Vairilth	-0.037036	-0.021704
#14.	Ninniach	-0.100200	-0.092579
#15.	Pelagonn	-0.172998	-0.058678
#16.	Valion	-0.194103	-0.0646671
#17.	Eriannon	-0.251480	-0.233478
#18.	Bregorme	-0.293602	-0.157509
#19.	Pelagund	-0.331139	0.037925

12.4.2 Preference Correlation

Figure 12.3 shows a plot of the actual game rankings versus the predicted aesthetic scores, giving a correlation of -0.6491 with a 95% confidence interval of 0.577 [-0.851..-0.274]. The correlation is negative as games with lower (i.e. better) rankings generally have higher predicted aesthetic scores, as expected.

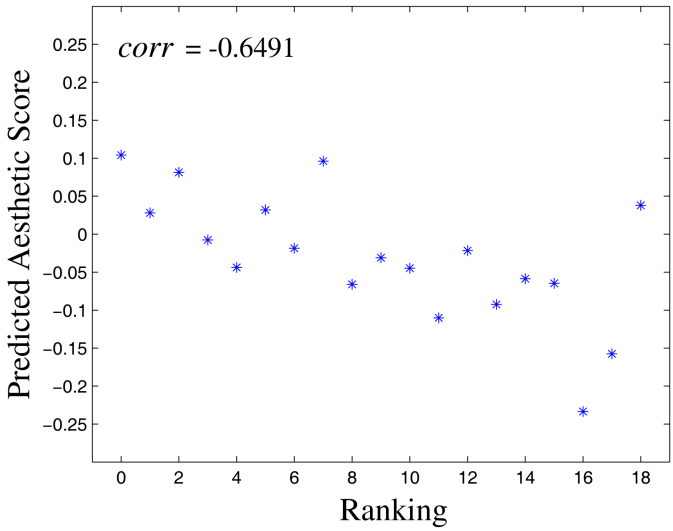


Figure 12.3 Final classification results over all comparisons.

Note that the aesthetic score function was based on data obtained from the original set of 79 source games and not on any of the evolved games. The 19 viable evolved games therefore act as a test set for this function, and it is encouraging that the aesthetic score predictions based on the earlier set of games are reliable indicators of quality for new games.

12.4.3 Viable Evolved Games

The 19 viable evolved games are the culmination of this thesis. Each game is now considered briefly, and their complete rule sets given in Appendix H.

#1. Ndengrod

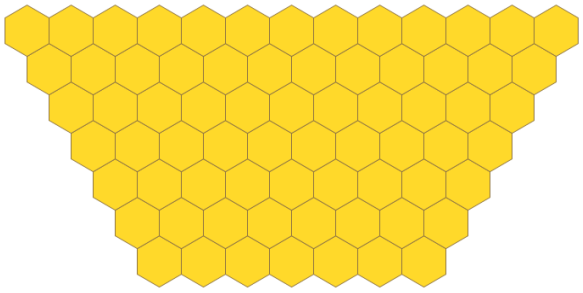


Figure 12.4 Starting position for Ndengrod.

Move: Add to empty cell, surround capture.
Aim: 5-in-a-row.

Ndengrod, the game most preferred by human players, was also the game with the highest predicted score. It plays well but involves little innovation apart from the surround capture rule and the unusual board shape. This rule combination does not exist for any known game, as far as can be determined, but is similar to one of the initial data set of 79 games.

#2. Yavalath

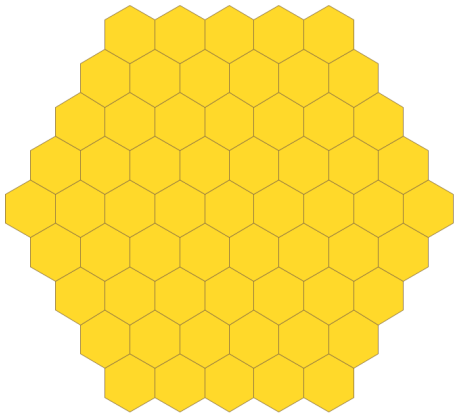


Figure 12.5 Starting position for Yavalath.

Move: Add to empty cell.
Aim: 4-in-a-row, but lose if 3-in-a-row and not 4-in-a-row.

Yavalath's end condition is probably the most interesting play mechanic to emerge from the evolutionary process:

```
(end
    (All win (in-a-row 4) )
    (All lose
        (and
            (in-a-row 3)
            (not (in-a-row 4) )
        )
    )
)
```

This set of winning conditions does not exist for any known game, as far as can be determined.

The development of this rule can be followed through the game's family tree, shown in Figure 12.6. This tree is obtained by recursively following the game's parents, and each of their parents in turn.

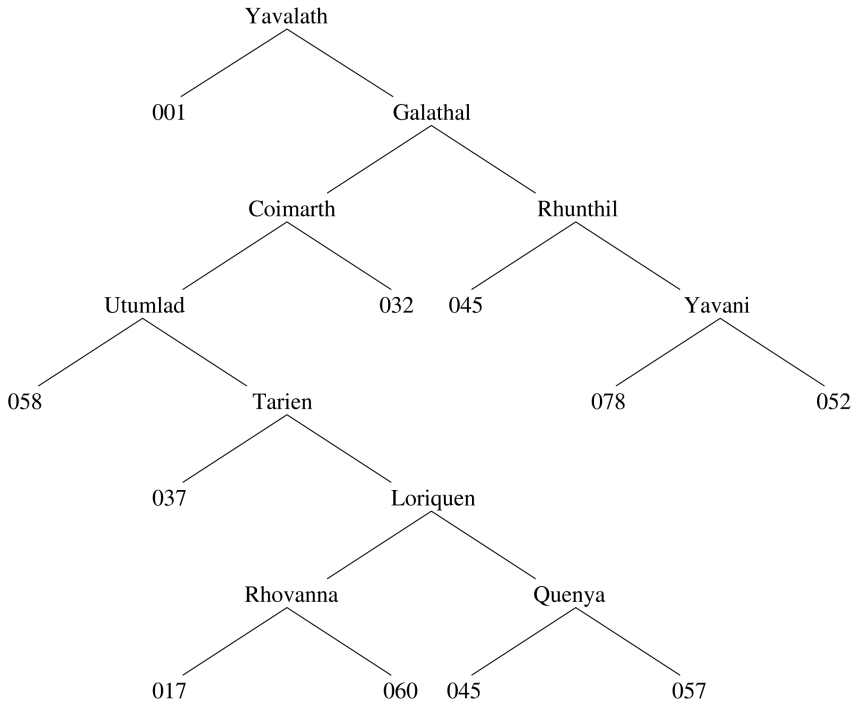


Figure 12.6 Yavalath's family tree.

An ancestor of the emergent rule occurred in Loriquen in the following form:

```
(All win
    (and
        (in-a-row 5)
        (not (in-a-row 4) )
     )
)
```

The (*in-a-row* 5) clause was presumably crossed over from Quenya. A mutation would have added the conjunction (*and* ...) for which the (*in-a-row* 4) clause was created as a second argument, then negated by another mutation.

A further mutation two generations later in Utumlad most likely changed the *win* attribute to *lose*:

```
(All lose
    (and
        (in-a-row 5)
        (not (in-a-row 4) )
    )
)
```

Note that this particular end condition can never be satisfied as 5-in-a-row cannot be formed without also forming 4-in-a-row. This rule is therefore superfluous and any game carrying it must rely on alternative end conditions to reach a conclusion; this ludeme constitutes a passive gene. It was not until the parameter 5 mutated to a 3 in the final rule set that this passive rule became active.

This example emphasises the importance of not culling superfluous and impossible rules. If this impossible rule were culled, then Yavalath, probably the most interesting of all of the bred games, would not have evolved.

Yavalath's genealogy includes two instances of the same ancestor (045) at various stages, as well as the third most preferred game Rhunthil. This is more than coincidence, as Rhunthil scored well even in the initial (approximate) aesthetic measurement, hence was placed near the top of the population and more likely to be selected for mating by stochastic universal sampling.

As an aside, one participant of the follow-up survey admitted to playing Yavalath repeatedly for over an hour, initially to beat the computer opponent but eventually to simply explore the game. Yavalath has good potential to interest human players.

Although the *N*-in-a-row games were generally played well by the computer due to the relative strength of the *N*-in-a-row advisor, the Ludi general game player was hampered in the case of Yavalath by the *rule tension* inherent in the fact that 3-in-a-row is infinitely bad while 4-in-a-row is infinitely good, introducing a nonlinearity that cannot be modelled by the strictly linear reward function. The advisor would encourage line growth where possible but suddenly find a growth to length 3 constituting a losing move contradiction, hence move ordering during adversarial search could become unreliable. However, the Ludi general game player was robust enough to play Yavalath competently despite this problem.

Rule tension is a technique used by game designers to add interest to a game. A game will generally be more challenging if each move must be carefully weighed up in terms of competing goals (in this case, the urge to complete a line of 4 and the urge to not complete a line of 3) rather than a single overriding goal; rule tension is good for humans, if not for machines. It is encouraging to see general principles of this sort emerge from the automated process.

Appendix I presents further analysis of Yavalath including an annotated sample game.

#3. Rhunthil

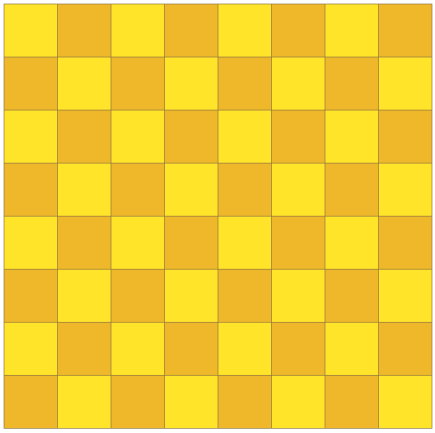


Figure 12.7 Starting position for Rhunthil.

Move: Add to empty cell, plus several complex movement choices.
Aim: 5-in-a-row.

Rhunthil's high ranking was unexpected; its rules are verbose, barely comprehensible and include a dangerously malformed ludeme. This is borne out in the anonymous comments provided by survey participants who invariably describe the rules as "confusing".

It is worth noting, however, one participant's intuition that Rhunthil might be quite interesting to play if only they could wrap their mind around the rules. This intuition was supported by the automatic measurements made during self-play trials.

#4. Teiglith

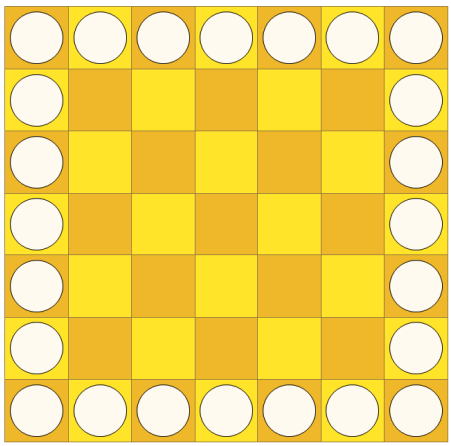


Figure 12.8 Starting position for Teiglith.

Move: Connected movement with stacking (phase is irrelevant).
Aim: Win if no move.

Teiglith is a game in which the White pieces are shared by both players. It is serendipitous that the game starts with White pieces already on the board, otherwise Black would have no moves and win on their first turn.

Teiglith exploits a feature of the Ludi general game player by specifying that all pieces start along White's home row (*start* (*place* (Stone White) *home*)). Since no direction is specified for White, then a default direction of *all* is used and pieces are placed along all sides.

Teiglith is the only game to satisfy viability testing at thresholds of 0.7/0.3 as shown in Table 12.1. This is the most viable evolved game in terms of the preliminary fitness tests performed during evolution.

The combination of connected movement with the *no-move* winning condition is another example of good emergence; these rules combine to result in games that converge elegantly to a solution. Further, this combination of rules instils a group connection aspect to the game which applies to the many isolated groups that develop rather than a single global group. This concept does not exist for any known game, as far as can be determined.

#5. Elrostir

Move: Add to empty cell.
Aim: Lose if no move or 3-in-a-row (includes diagonals).

Elrostir also features the intriguing "lose if 3-in-a-row" rule seen in Yavalath. This rule is surprisingly easy to fall victim to on the 8-connected square grid – much more so than on Yavalath's 6-connected hexagonal grid – and some concentration is required when playing this game.

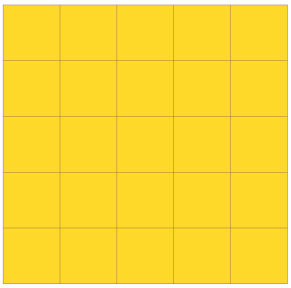


Figure 12.9 Starting position for Elrostir.

Elrostir is rather puzzle-like in nature but also quite cold; there are not many move choices each turn, and it is easy to make a disastrous mistake. This rule combination does not exist for any known game, as far as can be determined.

The "lose if no move" rule is detrimental to the game as the first player will win if the board has an odd number of cells and both players manage to avoid forming 3-in-a-row. Indeed, the first move advantage is measured at 0.35 for Elrostir, which is dangerously high.

Elrostir is the most evolved game of this group, being the result of fifteen generations of breeding.

#6. Lammothm

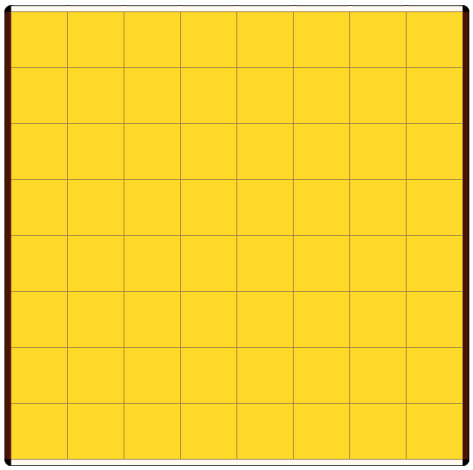


Figure 12.10 Starting position for Lammothm.

Move: Add to empty cell, surround capture.
Aim: Connect own regions (includes diagonals).

Lammothm is a first generation game and hence one of the least evolved viable games.

#7. Gorodrui

Move: Add to empty cell, move to empty cell if distance = state + 1, increment state.
Aim: Lose if no move.

Gorodrui is an interesting puzzle-like game in which each move increments the piece state and limits that piece's future movement choices. The fact that piece movement becomes more constrained with each move means that games quickly converge to a solution. This rule combination does not exist for any known game, as far as can be determined.

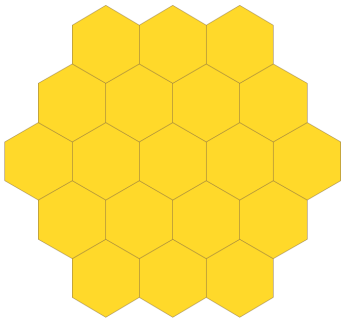


Figure 12.11 Starting position for Gorodrui.

The evolved rule set contains a *move*/*state* confusion similar to that found in Rhunthil (Appendix H).

Gorodrui is a first generation game and hence one of the least evolved viable games.

#8. Pelot

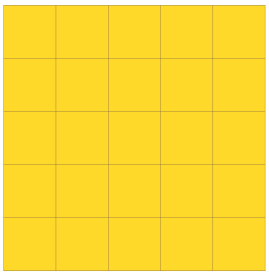


Figure 12.12 Starting position for Pelot.

Move: Add to empty cell, move to connected cell whose distance ≠ state, stacking allowed (increment piece state after moving).
Aim: 4-in-a-row (no diagonals).

#9. Hale

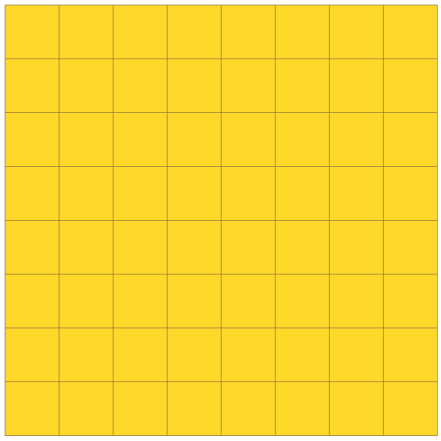


Figure 12.13 Starting position for Hale.

Move: Add to empty cell.
Aim: 4-in-a-row (no diagonals).

#10. Quelon

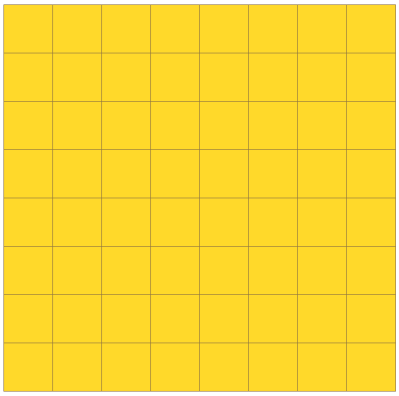


Figure 12.14 Starting position for Quelon.

Move: Add to empty cell.
Aim: Lose if 4-in-a-row (with diagonals).

#11. Duath

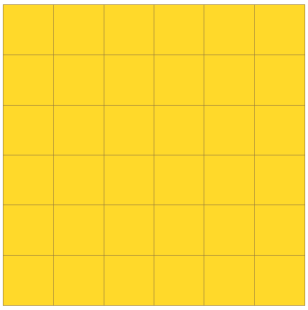


Figure 12.15 Starting position for Duath.

Move: Add to empty cell.
Aim: Lose if 4-in-a-row (with diagonals).

#12. Elrog

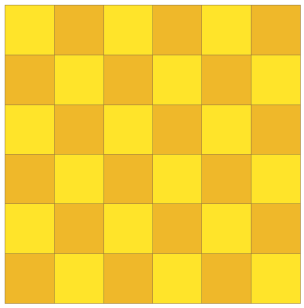


Figure 12.16 Starting position for Elrog.

Move: Add to empty cell.
Aim: 4-in-a-row (with diagonals).

The rules contain a vestigial *phase* element, otherwise this game would probably have been culled as an inbred duplicate. More aggressive culling of vestigial rules may be warranted at the conclusion of the evolutionary process.

#13. Vairilth

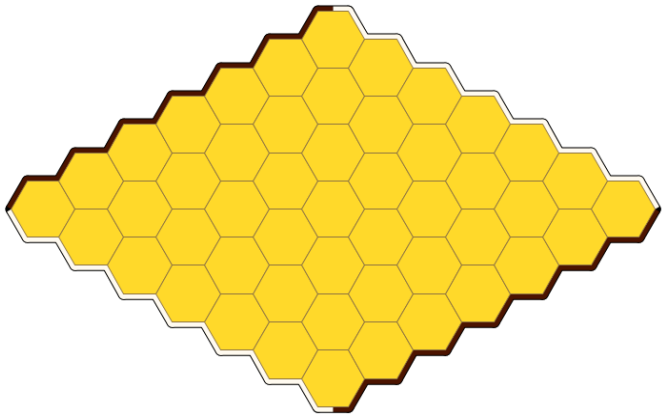


Figure 12.17 Starting position for Vairilth.

Move: Add to empty cell.
Aim: Connect own regions and no move.

Vairilth is identical to Hex except that games continue until all board cells are occupied, which is redundant in terms of game outcome and hence a flaw. However, this very flaw could be the reason that this game survived; without it, many games would have been much shorter (Hex games typically only use a small part of the board) perhaps causing this game to fail the duration part of the viability test.

This suggests that it may be beneficial to base the preferred game duration on board size and possibly rule complexity rather than an absolute number of moves.

#14. Ninniach

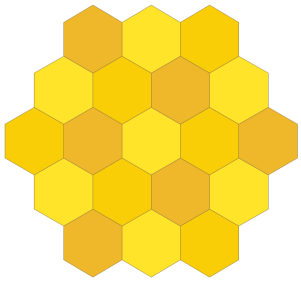


Figure 12.18 Starting position for Ninniach.

Move: Add to an empty cell of your phase, move to a connected cell at least as high to capture.
Aim: Lose if 4-in-a-row (with diagonals).

The rule specifying that pieces can only move to connected cells at least as high means that pieces cannot move to empty cells. This is an alternative way to phrase (*not* (*empty to*)), an emergent but inefficient rule combination.

#15. Pelagonn

Move: Add to empty cell.
Aim: 5-in-a-row.

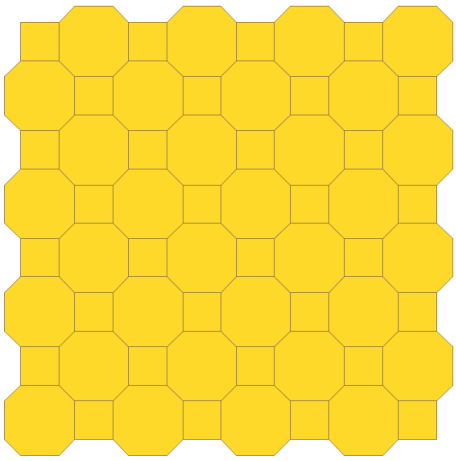


Figure 12.19 Starting position for Pelagonn.

#16. Valion

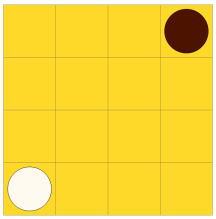


Figure 12.20 Starting position for Valion.

Move: Add to empty cell with at least one enemy neighbour.
Aim: 3-in-a-row but lose if 4-in-a-row or group (with diagonals).

There is a serendipitous combination of starting placements without which players would not be able to make any moves.

Valion also includes an impossible movement rule that has no bearing on the game: add to connected empty cell.

#17. Eriannon

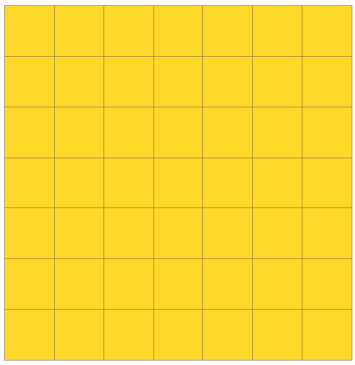


Figure 12.21 Starting position for Eriannon.

Move: Add to empty cell.
Aim: 4-in-a-row (no diagonals).

#18. Bregorme

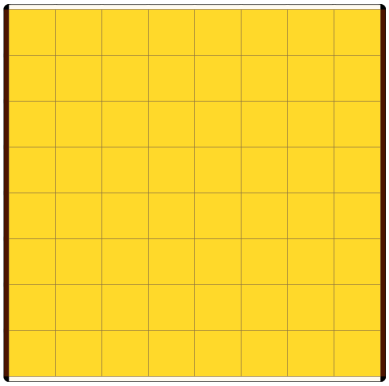


Figure 12.22 Starting position for Bregorme.

Move: Add to empty cell.
Aim: Connect own regions (with diagonals).

#19. Pelagund

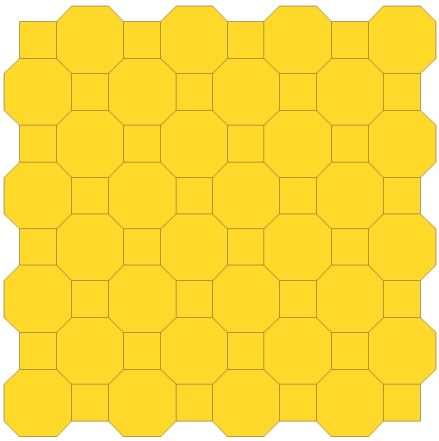


Figure 12.23 Starting position for Pelagund.

Move: Add to empty cell or move to connected cell, with stacking.
Aim: 6-in-a-row or no move.

12.5 Discussion

The correlation of -0.6491 between predicted aesthetic scores and human player rankings is less significant that the correlation obtained in the previous experiment, but that is to be expected as the aesthetic score predictors were trained on the previous set of 79 games and not the evolved games. In fact, this correlation is very good for aesthetic score predictors trained on a different data set, and suggest that they embody useful knowledge.

These results support the second hypothesis: *that these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games*. The search process was not strictly fully automated in this case (populations were manually merged between evolutionary runs and the policies of some of the viable games manually optimised) however these manual intrusions into the automated process could be easily avoided in future experiments.

Most importantly, the evolutionary process produced a number of viable games and recognised those more likely to be preferred by human players. Interesting and novel rule combinations emerged in several of the evolved games.

The prevalence of *N*-in-a-row games among the final viable set is worthy of note: 12 of the 19 evolved games feature an *N*-in-a-row end condition of some nature. In fact, several are effectively equivalent to existing Gomoku games and might have been culled as inbred except for the presence of superfluous rules. These games were not necessarily ranked highly by human players but obviously have a strong survival factor; *N*-in-a-row appears to be a robust rule that thrives in combination with a wide variety of other rules. The fact that the *N*-in-a-row advisor is the strongest of the advisors may also contribute to its prevalence.

As an aside, this robustness of the *N*-in-a-row winning condition may point to why Tic Tac Toe is one of the world's most well-known games (if not the most popular – it is in fact the least popular of the thousands of games ranked on the BoardGameGeek online database of games [Alden, 2000]). The 3x3 square grid is useful for introducing children to games and the choice of *N*-in-a-row winning condition is probably no coincidence as this rule is robust, self-contained, easy to explain and understand, and offers at least some variety of play on very small boards.

Similarly, there are many more games with the *no-move* end condition than would be suggested by its mutation frequency. This may be due to the fact that this rule provides a winner for some games that would otherwise end in a draw, or may simply indicate a high survival rate in a variety of situations. Perhaps the success of these two hardy rules is due to the fact that they are self-contained and do not rely on the presence of other rules to be achieved, whereas reach-a-goal games and connection games require the definition of regions, capture games require the presence of particular enemy pieces, and so on.

The relative absence of other types of end conditions in the evolved games should not be construed as a reflection on their general utility, as the bias towards *N*-in-a-row games might also indicate a bias in the original set of 79 survey games. A large number of these were well-formed *N*-in-a-row games, whereas there were relatively fewer connection and capture games and these generally occurred on larger boards. Future evolutionary runs would probably benefit from a broader and more varied initial population, perhaps including the extrapolation of each parent game over several board sizes.

12.6 Summary

An evolutionary search for new high quality games was performed, using the best 16 predictor set determined in Experiment II as the aesthetic fitness function. Preliminary measurements made during the evolutionary process proved unreliable in terms of game quality but proved useful for determining game viability based on:

- Completion,
- Balance,
- Advantage, and
- Duration.

Duration stood out as the most useful indicator of viability. Of the 1,389 games evolved during the search, 19 proved viable. The viability test is universal and reliable, being based on game outcomes during self-play rather than user preferences. The viable evolved games were then manually optimised and remeasured for aesthetic score.

A follow-up survey was conducted to determine human player preferences for the 19 viable games and a strong correlation found between these preferences and the predicted aesthetic scores. This

result was especially meaningful as the aesthetic scores were based on results obtained from the original 79 survey games only; the measurement proved useful across both existing and new games.

The evolved games reveal a strong prevalence towards the *N*-in-a-row winning condition and a notable occurrence of the *no-move* winning condition. It is deduced that these are strong rules robust to a variety of rule combinations.

Analysis of the 19 viable games reveals novel and interesting emergent rule combinations. Yavalath, in particular, contains a combination of opposed winning conditions carried for many generations as an impossible rule that eventually mutated into a superior combination. This demonstrates the benefit of allowing the propagation of passive rules that may not directly benefit their carriers but which may yield rewards in future generations. This example also demonstrates the unexpected simulation of rule tension as might be used by traditional game designers.

These results support the second hypothesis: *that these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games*.

Chapter 13

Conclusion

13.1 Introduction

This chapter summarises the conclusions of the study, highlighting improvements over previous work and new research contributions. Limitations of the study are considered and directions for future work suggested.

13.2 Conclusions of the Study

The results support both hypotheses.

I: *That there exist fundamental (and measurable) indicators of quality for combinatorial game design*.

It was demonstrated that aesthetic measurements of game quality made automatically during self-play trials could be correlated with human player preference scores for those games. Furthermore, this correlation also proved valid for new games evolved by the system that were not part of the initial training set, pointing to the general applicability of the aesthetic measurements for this group of subjects.

The results suggest that the subjects generally prefer games with uncertain outcomes that end within a reasonable number of moves, and in which strong moves are reasonably permanent. Outcome uncertainty appears to be the most important indicator of game quality.

II: *That these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games*.

Preliminary game quality measurements were used to position newly evolved games within the population during the evolutionary process. Although these measurements did not immediately provide reliable quality scores, they were useful for identifying viable games that warranted further study. Game duration stood out as the most reliable indicator of viability. More precise quality measurements of the viable evolved games then correlated strongly with human player rankings for those games.

In addition to demonstrating the general validity of the hypotheses, another important result of the study was the emergence of interesting new rule combinations in the evolved games. The game evolution process is therefore a potentially useful tool for game designers, who may use it to generate new rule combinations or even fully formed games to explore.

13.3 Improvements Over Previous Work

Previous research in combinatorial game analysis has focussed on solving games and finding optimal strategies; little work has been done on quantifying game quality. This study provides a general aesthetic model for measuring quality in combinatorial games, and implements a greater breadth of aesthetic criteria than any previous work.

At least one previous project has tackled the problem of automated game design [Pell, 1993b]. However, this process was constructive in nature and used generative rules to produce well-defined (if randomised) output. No previous study has undertaken an evolutionary search for new combinatorial game designs, allowing the emergence of truly surprising rule combinations.

The Ludi GDL offers improvements over previous GDLs, notably its simplicity and native support for higher-level game properties such as connectivity.

13.4 Research Contributions

This project provides two major research contributions:

- A complete aesthetic model for the measurement of game quality, and
- A complete process for the automated creation of new high quality games.

The proposed aesthetic model distinguishes between viability and quality measurements, and the results indicate that viability is a precondition for measuring quality. The viability measurements are robust and universal rather than specific to any given set of human player preferences. This aesthetic model is based on a player-centric model of game play, using move and lead histories, which is significantly different to approaches previously followed in combinatorial game analysis.

More importantly, this thesis practically demonstrates the automatic creation of new combinatorial games that human players find interesting. Modifications to standard genetic programming techniques for evolving rule sets are described. The importance of relaxing rule optimisation during the evolutionary process is demonstrated, in order to encourage the emergence of new and interesting rule combinations.

13.5 Limitations

The initial set of 79 survey games used to determine player preferences included a variety of rule combinations and end conditions, however most of these source games were viable and of reasonable quality. This bias towards high quality input may be beneficial in encouraging high quality output, however the under-representation of low quality input may reduce the chances of low quality outputs being recognised as such.

In addition, the participants who undertook the survey constitute only a small sample of the broader population of individuals capable of playing two combinatorial games and making an informed preference of one over the other (i.e. most living individuals). The results obtained from this particular sample do not necessarily have general applicability over the broader population.

The Ludi game description language has some limitations in its scope, and not all of the language was implemented for the Ludi general game player. The GDL allowed a sufficiently wide range of combinatorial games for the purposes of this project, however it is not hard to find existing games that would require some expansion of the GDL to accommodate their particular rules.

A limitation of the Ludi general game player was the speed of legal move generation and board evaluation for some of the more complex games (due to the need for generality), making the Ludi player unlikely to achieve a high level of play for some games within a reasonable amount of time. This did not unduly affect the current project for which the requirement was for strong beginner rather than expert play in self-play trials, but it did rule out the use of the UCT algorithm for move planning.

13.6 Future Research

The results of this study immediately open up a number of avenues for future research.

It would be preferable to base human player preference scores on a more representative sample of the broader population. The BoardGameGeek online database of games [Alden, 2000] provides such a sample as it ranks 4,340 board games based on votes by thousands of players worldwide. This information might be used to determine a more universally representative set of aesthetic predictors, however it would require the description of these thousands of games in a format suitable for the measuring system – a huge undertaking!

In addition, the human player preference scores provided by the BoardGameGeek database, while numerous and readily available, are not without contention. For example, the game of Hex, one of the most influential games of the previous century and widely regarded to be a game of exceptional quality, has a surprisingly low ranking of #987. It is also worth noting that the games in this collection may vary in quality but are almost exclusively viable games, hence subject to similar biases inherent in the smaller data set of 79 games used in this study.

The UCT algorithm has great potential for move planning in general games that overcome several of the limitations previously outlined. The UCT approach only requires the following information for a given game:

- The legal moves for a given board position, and
- Whether a given board position ends the game.

A UCT-based player would need to be optimised for speed in random playouts but would present a promising new direction for general game players that avoids the need for advisors and policies, which are sticking points for most existing general game systems (including Ludi).

Further, a UCT-based player could avoid the need for a game description language – typically the most limiting factor of any general game system – by using compiled or uncompiled scripts or code modules to define a game's board, starting position, legal moves and winning conditions. Tasks such as move generation and random playouts could then be optimised for particular rule sets. Genetic programming techniques could still be used to evolve games represented in this way; this would, in fact, be more in line with their intended purpose of modifying programmes rather than games.

Although a large number of aesthetic criteria were implemented for the study, there are many additional criteria that may be considered, and alternative ways to implement some of the existing ones. Quality measurement in multiplayer and video games are areas of significant interest.

Enhancing the system to conduct automatic analyses of particular design flaws, such as susceptibility to mirror strategies, might also be a beneficial tool with which game designers could test prototypes. Further, the automatic correction of such flaws, for instance by adjusting board size/topology, starting position, piece distribution, movement rules, and so on, might also prove useful.

13.7 Summary

The results of this study support both hypotheses: *that there exist fundamental (and measurable) indicators of quality for combinatorial game design,* and *that these fundamental indicators may be harnessed for the directed search of new high quality combinatorial games*.

A complete aesthetic model for the measurement of game quality is proposed and implemented, in addition to a complete process for the automated creation of new high quality games. It is

demonstrated that game viability is a precondition for measuring game quality. The importance of relaxing rule optimisation during the evolutionary process is demonstrated, in order to encourage the emergence of new and interesting rule combinations.

Future work might include the correlation of game measurements with a broader range of human player preferences, and investigation of the UCT algorithm as the next step in the development of truly universal general game players that transcend the limitations of the traditional GDL and advisor/policy approaches.

Appendices

Appendix A

Formal Definition of the Ludi GDL

This appendix presents a complete definition of the Ludi Game Description Language (GDL).

Text that is greyed out indicates sections of the grammar not implemented for the Ludi general game player.

A.1 Syntax

The following section describes the essential syntax of the Ludi GDL.

A.1.1 Definitions

The symbol → matches a declaration (LHS) with its definition (RHS).

UPPERCASE	→	Symbol types.
lowercase	→	Rule clause (ludeme) defined by the language.
Capitalised	→	Data type.
italicised	→	Keyword, operator, function or built-in clause identifier.
Italicised Capital	→	Player identifier or name.
[item]	→	Optional item.
|	→	Choice (disjunction).
{a | b | c …}	→	Exactly one of the listed items.
(clause)	→	Rule clause (ludeme) bracketed for scoping purposes.
%*n*	→	Variable item to be instantiated as the n^{th} argument.

The plural of any item denotes a list of one or more of those items. For example:

items	→	item [items]
CHARs	→	CHAR [CHARs]

A.1.2 Symbol Types

INT	→	Signed integer.
UINT	→	Unsigned integer.
FLOAT	→	Floating point number.
NUMBER	→	INT | UINT | FLOAT
BOOL	→	0 | 1
CHAR	→	ASCII character in the code range 32..128 (not including space).
STRING	→	CHARs
NAME	→	STRING // Unique name
ITEM	→	NUMBER | BOOL | FLAGS | STRING

A.2 Data Types

The Ludi GDL defines the following data types and records.

PlayerType → *White* | *Black* | *Grey* | *Red* | *Green* | *Blue* | *Cyan* | *Magenta* | *All* | *None* | *Neutral* | *Current* | *Friend* | *Enemy* | NAME

// Corresponds to players 1, 2, 3, 4, 5, 6, 7, 8, all, none, neutral, etc.
// *Current* refers to the current player.
// NAME matches the unique name of a player.

PlayerState → *active* | *resigned*

CompassDirection → *n* | *s* | *e* | *w* | *ne* | *se* | *nw* | *sw*
// Absolute compass directions.

TurtleDirection → *f* | *b* | *l* | *r* | *fl* | *fr* | *bl* | *br*
// Turtle steps relative to the current player's direction of travel
// (forward, back, left, right, etc).

PieceType → NAME | *any* | *current* | *captured*
// Either a predefined piece identified by name, or any piece.
// *current* specifies the current piece in a piece definition.
// *captured* specifies the set of pieces captured this turn.

PieceRecord → (PieceType [PlayerType] [(*state* UINT)] [(*flags* UINT)] [(*value* INT)])
// NAME specifies a generic piece of that type.
// Record of piece ownership, state, flags and value.

MoveRecord→ (PieceType MoveType [PlayerType])
// Specifies a particular type of move for a particular type of piece.

Direction → CompassDirection | TurtleDirection | *any* | *all* | *d-nbors* | *i-nbors* | *current* | *flagged* | *opposite*
// Direction; may be invalid depending on the current board tiling.
// *current* specifies the current move's direction (ie. *from* to *to*).
// *flagged* specifies directions in which the piece's flags point.
// *opposite* specifies the direction opposite to the current direction.

Step → Direction | (Direction {UINT | *line* | *line-of-sight*})
// Either step one cell in a direction or take a number of steps in a direction,
// may be anywhere along that line or in line of sight (empty cells only).

TilingType → *hex* | *square* | *tri* | *trunc-square*
// *trunc-sq*uare is truncated square tiling (semiregular Archimedean tiling 4.8.8).

ShapeType → *hex* | *square* | *tri* | *rhombus* | *trapezium* | *boardless* | *rot-square*
// *rot-square* is a square rotated 45 degrees.

CoordLabel → CHAR UINT
// Coordinate label by column and row, e.g. D12 or f7.

CoordType → CoordLabel | UINT | *from* | *to* | *current* | (CoordType Steps)
// Board cell coordinate (e.g. F12) or edge index (e.g. 7).
// *from* and *to* are cells or edges relative to the current move.
// *current* is the current cell in an iteration over a test condition.
// The last form describes a coordinate relative to the specified one.

CoordRecord → (CoordType PlayerType State [UINT])
// Record of cell/edge ownership and state.
// Use of the optional UINT value depends on the circumstances (default 0).

RegionType → Direction | (CompassDirection UINT) | CoordType | *home* | (*home* UINT) | *away* | (*away* UINT) | (*dirn* UINT) | *all-sides* | *alternating-sides*
// A connected region of board cells corresponding to either cells along a side, a specific cell, or cells along all sides.
// *home* and *away* are board sides relative to the player's direction.
// The Direction, *home* and *away* UINTs specify the number of rows out.
// *alternating-sides* allocates even sides to White and odd sides to Black.

RegionSet → {
UINTs |
own-regions | *all-sides* | *opposite-sides* | *alternating-sides* |
(*n-of* {UINT | *half* | *majority*} UINTs)
}
// Combination of one or more regions specified by index or description.
// Descriptions are expanded into region child trees.
// *n-of* returns true if the specified number of conditions is satisfied.

RegionRecord → (PlayerType {RegionType | RegionSet})
// Record of region ownership.

CaptureType → *replace* | *jump* | *surround* | *nbors* | *d-nbors* | *i-nbors* | *edges* | *cells* | *connected* | *swap* | *mimic* | *cap* | *any* | *all* | *this*

// *replace* = capture topmost enemy pieces by replacement.
// *jump* = capture topmost enemy pieces jumped over.
// *surround* = capture topmost enemy pieces surrounded (d-nbor freedoms only).
// *nbors/ d-nbors/ i-nbors* = capture enemy neighbors upon landing.
// *edges / cells* = capture edges/cells moved to.
// *connected* = capture connected enemy pieces.
// *swap* = the capturing piece swaps with the captured piece.
// *mimic* = the capturing piece takes on the properties of the captured piece.
// *cap* = capture enemy line capped at both ends.
// *any* = capture any single enemy piece on the board.
// *all* = capture all pieces landed on.
// *this* = capture or convert the moving piece.

ResultType → *win* | *lose* | *draw*

Tree<x> → (
x |
(*and* Tree<x>) | (*or* Tree<x>) | (*not* Tree<x>) |

```
        (if x Tree<y> Tree<z>) |
        (n-of {UINT | half | majority} Tree<x>)
    )
    // Logical and/or tree in which each node is either an instance of x or a child.
    // n-of combinations are expanded to the corresponding child trees.

Alliance  →  (friend NAME NAMEs)
             // List of names forming an alliance.
```

A.3 Functions

Functions return either a boolean or integer value, and can be inserted in lieu of any atomic value of the same type. Boolean functions are generally used for piece movement preconditions.

```
bool_function →  {
        BOOL |
        (not BOOL) | (and BOOL BOOLs) | (or BOOL BOOLs) |
        (xor BOOL BOOLs) |
        (= INT INTs) |  (!= INT INT) |
        (< INT INT) | (<= INT INT) | (> INT INT) | (>= INT INT) |
        (empty CoordType) |
        (occupied CoordType) |
        (owner CoordType [PlayerType]) |
        (enemy CoordType [PlayerType]) |
        (friend CoordType [PlayerType]) |
        (neutral CoordType) |
        (adjacent [i-nbors] [d-nbors]) |
        (connected [PlayerType]) |
        (in-region CoordType RegionSet) |
        (line [PlayerType] [(dirn Direction)] [i-nbors] [d-nbors]) |
        (steps TurtleDirections) |
        (contains CoordType PieceRecord [UINT]) |
        (has-freedom) |
        (creates-freedom) |
        (no-ko) |
        (forwards) |
        (backwards)
    } → returns BOOL

    // A friend is a piece or cell belonging to the player or an ally.
    // Two sites are connected if one can be reached from the other via a series of
          adjacent steps through sites owned by the player.
    // Moves forwards and backwards are relative to the current player's
          direction.
    // The line may lie along i-nbors or d-nbors or both (default).
    // steps are turtle directions through adjacent d-nbors (i-nbors can still be
          reached by a series of such steps).
    // knights-move returns true if to and from are a knight's move apart.
    // steps may contain both compass (absolute) and turtle (relative) dirns.
    // An example of a knight's move may be (steps n n e).
    // flagged-dirn returns true if the piece has a flag in that direction.
    // in-region returns true if the specified site is in the specified region.
```

// *contains* returns true if the specified site contains the specified type (and optionally number) of pieces.
// *has-freedom* returns true if the *to* cell is connected to at least one empty cell.
// *creates-freedom* returns true if a move *to* will remove all freedoms from a group to create a freedom (needed for the Go capture mechanism).
// *no-ko* returns true if the current board position is not identical to that last turn.
// *forwards* and *backwards* return true if the move *from-to* is forwards/backwards relative to the player's current direction.

int_function → {
INT |
(+ INT INTs) | (- INT INT) | (* INT INTs) | (/ INT INT) | (% INT INT) |
(*max* INT INTs) | (*min* INT INTs) |
from | *to* | *current* |
(*from* Directions) | (*to* Directions) |
(*owner* CoordType) |
(*height* CoordType) |
(*num-pieces* CoordType {PlayerType | PieceRecord}) |
(*num-nbors* CoordType PlayerType) |
(*num-edges* CoordType State) |
(*num-moves*) |
(*mover*) |
(*distance*) |
(*dist-to* UINTs) |
(*phase* CoordType) |
(*num-flags* [PieceRecord]) |
(*state* CoordType) |
(*piece-state* [CoordType]) |
(*die* [*double*]) |
(*if* bool_function [int_function [int_function]]) |
(*total-cells*) |
(*board-width*) |
(*board-height*) |
(*min-dim*) | (*max-dim*) |
(*group-size* CoordType) |
(*nbors* CoordType [int_function]) |
(*line* [int_function]) |
(*capture-num*) |
(*capture-value*) |
(*num-between* PlayerType) |
(*territory* [*no-i-nbors*] [*empty*] [*use-edge*]) |
(*row* CoordType) |
(*col* CoordType)
} → returns INT

// *from* and *to* returns the index of the move's "from" and "to" cells. These will be the same if the move is a placement.
// *current* returns the index of the current cell in a *line* or *nbors* iteration.
// (*from* Directions) returns the index of the cell reached after taking the specified steps from the "from" cell.
// (*to* Directions) returns the index of the cell reached after taking the

specified steps from the “to” cell.
// *owner* returns the stack height at the specified coordinate.
// *height* returns owner of the specified cell: 0=Empty, 1=White, 2=Black…
// *num-pieces* returns the number of specific pieces, or pieces belonging to a specific player.
// *num-nbors* returns number of adjacent pieces of a specific type or player.
// *num-edges* returns the number of edges in a particular state for that site.
// *num-moves* returns the current move count.
// *mover* returns the index of the player to move.
// *distance* returns minimum number of adjacent steps required to move from the first site to the second.
// *dist-to* returns the minimum number of adjacent steps to the specified region(s).
// *phase* returns either 1, 2 or 3 depending on the current tiling.
// *num-flags* returns the number of flagged directions on the specified piece.
// *state* returns the current state value of the specified site.
// *piece-state* returns state value of the piece at the specified site (0 if none).
// *die* returns the number of pips on one of the dice, and excludes that die from further use this turn. *double* indicates that doubles count twice.
// *if* bool_function is true then return the value of the first int_function, else return the value of the second. If the int_functions are omitted, then the appropriate boolean *true* or *false* value is returned.
// *min-dim* and *max-dim* return the minimum/maximum board dimension.
// *group-size* returns the size of the group of which the piece at the specified coordinate is a member, else 0 if none.
// *nbors* returns the total value of the specified function applied to all adjacent cells (if given) else the number of adjacent non-empty cells.
// *line* returns total value of the specified function applied to all cells in a line between “from” and “to” (if given) else 1 if “from” and “to” are in line. In any event, *line* will return 0 if “from” and “to” are not in line.
// c*apture-num* is the total number of pieces captured this turn.
// *capture-value* is the total value of pieces captured this turn.
// *num-between* returns the number of PlayerType cells between *from* and *to*.
// *territory* returns number of cells enclosed by groups of the player’s pieces.
If *no-inbors* is specified then the cycle must be directly connected.
If *empty* is specified then only empty cells are counted.
If *use-edge* is specified then the board edge close cycles.
// *row/col* return row/column of the specified site (usually *from* or *to*).

Optional [CoordType CoordType] pairs default to *from* and *to* if not specified.
Optional PlayerType values default to *Current* if not specified.
Optional PieceType values default to *Current* if not specified.
Adjacency is defined in the board clause, and may optionally include indirect neighbours.
If a clause is an *if* condition, then *current* iterates over all appropriate cells.

A.4 Grammar

Each game, ludeme and metarule is defined by a single top-level clause as follows.

A.4.1 Game Grammar

game → (*ludeme* NAME [ITEMs] players equipment rules [support])

// ITEMs are optional arguments that propagate throughout the ludeme tree, typically describing common attributes such as the board size.

rules → {
[pieces]
[start]
[play]
end
}
// Describes the pieces, starting position, play order and end conditions.

---

players → (*players* {NAME | (NAME CompassDirection)}s [Alliances])

// Describes the number of players (1..8). Every player must have an entry.
// One or more alliances may optionally exist for multiplayer games.
// If no CompassDirection is specified, the player's default direction is *all*.

---

equipment → { board [dice] }

// Includes the board and optionally dice (pieces are handled separately)

---

board → (*board* [*phase*]
(*tiling* TilingType [*i-nbors*][*knight*-nbors][*no-d-nbors*])
(*shape* ShapeType)
(*size* UINTs)
[(*clip-corners* UINT)]
[(*regions* RegionRecords)]
[NAME]
)
// If *phase* is specified, then phases are shown.
// If *i-nbors* is specified, then indirect (diagonal) neighbours are included.
// If *knight-nbors* is specified, then knight's move neighbours are included.
// If *no-d-nbors* is specified, then direct (adjacent) neighbours are removed.
// The *size* UINTs specify the board size in cells per side.
// The *clip-corners* UINT specifies how many rows are to be clipped.
// NAME is the name of a ludeme from which this game's board is defined.

---

dice → (*dice* UINT*d*UINT [*enhanced-doubles*])

// Describes whether dice are to be used in this game.
// For instance (*dice* 2*d*6) will cause two six-sided dice to be rolled each turn.
// If *enhanced-doubles*, then twice the die count is used when a double is rolled.

---

pieces → (*pieces* piece_defns [NAME])

// NAME is the name of a ludeme from which this game's pieces are defined.

```
piece_defn →        (NAME PlayerType
                        [(label STRING)]
                        [(value INT)]
                        [(state State)]
                        [(flags Flags)]
                        (moves move_defns)
                    )
                    // This piece's name (unique within the game), its owner, and its moves.
                    // The piece may also have optional state, flags, and a label that is displayed.
                    // The piece's value is used in the material evaluation (default 1).

move_defn →         (move [(priority UINT)] [mandatory]
                        [(label STRING)]
                        [(dirn Direction)]
                        [(owner PlayerType)]
                        [(pre bool_function)]
                        (action {pass | action_defns})
                        [(post post_conditions)]
                    )
                    // dirn specifies the direction to which this move is relative (default player's
                        current direction).
                    // owner specifies who must own the piece being moved (default is current).
                    // The preconditions, actions, and postconditions that make up a move.
                    // Mandatory moves take precedence over non-mandatory ones.
                    // Higher priority moves take precedence over lower priority ones.
                    // Note: from and to are CoordTypes defined with each move, and are
                        assumed to be different.

action_defn →       {
                        (pop [CoordType] [UINT | all]) |
                        (push [CoordType] [UINT] [PlayerType] [trail])
                    }
                    // pop a number of pieces (default 1) from a site (default from).
                    // push a number of pieces (default 1) to a site (default to), optionally laying
                        them in a trail by unstacking along the line of movement.

post_condition →    {
                        (capture
                            CaptureTypes [PieceRecord] [optional] [subsume] [recycle]
                            [(if fn)]
                        ) |
                        (convert
                            {
                                [PlayerType] State |
                                CaptureTypes [PieceRecord] [PlayerType] State
                            }
                            [optional] [(if fn)]
                        ) |
                        (rotate {INT | any} [optional] [(if fn)]) |
                        (change-dirn Direction) |
                        (displace Direction [PieceRecord] [UINT] [optional] [(if fn)]) |
                        (cell-state CoordType [PlayerType] State [optional] [(if fn)]) |
```

(*piece-state* [CoordType] [PlayerType] INT [*optional*] [(*if* fn)]) |
(*inc-state* [CoordType] [PlayerType] [(*amount* INT)] [(*mod* UINT)]
[*optional*] [(*if* fn)]) |
(*add-flags* [CoordType] Flags [*optional*] [(*if* fn)]) |
(*remove-flags* [CoordType] Flags [*optional*] [(*if* fn)]) |
(*score* int_function [(*if* fn)]) |
(*repeat* [*optional*] [(*if* fn)]) |
(*swap*)
}
// *capture*: If no PieceRecord, then all affected enemy pieces are captured.
Captures are relative to *to*.
recycle specifies that captured pieces go back in the hand.
subsume is a very powerful option – the piece subsumes all
movements and capabilities of the captured piece.
// *convert*: Either convert this piece to the specified owner and state, or
convert all capturable pieces of the specified type.
Note: Pieces to be converted to must be listed before this piece.
This may cause circular definitions.
// *rotate*: rotate the piece the specified number of increments clockwise so
that its flags point elsewhere.
// *change-dirn*: Change the player's direction.
// *displace*: Push pieces away. Specify *all* for all directions.
// *cell-state*: Set the state (and optionally owner) of the specified cell.
// *piece-state*: Set the state (and optionally owner) of the top piece at coord.
// *inc-state*: Increment the state of the topmost piece at coord.
// *add/remove-flags*: Set flags of topmost piece at the specified cell/edge.
// *score*: Update the player's score by a specified amount, e.g. 1 or
(capture-value).
// *repeat* specifies that the player may repeat this move type
(e.g. multiple hops).
// Postconditions are mandatory unless specified as *optional*.
// The *if* function, if given, must be satisfied for the postcondition to occur.
// *swap* specifies that the pieces at the *from* and *to* cells swap positions.

---

start → (*start* [place_clauses]s [in_hand_clauses]s [claim_clauses]s)

place_clause → (*place*
PieceRecord [UINT] [(*phase* UINT)]
{
home | (*home* UINTs) | CompassDirn | (CompassDirn UINTs) |
(*region* UINTs) | (*site* UINTs) | CoordLabel | *opposed*
}s
)
// Place the specified piece(s) at the specified cells.
// The PieceRecord UINT specifies the number of pieces.
// If *phase* is specified, then pieces are only placed on cells of that phase.
// The *home* UINTs refer to the number of rows out from the home edge.
// The *dirn* UINTs refer to the number of rows out from the edge in that dirn.
// Note: *dirn* may specify corners.
// The *region* UINTs refers to region indices.
// The *site* UINTs refers to site indices.
// *opposed* placements are made in opposed corners of the board.

```
in_hand_clause →    (in-hand PieceRecord UINT)
                    // Number of pieces the player starts with in-hand (default unlimited).

claim_clause  →     (claim
                      PlayerType
                          {
                              home | (home UINT) | dirn | (dirn UINT) |
                             (region UINT) | (site UINT) | CoordLabel
                          }s
                    )
                    // Claims cell(s) for the specified player.
```

```
play        →       (play
                            [(MoveType [repeat])s]
                            [(swap UINT)]
                            [(progressive UINT)]
                            [can-pass]
                    )
                    // Cyclically perform the specified move types until the game ends.
                            Default: cycle through each player in turn, allowing one of any type
                            of legal move per turn.
                    // The repeat option means that that particular move may be repeated.
                    // If the swap option is specified, then the UINT value indicates how many
                            moves must elapse before the swap may be taken.
                    // Progressive games start with 1 move per turn, then 2, 3 etc up to UINT.
                    // If can-pass, then players may pass in lieu of moving. The game ends if all
                            players pass in succession, and the leading player wins.
```

```
end         →       (end end_clauses [mover-wins | mover-loses | tie] [opp-turn])
                    // Tiebreakers:
                    //        mover-wins: The player who just moved wins (default).
                    //        mover-loses: The next player wins.
                    //        tie: The game is tied.
                    // If opp-turn is specified, then a player wins only if their end_clause is met
                            on the opponent's turn.

end_clause  →       (PlayerType ResultType Tree<end_condition>)

end_condition →     {
                            (connect
                                    [PieceRecord] [d-nbors | i-nbors | all-nbors] [flagged]
                                    RegionSet [stack]
                            ) |
                            (cycle [PieceRecord] [UINT] [full] [d-nbors | i-nbors | all-nbors]) |
                            (group [PieceRecord] [UINT] [d-nbors | i-nbors | all-nbors] [stack]) |
                            (n-in-a-row [PieceRecord] UINT [d-nbors | i-nbors | all-nbors]) |
                            (pattern [PieceRecord] Steps) |
                            (reach
                                [PieceRecord] [UINT| all| fill] [stacks]
                                {
```

```
                    away | (away UINT) | CompassDirn | (CompassDirn UINT)
|
                    (region UINT) | (site UINT) | CoordLabel |
                    all-sides | (all-sides UINT)
                }s
        ) |
        (capture [PieceRecord] [UINT | all]) |
        (eliminate PlayerTypes)
        (score { UINT | best }) |
        (stack [PieceRecord] UINT [owner]) |
        (state { CoordRecord | PieceRecord } ) |
        (no-move)
}
// connect: Connect all specified regions (must be at least two).
        If a PieceRecord is given, then only connect that piece type.
        If i-nbors or d-nbors are specified, only those neighbours connect.
        If flagged is specified, only currently flagged directions are allowed.
        If stack is specified then all pieces in stacks are counted, else only the
                topmost piece of each stack is examined.
// cycle: Form a cycle, or a full cycle if full is specified.
        If a PieceRecord is given, then the cycle must consist of that piece.
        The UINT value, if given, specifies area that the cycle must enclose.
// group: Gather pieces together into a single connected group.
        If a PieceRecord is given, then that piece type must be gathered.
        The UINT value, if given, specifies the number of pieces to be
                gathered (default all).
         The STRING value, if given, specifies type of piece to be gathered.
        If stack is specified then all pieces in stacks are counted, else only the
                topmost piece of each stack is examined.
// n-in-a-row: Form a line of n pieces (of PieceRecord, if given).
// pattern: Form a pattern of pieces defined by the Steps.
// reach: The optional UINT value specifies the number of pieces that must
        reach the target regions (default 1).
        If all, then all pieces must reach the target region(s).
        If fill, then all target region(s) must be filled.
        If stacks, then only stacks are counted (buried pieces are ignored).
        If a PieceRecord is given, then only that type of piece is counted.
// capture: Capture piece(s) of the specified type and state.
        If no piece is specified, then all enemy captures are counted.
        The optional UINT value specifies the number of pieces (default 1).
        If all, then all specified pieces must be captured.
        If all and no piece is specified, all enemy pieces must be eliminated.
// eliminate: Eliminate the specified player(s).
// end_score: Achieve a specified score.
        If UINT is specified then end as soon as that score is reached, else
        if best is specified then best score wins/loses.
// stack: Achieve a stack of the specified height.
        If owner is specified, then only the owner's pieces in the stack are
        counted.
// state: Achieve the specified cell/edge or piece state.
// no-move: Current player has no legal moves.
```

support → {
[advisors]
[description]
[aim]
[ancestry]
[ranking]
[viable]
[score]
}
// Supplemental support information including the game's policy, evaluation, ancestry, help, and so on.

---

advisors → (*advisors* (NAME FLOAT)s)

// Describes a weight vector for advisors relevant to this game.
// Each FLOAT describes an advisor weight (no range). Others are set to 0.
// Valid advisor names include:
conn: Connection advisor.
cycle: Cycle advisor.
group: Group advisor.
nina: N-in-a-row advisor.
linecap: Line cap advisor (enemy cap potential).
capt: Capture advisor (target pieces only).
mater: Material advisor (all pieces).
proxe: Proximity advisor (average distance to enemy pieces).
proxm: Proximity advisor (average distance middle of the board).
proxs: Proximity advisor (average distance to board sides).
proxc: Proximity advisor (average distance to board corners).
proxh: Advancement advisor (average distance from home region).
proxg: Proximity to goal (distance to goal).
scor: Score advisor.
state: State advisor.
stack: Stack advisor.
mobil: Mobility advisor (number of legal moves).
infl: Influence advisor (number of empty sites landed upon).
attack: Attack advisor (number of enemy sites landed upon).
defen: Defence advisor (friendly sites landed upon by friends).
threat: Defence advisor (friendly sites landed upon by enemies).

---

description → (*description* STRINGSs)

// English description of the game, to be displayed as a help file to the user.
// Each STRING is a separate paragraph.

---

aim → (*aim* STRINGSs)

// Brief summary of the aim of the game. This, in conjunction with explicitly shown legal moves, is intended to help new players play the game.
// Each STRING is a separate paragraph.

Ancestry → (*ancestry*
(*parents* NAME NAME)
(*maturity* UINT)
(*dist* FLOAT FLOAT FLOAT)
)
// Details regarding the ancestry of the game.
// *parents* are the two parent games from which this game was derived.
// If a game has no parents, then it is a source game.
// *maturity* is the maximum number of generations leading to this game.
// *dist* indicates the minimum distance game to:
a) The game's parents,
b) All source games, and
c) The entire population.

---

ranking → (*ranking* FLOAT)

// *ranking* is the ranking of this game within its data set.

---

viable → (*viable* FLOAT)

// *viable* is an indication of the estimated viability of this game [0..1].
// In practice, a value of 1 is used to indicate that the game is viable.

---

score → (*score* FLOAT [(*old* FLOAT)])

// *score* is the aesthetic score of this game.
// If the game has parents then the score is predicted, else the score is known.
// The *old* subelement stores the game's previous score value, if any.
This is for games with an old score to settle, such as those remeasured after optimisation.

A.4.2 Ludeme Grammar

ludeme → (*ludeme* NAME clauses)
// The clauses may be any of those defined above.
// The clauses overwrite the equivalent clauses of the parent game.

Once defined, a ludeme can be used in the language as a clause.

A.4.3 Metarule Grammar

metarule → (*metarule* NAME meta_clauses)

meta_clause → (clause {{*set* | *reset*} meta_arg meta_arg | {*add* | *remove*} meta_arg})
// The clause may be any of those defined above.
// *set/reset* or *add/remove* elements of the specified clause.

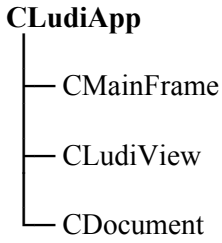
meta_arg → ITEM | (clause)

Appendix B

Ludi Class Structure

This appendix describes the basic class structure of the Ludi general game player.

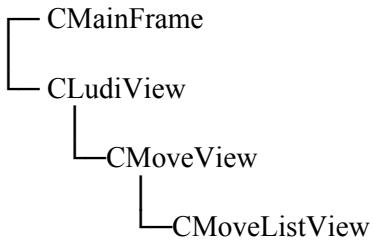
The Ludi general game player was written in C++ using the Visual C++ 6.0 compiler and Microsoft Foundation Classes (MFC). The basic structure of the application follows the standard MFC *single document interface* (SDI) design, as shown below.



Objects are single instances unless suffixed by [s] to denote a collection of zero, one or more items.

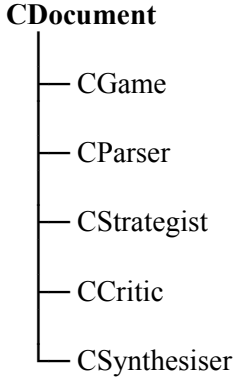
User Interface

The user interface for the Ludi player is structured as follows:



The CLudiView window is the main board display showing the current state of the currently loaded game. The CMoveView window, embedded in the main board display, shows the player status and move history for the current game. The CMoveListView window, embedded in the CMoveView window, shows the move history for the current game.

Internal Data

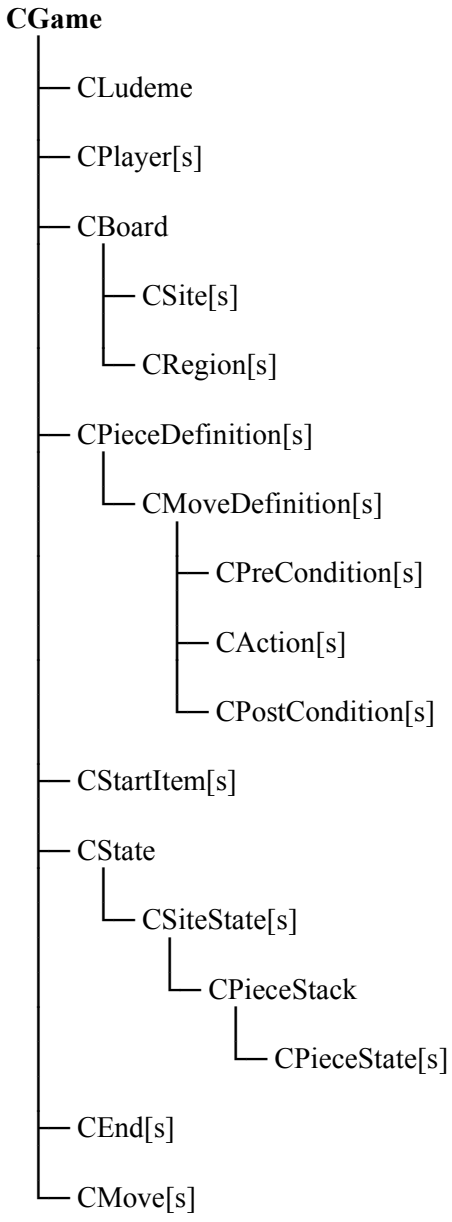


The CDocument class encapsulates the application's internal data. Most importantly, this includes the CGame object and its supporting modules.

Since the application is based on a single document interface, then only a single document and hence a single game can be loaded in the Ludi player at any time.

Game Object

The game object is a single instance of the CGame class, structured as follows.



The CLudeme object is the root of the game's ludeme tree. The CPlayer[s] collection contains details and search settings for each player. The game has a single CBoard object consisting of collections of CSite and CRegion objects.

The CPieceDefinition[s] collection contains the piece definitions. Each piece definition consists of a collection of CMoveDefinition objects, each of which consists of collections of CPrecondition, CAction and CPostCondition objects.

The CStartItem[s] collection contains starting information such as initial piece placements and number of pieces in hand. Pieces are identified by unique name.

The CState object describes the current board state of the current game, which is represented by a collection of CSiteState objects corresponding to each board cell. Each site state contains a CPieceStack object describing the pieces stacked on that cell, if any. This takes the form of a collection of CPieceStack objects describing the type and state of each piece ordered bottom-to-top.

The CEnd[s] collection describes game's end condition, ordered by player. Each end condition constitutes a logical tree of conditions that must exist for that end condition to be satisfied.

The current game's move history is described by a collection of CMove objects. The CState and CMove[s] objects are updated after each move. All other objects are initialised when the game is loaded and not modified thereafter.

Appendix C

Source Games

This section lists the 79 source games used for the survey component of Experiment I and the initial population of Experiment III. The **Id** column gives each game's index which was used to identify games for survey participants in lieu of game names, in order to reduce user bias. The **Name** column refers to the game's name or the name of the source game it was derived from.

The **Score** column shows each game's estimated score in the range [-0.25 .. 0.25] based on paired comparison results returned by the survey, and **Varn** shows the variance within each estimated score. The final column shows the game's **Rank** compared to other games (lower is better).

The mean classification rate for ranking this set of games using the method described in Appendices C and D was 0.899685 (variance = 0.000317653) after 100 epochs.

Id	Name	Score	Varn	Rank
000	Breakthrough	0.1575	0.0019	#9
001	Chameleon (11x11)	0.0447	0	#35
002	Chess	0.1102	0	#18
003	4-in-a-Row (Line Move, Capture, 10x10)	0.0438	0	#36
004	Gomoku (Hexhex, Conn. Move, Capture)	0.2335	0.0002	#1
005	Hex (Connected Move, Capture)	0.0184	0	#43
006	Capture 5 (Hexhex, Conn. Move, 5inHand)	0.0661	0.0004	#29
007	Capture 5 (Hexhex, Conn. Move, Unbal.)	-0.0319	0.0001	#52
008	Capture All (Hexhex, Conn. Move)	-0.1042	0.0002	#67
009	Fox & Geese (Square)	0.0926	0.0002	#23
010	Fox & Geese (Square, 2 Rows)	0.0587	0.0003	#32
011	Fox & Geese (Square, 2 Rows, No Diags)	-0.0380	0	#53
012	Fox & Geese (Square, 3 Rows, No Diags)	-0.0962	0.0011	#65
013	Fox & Geese (Hexhex, 3 Rows)	0.0309	0.0001	#38
014	Fox & Geese (Hexhex)	-0.0814	0.0009	#63
015	Fox & Geese (Truncated Square, 2 Rows)	0.0077	0.0001	#45
016	Fox & Geese (Truncated Square, 3 Rows)	0.0217	0.0002	#41
017	Gomoku (Line Move, Capture, 8x8)	0.0647	0	#30
018	4-in-a-Row (Line Move, Capture)	0.0980	0	#20
019	4-in-a-Row (Move, Piece State)	0.0955	0.0002	#21
020	Capture All (Hexhex, Move, Piece State)	0.0001	0.0028	#47
021	4-in-a-Row (Line Move, Capture)	0.0010	0.0002	#46
022	Gomoku (5x5)	-0.1899	0.0003	#79
023	Gomoku (11x11)	0.0643	0.0003	#31
024	Gomoku (19x19)	0.1366	0.0001	#14
025	Gomoku (Hexhex)	0.0436	0.0013	#37
026	Gomoku (Move)	0.1145	0	#17
027	Gomoku (Move, Capture)	0.1034	0.0002	#19
028	Gomoku (Move, Group)	0.0860	0.0004	#25
029	Gomoku (Move, 12 in Hand)	0.0822	0	#26
030	Gomoku (Move, 6 In Hand) - Bad	-0.0944	0.0008	#64
031	Gomoku (Move, Stack)	0.1502	0	#12

032	Gomoku (Phased)	-0.1241	0.0005	#72
033	Gomoku (Truncated Square)	0.0745	0.0010	#28
034	Hex (17x17)	0.1307	0.0001	#15
035	Hex (Hexhex Board)	0.1559	0.0004	#10
036	Hex (Move)	0.0273	0.0004	#39
037	Hex (Move, 8 In Hand)	0.1996	0.0005	#5
038	Hex (Move, Capture)	-0.1465	0.0077	#75
039	Hex (Square, No Diagonals) – A	-0.0742	0.0007	#61
040	Hex (Square, With Diagonals) – A	-0.0011	0	#48
041	Reach Any Side (Hexhex)	0.0093	0.0003	#44
042	Reach Any Side (Hexhex, Small)	-0.1675	0.0003	#78
043	Hex (Square, No Diagonals) – B	-0.0591	0.0003	#55
044	Hex (Square, With Diagonals) – B	-0.0147	0	#49
045	Gomoku (Surround Capture)	0.1452	0	#13
046	Gomoku (Has Freedom, Surround Capt.)	0.1641	0	#8
047	Kill the King (Conn. Move, Capture LT)	-0.0702	0.0014	#60
048	Gomoku (Makes Freedom, Surr. Capt.)	0.1502	0.0014	#11
049	Stack 5 (Line Convert)	0.1769	0	#7
050	Capture All (Convert Neighbours)	0.0861	0.0004	#24
051	Capture 2 (Group Size)	0.0788	0.0001	#27
052	Capture 2xA + 3xB (Phased Moves)	-0.0605	0.0002	#56
053	Form A Group (Hexhex, Conn. Moves)	-0.0624	0.0015	#59
054	Reach Far Side (Adjacent Moves)	-0.0609	0.0005	#57
055	Reach Far Side (Must Move)	-0.0218	0.0002	#51
056	Stack 3 (Adjacent Moves)	-0.0995	0.0002	#66
057	Capture 5 (Jump Capture)	0.2183	0.0018	#3
058	Reach Far Side (Diagonal Moves, 3 Rows)	0.0210	0.0002	#42
059	Reach Far Side (Diagonal Moves, 2 Rows)	0.0947	0	#22
060	Reach Far Side (Diagonal Moves, Small)	-0.1143	0.0005	#68
061	Form A Group (Knight Moves, Capture)	0.1896	0.0001	#6
062	Form A Group (Knight Moves)	0.0491	0.0025	#33
063	Connect 2 Corners (Square)	-0.0454	0.0002	#54
064	Capture All	-0.0153	0	#50
065	Tic Tac Toe	-0.0617	0.0001	#58
066	Tic Tac Toe (No Diagonals)	-0.1260	0.0004	#73
067	Reach Far Side (Trivial) – A	-0.1218	0.0001	#71
068	Reach Far Side (Trivial) – B	-0.1172	0.0015	#69
069	Reach Far Side (Trivial) – C	-0.1351	0.0005	#74
070	Reach Far Side (Trivial) – D	-0.1496	0.0004	#77
071	Reach Far Side (Trivial) – E	-0.0787	0.0014	#62
072	Reach Far Side (Trivial) – F	-0.1490	0.0001	#76
073	Reach Far Side (Trivial) – G	-0.1181	0.0013	#70
074	Truncated Square Hex	0.1299	0	#16
075	Unlur	0.2178	0.0002	#4
076	Y (11 per side)	0.2186	0.0001	#2
077	Form A Group (Knight Moves)	0.0265	0.0002	#40
078	Hex (Move Enemy)	0.0449	0.0003	#34

Appendix D

Cross-Entropy Method for Paired Comparisons

The following section introduces a cross-entropy (CE) method due to Frederic Maire for inducing a value function from a set of ordered pairs.

Let $T = \{(x_i, y_i)\}_{i=1\ldots m}$ be a set of ordered pairs of R^k such that $V(x_i) \geq V(y_i)$, where $V(x)$ denotes the value function that we want to induce from the training set T.

For example, an ordered pair (x_i, y_i) could be a pair of games in which case x is the sparse coding of a game, or a pair of board positions in which case x is the feature vector of a board position.

The value function $V(x)$ to be induced is of the form $V(x) = w^T x$

We want to find a w such that for each pair (x_i, y_i), we can find a scalar b_i such that the hyperplane $w^T x + b_i = 0$ separates (x_i, y_i).

$\begin{cases} w^T x_i + b_i > 0 \\ w^T y_i + b_i < 0 \end{cases}$ can be rewritten as $\begin{cases} w^T x_i + b_i \geq 1 \\ w^T y_i + b_i \leq -1 \end{cases}$ after normalisation.

as the inequalities can be divided by ε if $\begin{cases} w^T x_i + b_i \geq \varepsilon \\ w^T y_i + b_i \leq -\varepsilon \end{cases}$. Notice that w and b_i are changed in the process.

To maximise the margin between (x_i, y_i) we want to minimise $\|w\|$, as the distance between the two parallel hyperplanes $w^T x + b_i = 1$ and $w^T x + b_i = -1$ going through respectively x_i and y_i is $\frac{2}{\|w\|}$.

Non-negative slack variables a_i are introduced as the training set T might contain contradictions.

The system becomes $\begin{cases} w^T x_i + b_i \geq 1 - a_i \\ w^T y_i + b_i \leq -1 + a_i \\ a_i \geq 0 \end{cases}$.

We want also to minimise $\sum_i a_i$.

To sum up, we want to minimize the cost function $\|w\|^2 + C \sum_i a_i$ with respect to w, b_i, a_i, where C is a regularization parameter, under the constraints:

$$\begin{cases} w^T x_i + b_i \geq 1 - a_i \\ w^T y_i + b_i \leq -1 + a_i \\ a_i \geq 0 \end{cases}$$

Frederic Maire, August 2006

Appendix E includes Matlab code for inducing a ranking scheme of games from the paired comparisons obtained from the survey component of Experiment I, based on this method.

Appendix E

Matlab Code for Ranking Paired Comparisons

The following listing shows the Matlab code used to rank games based on the paired comparisons obtained from the survey conducted in Experiment I. This code implements the cross-entropy (CE) method approach outlined in Appendix D.

```
% testCE.m
%
% Frederic Maire, January 2007

% The optimization toolbox has the following function
% xx = quadprog(HH,ff,AA,bb,Aeq,beq,lb,ub)
% which returns a vector x that minimizes
% 1/2*xx'*HH*xx + ff'*xx, subject to AA*xx <= bb,
% so that the solution xx is in the range lb <= xx <= ub

%     Notation
%     k: common dimension of w, x, and y
%     m: number of training pairs
%     C: regularisation parameter
%     xx = [w' a' b']

% Training set T = {(3,1), (1,4) ,(4,2)}  (sparse coding)
T = load ('out-all-00.txt');

TX = 1+T(:,1)';  % Indexing starts at 1 in Matlab
TY = 1+T(:,2)';  % Indexing starts at 1 in Matlab

k = max([TX TY]);
m = length(TX);

% X: k by m matrix of xi
% Y: k by m matrix of yi
% In others words, X(:,i) Y(:,i) is the ith pair of the training set
X = zeros(k,m);
Y = zeros(k,m);

for i=1:m
    X(TX(i),i) = 1;
    Y(TY(i),i) = 1;
end

% - - - - CE for optimization - - - -

N   = 1000;  % Size of CE sample
rho = 0.1;   % Elite percentile threshold
eliteSize = ceil(rho*N);
maxIter   = 100;
eliteHistory = zeros(2,maxIter);

% EliteHistory(1,iter): Score of best at iter
% EliteHistory(2,iter): Score of worst in the elite at iter

mu    = 0.5* ones(k,1);  % Mean for popu
sigma = 2*ones(k,1);     % Standard deviation for popu

for iter= 1:maxIter
    iter
```

```
    popu = repmat(mu,1,N) + repmat(sigma,1,N) .* randn(k,N);  % CE popn
    normPopu = sqrt(sum(popu.*popu,1));
    popu = popu ./ repmat(normPopu,k,1);  % Normalize
    score = zeros(1,N);  % Performance of the candidates

    % Evaluate the N elements of the CE sample
    for j = 1:N
        w = popu(:,j);
        score(j) = sum((w'*X-w'*Y)>=0); % Count number correctly ranked
    end

    [S,SI] = sort(-score);

    eliteHistory(1,iter) = -S(1);
    eliteHistory(2,iter) = -S(eliteSize);

    % Re-evaluate the CE prob mu
    mu    = mean(popu(:,SI(1:eliteSize)),2);
    sigma = std(popu(:,SI(1:eliteSize)),0,2);
end

bestW     = popu(:,SI(1));  % First from last population
bestScore = abs(S(1))/m     % Proportion of correct pairs

% Display results
figure
plot(eliteHistory(1,:),'b:')
hold on
plot(eliteHistory(2,:),'r:')
title('Evolution of the elite club (10% best of population) ; blue - first
in club, red - last in club')

rr  = bestW'*(X-Y);  % rr should be mostly positive
trEr = sum(rr>0)/length(rr) % training error

figure, bar(rr) % Plot the result
title('Classification of the training pairs by the best w')

[ignored, decrOrd] = sort(-bestW);  % Sort the games in decreasing order
      % decrOrd(j) is the jth best game
% Output the result to file
fp = fopen('out.txt', 'w');
fprintf(fp, 'trEr=%f\r\n', trEr);
fprintf(fp, '%f\r\n', bestW);
fclose(fp);
```

Listing E.1 Matlab code for ranking games based on paired comparisons.

Appendix F

Top Ten Survey Games

This section contains ludeme descriptions for the ten games most preferred by human players out of the initial data set of 79 games used in Experiment I, listed in order of preference. The descriptions have been reformatted for readability and some notations added.

Note that the game names tend to be abbreviated and cryptic as they were intended for internal use only; games were presented to survey participants by index only for the sake of anonymity.

#1: 004 – Gomoku (hexhex board, connected move, capture)

```
(ludeme 004
   (players White Black)
   (board
      (tiling hex)
      (shape hex)
      (size 5)
   )
   (pieces
      (Stone All
         (moves
            (move
               (pre (empty to) )
               (action (push) )
            )
            (move
               (pre (and (owner from) (connected) (<= (height from) (height to) ) )
               (action (pop) (push) )
            )
            (move
               (pre (and (owner from) (connected) (> (height from) (height to) ) )
               (action (pop) (push) )
               (post (capture) )
            )
         )
      )
   )
   (end (All win (in-a-row 5) ) )
)
```

#2: 076 – Y (11 per side)

```
(ludeme 076 11
   (players White Black)
   (board
      (tiling hex)
      (shape tri)
```

```
    (size $1)
    (regions all-sides)
  )
  (play (swap 1) )
  (end (All win (connect all-sides) ) )
)
```

#3: 057 – Capture 5 (jump capture)

```
(ludeme 057
  (players (White n) (Black s) )
  (board
    (tiling square i-nbors)
    (shape square)
    (size 6)
  )
  (pieces
    (Stone All
      (moves
        (move
          (pre
            (and
              (owner from)
              (empty to)
              (line)
              (= (num-between empty) 0)
              (> (distance) 1)
            )
          )
          (action (pop) (push) )
          (post (capture jump) )
        )
      )
    )
  )
  (start (place (Stone All) home (home 1) ) )
  (end (All win (capture 5) ) )
)
```

#4: 075 – Unlur

```
(ludeme 075 6
  (players White Black)
  (board
    (tiling hex)
    (shape hex)
    (size $1)
    (regions all-sides)
  )
  (play (swap 1) )
  (end
```

```
      (White win (connect opposite-sides) )
      (Black win (connect alternating-sides) )
      (White lose (connect alternating-sides) )
      (Black lose (connect opposite-sides) )
   )
)
```

#5: 037 – Hex (move, 8 in hand)

```
(ludeme 037
   (players White Black)
   (board
      (tiling hex)
      (shape rhombus)
      (size 5)
      (regions (White ne) (White sw) (Black se) (Black nw) )
   )
   (pieces
      (Stone All
         (moves
            (move
               (pre (empty to) )
               (action (push) )
            )
            (move
               (pre (owner from) )
               (action (pop) (push) )
            )
         )
      )
   )
   (start (in-hand (Stone All) 8) )
   (end (All win (connect own-regions) )
)
```

#6: 061 – Form a Group (knight moves, capture)

Game #6 involves pieces that make knight moves that capture by replacement. The rule set contains a *phase* ludeme in the *start* section even though cell phase plays no real part in the game; this superfluous ludeme is included purely to trigger the board display to show cell phases in the default checkerboard pattern, in order to visually aid the player in planning knight moves.

```
(ludeme 061
   (players (White n) (Black s) )
   (board
      (tiling square)
      (shape square)
      (size 5)
   )
   (pieces
      (Knight All
```

```
        (moves
          (move
            (dirn all)
            (pre
              (and
                (owner from)
                (or
                  (steps f f r f)
                  (steps f f l f)
                )
              )
            )
            (action (pop) (push) )
            (post (capture replace) )
          )
        )
      )
    )
    (start
      (place
          (Knight White) home (home 1) (home 2) (home 3) (home 4) (home 5) (phase 0)
      )
      (place
          (Knight Black) home (home 1) (home 2) (home 3) (home 4) (home 5) (phase 1)
      )
    )
    (end (All win (group) ) )
  )
```

#7: 049 – Stack 5 (line convert)

```
  (ludeme 049
    (players White Black)
    (board
      (tiling square i-nbors)
      (shape square)
      (size 6)
    )
    (pieces
       (Stone All
          (moves
            (move
              (pre (empty to) )
              (action (push) )
              (post (convert cap) )
            )
            (move
              (pre (and (owner from) (enemy to) (adjacent) ) )
              (action (pop) (push) )
              (post (convert cap) )
            )
          )
```

```
        )
    )
    (end (All win (stack 5) ) )
)
```

#8: 046 – Gomoku (move if freedom, surround capture)

```
(ludeme 046
    (players White Black)
    (board
        (tiling square i-nbors)
        (shape square)
        (size 8 8)
    )
    (pieces
        (Stone All
            (moves
                (move
                    (pre (and (empty to) (has-freedom) ) )
                    (action (push) )
                    (post (capture surround) )
                )
            )
        )
    )
    (end (All win (in-a-row 5) ) )
)
```

#9: 000 – Breakthrough

```
(ludeme 009
    (players (White n) (Black s) )
    (board
        (tiling square)
        (shape square)
        (size 7)
    )
    (pieces
        (Fox All (label "F")
            (moves
                (move
                    (pre (and (owner from) (empty to) (adjacent) ) )
                    (action (pop) (push) )
                )
            )
        )
        (Goose All (label "G")
            (moves
                (move
                    (pre
                        (and
```

```
                    (owner from)
                    (empty to)
                    (or (steps f) (steps f l f) (steps f r f) )
                  )
                )
                (action (pop) (push) )
              )
            )
          )
        )
        (start
          (place (Fox White) D1)
          (place (Goose Black) home)
        )
        (end
          (White win (reach away) )
          (White lose (no-move) )
        )
      )
```

#10: 035 – Hex (hexhex board)

```
(ludeme 035
  (players White Black)
  (board
    (tiling hex)
    (shape hex)
    (size 5)
    (regions all-sides)
  )
  (end (All win (connect opposite-sides) ) )
)
```

Appendix G

Aesthetic Value Correlations

This appendix lists the complete results of the aesthetic value correlations determined in Experiment II. Correlation between the human player preferences obtained in Experiment I and the actual game measurements were determined using linear regression and a standard *leave-one-out* method of cross-validation of, as described in Chapter 11.

In each case, $corr_s$ indicates the correlation between game preference score and measured aesthetic value and $corr_r$ indicates the correlation between game ranking and measured aesthetic value. Each section of this appendix shows the results for a specific category of aesthetic criteria.

All Criteria

There are a total of 57 aesthetic criteria.

Figure G.1 shows plots of predicted versus actual scores and rankings, and Figure G.2 shows the weightings obtained when correlating over all aesthetic criteria.

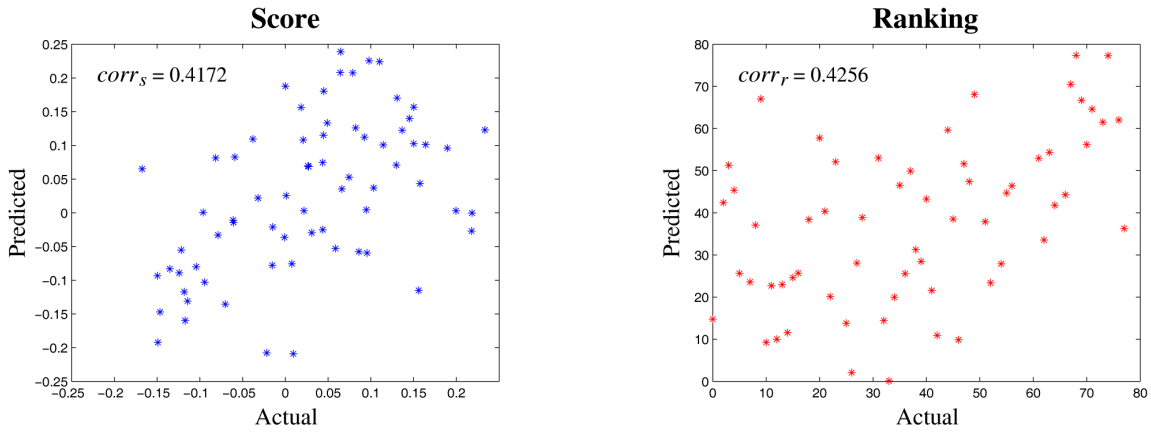


Figure G.1 Correlation of all 57 aesthetic criteria with game scores and rankings.

The game score correlation $corr_s = 0.4172$ over all 57 criteria is reasonable – or at least better than random – but can be significantly improved by using subsets of criteria as predictors. The 95% confidence interval for this correlation is 0.435 [0.176..0.611].

Conclusion: The 57 aesthetic criteria perform to some degree as predictors of game preference score.

Criteria	Weight
0. Complexity	0.1460
1. Goal (connect)	-0.0806
2. Goal (capture)	-0.0820
3. Goal (N-in-a-row)	0.0000
4. Goal (race)	0.1209
5. Goal (group)	-0.1093
6. Goal (stack)	-0.0662
7. Goal (score)	0.0000
8. Goal (block)	-0.1813
9. Piece addition	-0.0020
10. Piece removal	0.0000
11. Piece movement	-0.0280
12. Piece stacking	-0.0024
13. Piece capture	0.0658
14. Piece conversion	-0.0691
15. Piece promotion	-0.0624
16. Branching factor	0.1932
17. Convergence	0.0806
18. Response time	0.0591
19. Move effort	0.2014
20. Search penetration	-0.2601
21. Clarity (variance)	-0.4779
22. Clarity (narrowness)	0.0774
23. Uncertainty (overall)	-1.0133
24. Uncertainty (early)	0.7755
25. Uncertainty (late)	0.4500
26. Drama (average)	0.4415
27. Drama (maximum)	-0.0911
28. Killer moves	0.7380
29. Permanence	-0.0032
30. Lead change	-0.3796
31. Stability	0.2775
32. Decisiveness threshold	-0.0620
33. Decisiveness	-0.1288
34. Depth (discrepancy)	0.0614
35. Foulup factor	0.2367
36. Momentum (1)	0.1301
37. Momentum (2)	0.3392
38. Momentum (3)	-0.6900
39. Correction (1)	0.3098
40. Correction (2)	-0.1137
41. Correction (3)	-0.1626
42. Control	0.0830
43. Coolness	0.1232
44. Board coverage	-0.1547
45. Puzzle quality	0.1020
46. Completion	0.5941
47. Balance	0.1880
48. First move advantage	0.0394
49. Drawishness	0.4634
50. Timeouts	0.4962
51. Duration	-0.1645
52. Depth (skill level)	-0.0235
53. Selfishness	0.0152
54. Obstructive play	-0.0395
55. Resilience	0.0365
56. Puzzle potential	0.0760
Bias	-1.0658

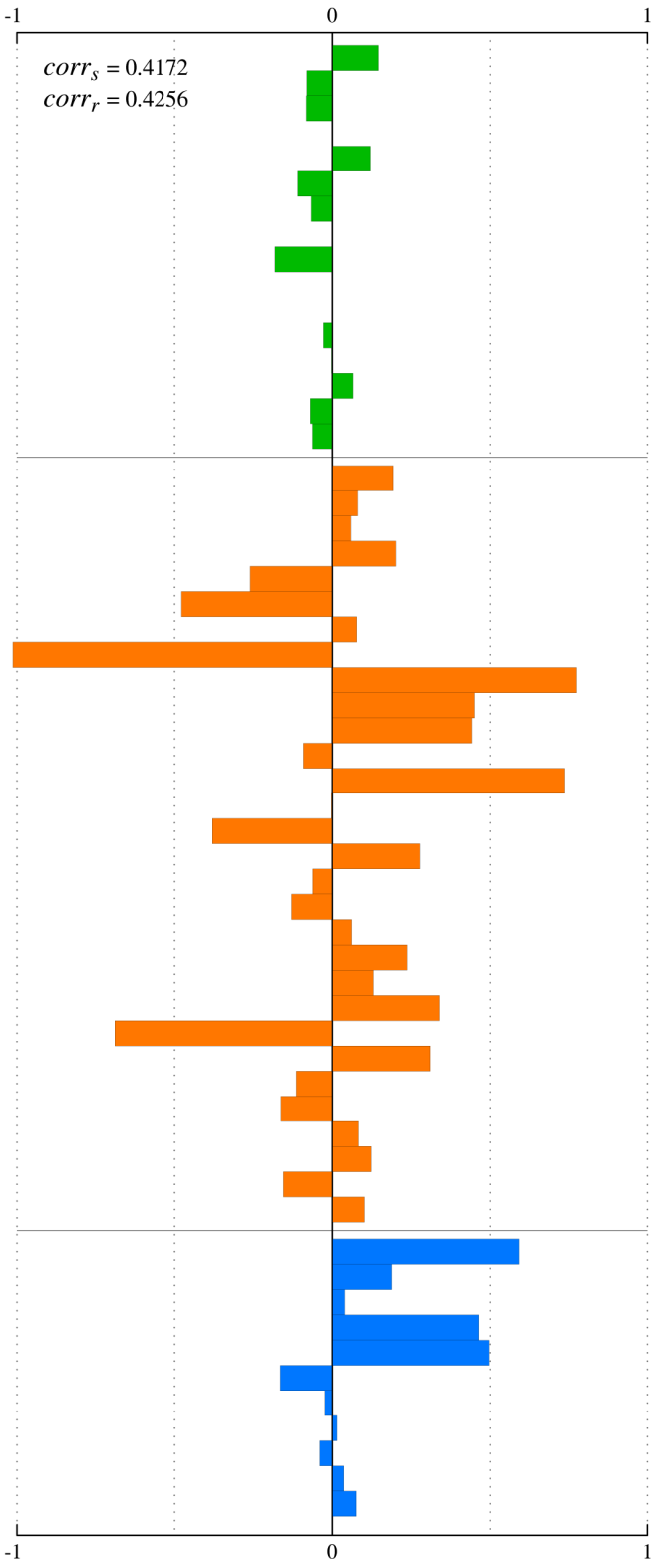


Figure G.2 Correlation weightings of all 57 aesthetic criteria.

Intrinsic Criteria

There are 16 intrinsic criteria.

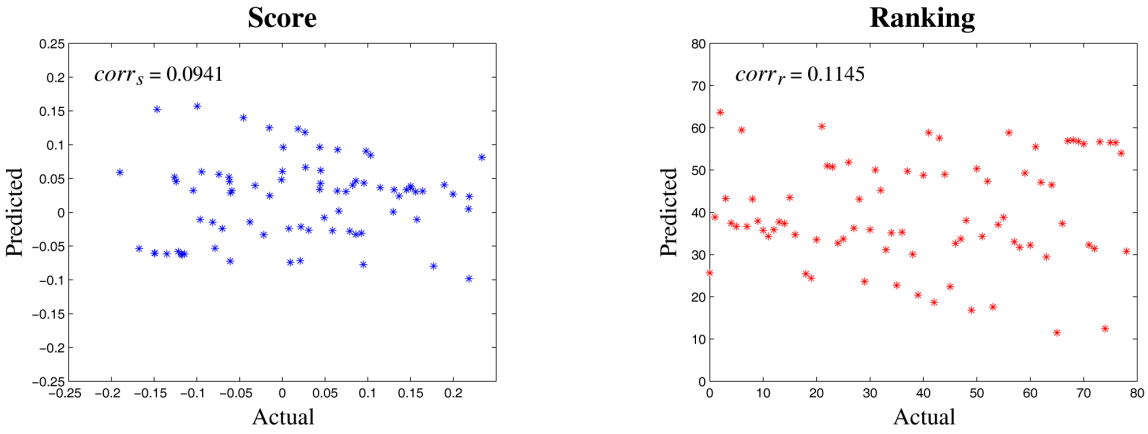


Figure G.3 Correlation of the 16 intrinsic criteria with game scores and rankings.

The game score correlation $corr_s = 0.0941$ is significantly worse for the 16 intrinsic criteria, in fact hardly better than random. The 95% confidence interval for this correlation is 0.984 [-0.421..0.563].

Criteria	Weight
0. Complexity	0.0505
1. Goal (connect)	0.0093
2. Goal (capture)	-0.1140
3. Goal (N-in-a-row)	-0.0000
4. Goal (race)	-0.1025
5. Goal (group)	-0.0233
6. Goal (stack)	-0.0468
7. Goal (score)	-0.0000
8. Goal (block)	0.0306
9. Piece addition	0.0125
10. Piece removal	0.0000
11. Piece movement	0.0263
12. Piece stacking	-0.0359
13. Piece capture	0.0391
14. Piece conversion	0.0924
15. Piece promotion	0.0610
Bias	0.0218

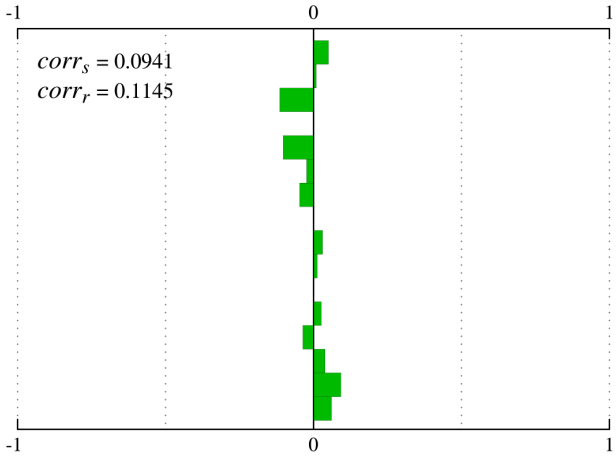


Figure G.4 Correlation weightings of the 16 intrinsic criteria.

Conclusion: The 16 intrinsic criteria, in isolation, are poor predictors of game preference score.

Quality Criteria

There are 30 quality criteria.

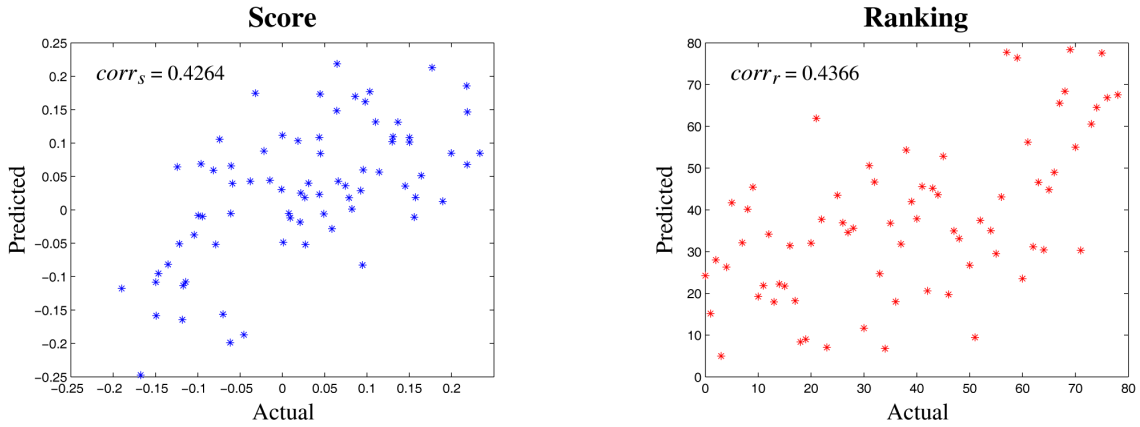


Figure G.5 Correlation of the 30 quality criteria with game scores and rankings.

The game score correlation $corr_s = 0.4264$ for the 30 intrinsic criteria is similar to that found over all criteria. The 95% confidence interval for this correlation is 0.602 [0.079..0.681].

Criteria	Weight
16. Branching factor	0.0014
17. Convergence	0.0118
18. Response time	0.1766
19. Move effort	0.0824
20. Search penetration	-0.1499
21. Clarity (variance)	-0.5178
22. Clarity (narrowness)	0.0998
23. Uncertainty (overall)	0.3848
24. Uncertainty (early)	-0.1174
25. Uncertainty (late)	-0.2782
26. Drama (average)	0.3975
27. Drama (maximum)	-0.2036
28. Killer moves	0.2784
29. Permanence	0.0042
30. Lead change	-0.1693
31. Stability	0.1408
32. Decisiveness threshold	0.3271
33. Decisiveness	-0.0484
34. Depth (discrepancy)	0.0924
35. Foulup factor	0.2121
36. Momentum (1)	0.0983
37. Momentum (2)	0.4359
38. Momentum (3)	-0.5564
39. Correction (1)	0.5288
40. Correction (2)	-0.0983
41. Correction (3)	-0.1911
42. Control	0.0192
43. Coolness	-0.0051
44. Board coverage	-0.0568
45. Puzzle quality	0.0664
Bias	-0.2258

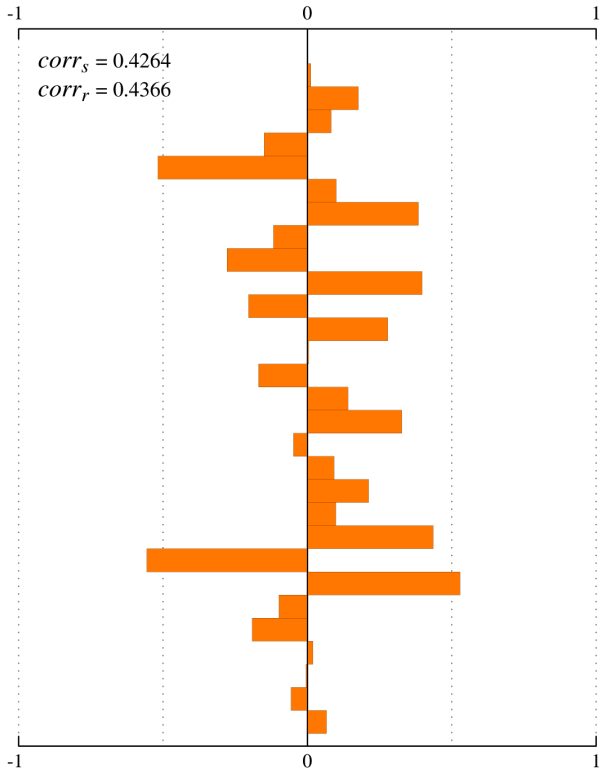


Figure G.6 Correlation weightings of the 30 quality criteria.

Conclusion: The 30 quality criteria, in isolation, perform to some degree as predictors of game preference score.

Viability Criteria

There are 11 viability criteria.

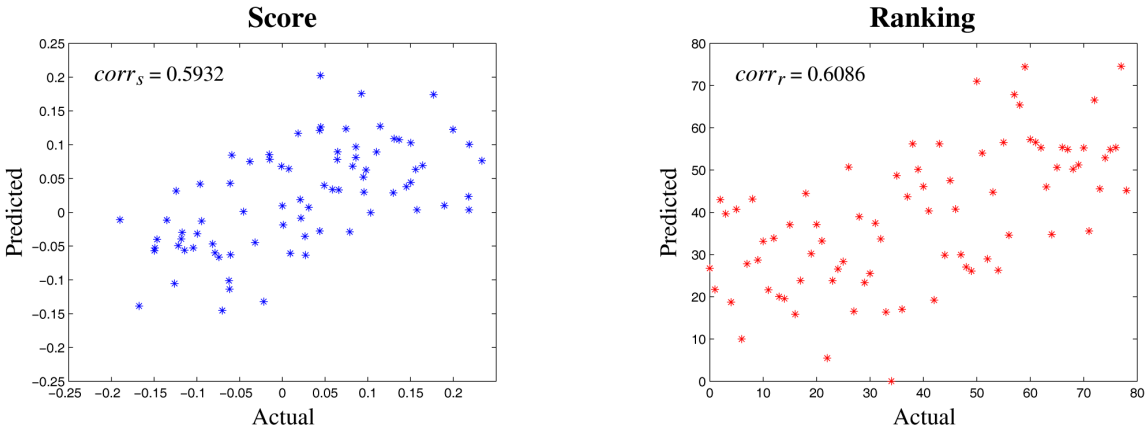


Figure G.7 Correlation of the 11 viability criteria with game scores and rankings.

The game score correlation $corr_s = 0.5932$ is significantly better for the 11 viability criteria than for the previously considered predictor sets (including the set of all criteria). The 95% confidence interval for this correlation is 0.880 [-0.010..0.879].

Criteria	Weight
46. Completion	0.6715
47. Balance	0.0856
48. First move advantage	-0.0367
49. Drawishness	0.6002
50. Timeouts	0.7155
51. Duration	-0.2448
52. Depth (skill level)	0.1010
53. Selfishness	-0.0053
54. Obstructive play	-0.0153
55. Resilience	0.0054
56. Puzzle potential	0.0642
Bias	-0.6466

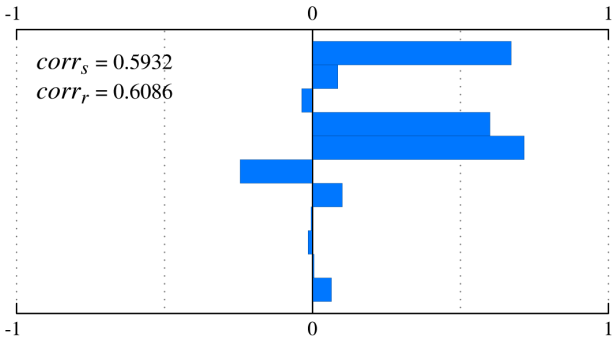


Figure G.8 Correlation weightings of the 11 viability criteria.

Conclusion: The 11 viability criteria, in isolation, are reasonably good predictors of game preference score.

Best 17 Predictor Set

A particular combination of 17 criteria over all categories was found to produce the best results.

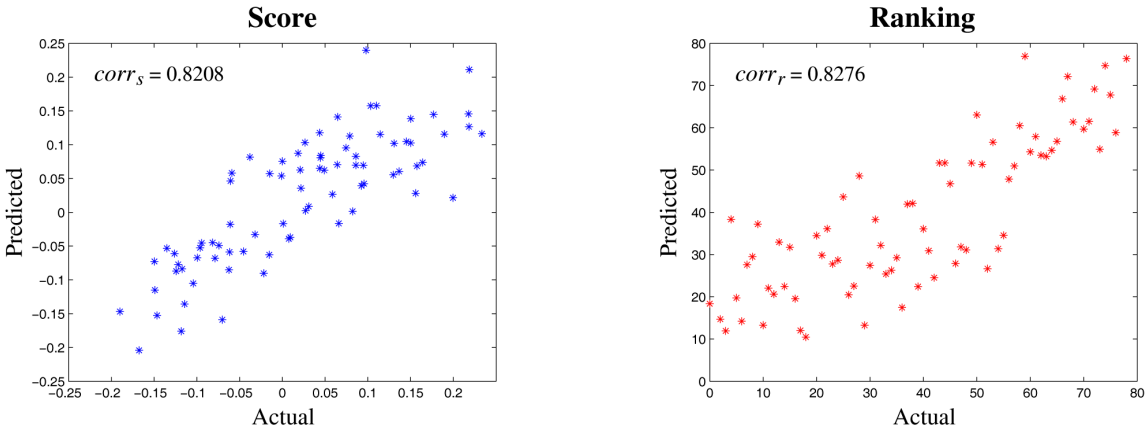


Figure G.9 Correlation of the best 17 aesthetic criteria with game scores and rankings.

This best 17 predictor set produces a stronger game score correlation of $corr_s$ = 0.8208 with a 95% confidence interval of 0.371 [0.562..0.933].

Criteria	Weight
5. Goal (group)	0.0818
6. Goal (stack)	-0.0804
8. Goal (block)	-0.1055
13. Piece capture	0.0585
17. Convergence	-0.0221
21. Clarity (variance)	-0.3763
25. Uncertainty (late)	0.2023
26. Drama (average)	0.2167
28. Killer moves	0.3585
29. Permanence	0.1027
30. Lead change	-0.2769
32. Decisiveness threshold	0.1311
36. Momentum (1)	0.1244
39. Correction (1)	0.1622
45. Puzzle quality	0.0826
46. Completion	0.0788
51. Duration	-0.0907
Bias	-0.2576

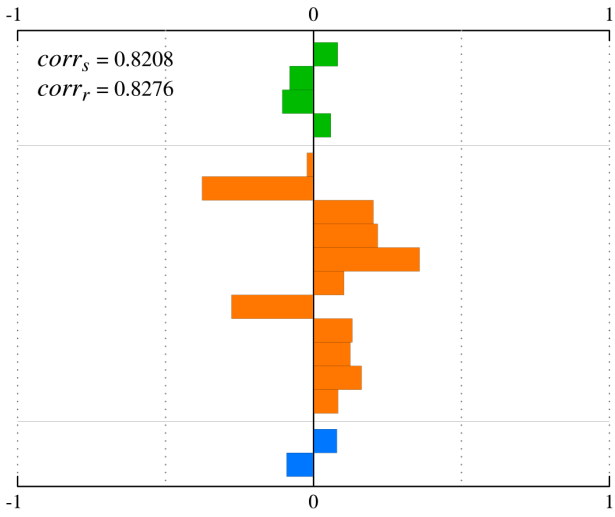


Figure G.10 Correlation weightings of the best 17 aesthetic criteria.

Conclusion: The best 17 criteria are good predictors of game preference score.

Appendix H

Viable Evolved Games

This section contains the complete ludeme descriptions of the 19 viable games evolved in Experiment III. The descriptions have been reformatted for readability and some notations added.

The games are listed in order of human player preference as determined in Experiment III.

Game #1: Ndengrod

```
(ludeme Ndengrod
   (players White Black)
   (board
      (tiling hex)
      (shape trapezium)
      (size 7 7)
   )
   (pieces
      (Piece All
         (moves
            (move
               (pre (empty to) )
               (action (push) )
               (post (capture surround) )
            )
         )
      )
   )
   (end (All win (in-a-row 5) ) )
   (ancestry
      (parents 035 Amloth)
      (maturity 2)
   )
   (advisors
      (nina 20)
      (linecap 2)
      (mater 4)
      (mobil 1)
      (infl 1)
   )
)
```

Game #2: Yavalath

```
(ludeme Yavalath
   (players White Black)
   (board
      (tiling hex)
```

```
    (shape hex)
    (size 5)
  )
  (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
  (end
    (All win (in-a-row 4) )
    (All lose (and (in-a-row 3) (not (in-a-row 4) ) ) )
  )
  (ancestry
    (parents Galathal 001)
    (maturity 7)
  )
  (advisors
    (nina 10)
    (linecap 2)
    (infl 1)
  )
)
```

Game #3: Rhunthil

This game contains a badly formed movement element in the Stone piece definition. This piece's *state* element was presumably mistaken for a *moves* element and another game's *moves* element incorrectly crossed over to it.

This badly formed element contains a more serious bug; an empty *action* element that could have undefined consequences if triggered during actual play. However an *action* clause is not expected in a *state* element so this potential problem will never be triggered.

```
(ludeme Rhunthil
  (players White Black)
  (board
    (tiling square i-nbors)
    (shape square)
    (size 8 8)
  )
  (pieces
    (StoneA All
      (label A
        (move
          (pre (empty to) )
          (action (push) )
          (post (capture surround) )
        )
        (move
          (action (push) )
          (post (capture surround) )
        )
      )
      (moves
        (move
          (pre (and (empty to) (= (phase to) 1) ) )
```

```
            (action (push) )
        )
        (move
            (pre (and (Piece-state from) (connected) ) )
            (action (pop) (push) )
            (post (capture) )
        )
    )
)
(StoneB All
    (label B)
    (moves
        (move
            (pre
                (and
                    (occupied to)
                    (or
                        (= (phase to) 1)
                        (!= (phase to) 2)
                    )
                )
            )
            (action (push) )
        )
    )
)
(Stone All
    (state 1                    // Bug: move element crossed over to a state element.
        (move
            (pre (empty to) )
            (action)            // Bug: Empty (action) element.
            (post (capture surround) )
        )
    )
    (moves
        (move
            (pre (empty to) )
            (action (push) )
        )
        (move
            (action (pop) )
            (post (inc-state) )
        )
    )
)
)
(start
    (in-hand (StoneB) 4)
    (in-hand (StoneA Black) 2)
)
(end (All win (in-a-row 5) ) )
(ancestry
    (parents 045 Yavani)
```

```
      (maturity 2)
    )
    (advisors
      (nina 16)
      (mater 6)
      (proxe 1)
      (stack -5)
      (mobil 1)
      (infl 1)
    )
  )
```

Game #4: Teiglith

```
  (ludeme Teiglith
    (players White Black)
    (board
      (tiling square)
      (shape square)
      (size 7)
    )
    (pieces
      (Stone All
        (moves
          (move
            (pre
              (and
                (> (group-size to) (phase to) )
                (connected)
              )
            )
            (action (pop) (push) )
          )
        )
      )
    )
    (start (place (Stone White) home) )
    (end (All win (no-move) ) )
    (ancestry
      (parents Hunenar Culumuri)
      (maturity 2)
    )
    (advisors
      (mater 2)
      (proxe 1)
      (stack -1)
      (mobil 1)
      (infl 1)
    )
  )
```

Game #5: Elrostir

```
(ludeme Elrostir
   (players White Black)
   (board
      (tiling square i-nbors)
      (shape square)
      (size 5)
   )
   (end (All lose (or (no-move) (in-a-row 3) ) ) )
   (ancestry
      (parents Gandil Yavarda)
      (maturity 15)
   )
   (advisors
      (group 1)
      (nina -10)
      (capt 10)
      (proxe 2)
      (proxm -1)
      (mobil 6)
      (infl 5)
   )
)
```

Game #6: Lammothm

```
(ludeme Lammothm
   (players White Black)
   (board
      (tiling square i-nbors)
      (shape square)
      (size 8 8)
      (regions (White n) (White s) (Black e) (Black w) )
   )
   (pieces
      (Stone All
         (moves
            (move
               (pre
                  (and
                     (empty to)
                     (or
                        (has-freedom)
                        (creates-freedom)
                     )
                  )
               )
               (action (push) )
               (post (capture surround) )
            )
         )
```

```
    )
  )
  (end (All win (connect own-regions) ) )
  (ancestry
    (parents 048 043)
    (maturity 1)
  )
  (advisors
    (conn 10)
    (nina 2)
    (mater 8)
    (infl 1)
  )
)
```

Game #7: Gorodrui

This game contains a *move*/*state* rule confusion similar to Rhunthil.

```
(ludeme Gorodrui
  (players White Black)
  (board
    (tiling hex)
    (shape hex)
    (size 3)
  )
  (pieces
    (Stone All
      (state 1            // Bug: moves element crossed over to with a state element.
        (move
          (pre (empty from) )
          (action (push) )
        )
      )
      (moves
        (move
          (pre (empty to) )
          (action (push) )
        )
        (move
          (pre
            (and
              (enemy from)
              (empty to)
              (= (+ (piece-state) 1) (distance) )
            )
          )
          (action (pop) (push) )
          (post (inc-state) )
        )
      )
    )
```

```
    )
    (start (in-hand (Stone All) 5) )
    (end (All lose (no-move) ) )
    (ancestry
      (parents 078 006)
      (maturity 1)
    )
    (advisors
      (mobil 1)
      (infl 1)
    )
  )
```

Game #8: Pelot

```
  (ludeme Pelot
    (players White Black)
    (board
      (tiling square)
      (shape square)
      (size 5)
    )
    (pieces
      (Stone All
        (moves
          (move
            (pre (empty to) )
            (action (push) )
          )
          (move
            (pre
              (and
                (owner from)
                (!= (piece-state) (distance) )
                (connected)
              )
            )
            (action (pop) (push) )
            (post (inc-state) )
          )
        )
      )
    )
    (end (All win (in-a-row 4) ) )
    (ancestry
      (parents Wilwari 019)
      (maturity 3)
    )
    (advisors
      (nina 10)
      (proxe 1)
      (stack -1)
```

```
      (mobil 1)
      (infl 1)
    )
  )
```

Game #9: Hale

```
  (ludeme Hale
    (players White Black)
    (board
      (tiling square)
      (shape square)
      (size 8 8)
    )
    (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
    (end (All win (in-a-row 4) ) )
    (ancestry
      (parents Glos Coimar)
      (maturity 8)
    )
    (advisors
      (nina 10)
      (proxe 1)
      (mobil 1)
      (infl 1)
    )
  )
```

Game #10: Quelon

```
  (ludeme Quelon
    (players White Black)
    (board
      (tiling square i-nbors)
      (shape square)
      (size 8 8)
    )
    (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
    (end (All lose (in-a-row 4) ) )
    (ancestry
      (parents Bar Elros)
      (maturity 9)
    )
    (advisors
      (nina -10)
      (proxe 2)
      (mobil 2)
      (infl 2)
    )
  )
```

Game #11: Duath

```
(ludeme Duath
   (players White Black)
   (board
      (tiling square i-nbors)
      (shape square)
      (size 6)
   )
   (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
   (end (All win (in-a-row 4) ) )
   (ancestry
      (parents Coimar 057)
      (maturity 2)
   )
   (advisors
      (nina 20)
      (linecap 3)
      (proxe 1)
      (proxs 1)
      (proxc 1)
      (infl 2)
   )
)
```

Game #12: Elrog

```
(ludeme Elrog
   (players White Black)
   (board phase
      (tiling square i-nbors)
      (shape square)
      (size 6)
   )
   (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
   (end (All win (in-a-row 4) ) )
   (ancestry
      (parents Radhross Urufin)
      (maturity 5)
   )
   (advisors
      (group 1)
      (nina 70)
      (linecap 3)
      (proxe 5)
      (proxs 1)
      (proxc 1)
      (infl 6)
   )
)
```

Game #13: Vairilth

```
(ludeme Vairilth
   (players White Black)
   (board
      (tiling hex)
      (shape rhombus)
      (size 7)
      (regions (White ne) (White sw) (Black se) (Black nw) )
   )
   (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
   (end (All win (and (no-move) (connect own-regions) ) ) )
   (ancestry
      (parents Gorodrui 034)
      (maturity 2)
   )
   (advisors
      (conn 10)
      (infl 1)
   )
)
```

Game #14: Ninniach

```
(ludeme Ninniach
   (players White Black)
   (board
      (tiling hex)
      (shape hex)
      (size 3)
   )
   (pieces
      (Stone All
         (moves
            (move
               (pre
                  (and
                     (= (phase to) (mover) )
                     (empty to)
                  )
               )
               (action (push) )
            )
            (move
               (pre
                  (and
                     (owner from)
                     (connected)
                     (<= (height from) (height to) )
                  )
               )
               (action (pop) (push) )
```

```
                (post (capture) )
            )
        )
    )
  )
  (end (All lose (no-move) ) )
  (ancestry
    (parents Ningwe Ningon)
    (maturity 3)
  )
  (advisors
    (mater 3)
    (proxe -3)
    (mobil 2)
    (infl 2)
  )
)
```

Game #15: Pelagonn

```
(ludeme Pelagonn
  (players White Black)
  (board
    (tiling trunc-square)
    (shape square)
    (size 8)
  )
  (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
  (end (All win (in-a-row 5) ) )
  (ancestry
    (parents Niphrend Gale)
    (maturity 8)
  )
  (advisors
    (nina 10)
    (proxe 1)
    (mobil 1)
    (infl 1)
  )
)
```

Game #16: Valion

```
(ludeme Valion
  (players White Black)
  (board
    (tiling square i-nbors)
    (shape square)
    (size 4)
  )
  (pieces
```

```
      (Stone All
        (moves
          (move
            (pre (>= (num-nbors to enemy) 1) )
            (action (push) )
          )
          (move
            (pre (and (empty to) (connected) ) )
            (action (push) )
          )
        )
      )
    )
    (start
      (place (Stone White) A1)
      (place (Stone Black) D4)
    )
    (end
      (All win (in-a-row 3) )
      (All lose (or (in-a-row 4) (group) ) )
    )
    (ancestry
      (parents Balronwe Kelvardg)
      (maturity 12)
    )
    (advisors
      (group 10)
      (nina 10)
      (mater 2)
      (proxe 1)
      (stack -1)
      (mobil 1)
      (infl 1)
    )
  )
```

Game #17: Eriannon

```
  (ludeme Eriannon
    (players White Black)
    (board
      (tiling square)
      (shape square)
      (size 7)
    )
    (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
    (end (All win (in-a-row 4) ) )
    (ancestry
      (parents 015 Rin)
      (maturity 5)
    )
    (advisors
```

```
      (nina 12)
      (linecap 3)
      (proxe 1)
      (infl 1)
    )
)
```

Game #18: Bregorme

```
(ludeme Bregorme
  (players White Black)
  (board
    (tiling square i-nbors)
    (shape square)
    (size 8 8)
    (regions (White n) (White s) (Black e) (Black w) )
  )
  (pieces (Piece All (moves (move (pre (empty to) ) (action (push) ) ) ) ) )
  (end (All win (connect own-regions) ) )
  (ancestry
    (parents Carnil 043)
    (maturity 3)
  )
  (advisors
    (conn 10)
    (nina 4)
    (infl 1)
  )
)
```

Game #19: Pelagund

```
(ludeme Pelagund
  (players White Black)
  (board
    (tiling trunc-square)
    (shape square)
    (size 8)
  )
  (pieces
    (Stone All
      (moves
        (move
          (pre (empty to) )
          (action (push) )
        )
        (move
          (pre (and (owner from) (connected) ) )
          (action (pop) (push) )
        )
      )
```

```
        )
    )
    (end
        (All win (in-a-row 6) )
        (All win (no-move) )
    )
    (ancestry
        (parents Gale Thaldar)
        (maturity 8)
    )
    (advisors
        (nina 10)
        (mater 5)
        (proxe -2)
        (stack -9)
        (mobil 1)
        (infl 1)
    )
)
```

Appendix I

Analysis of Yavalath

This appendix presents a brief analysis of the evolved game Yavalath. This includes a more complete restatement of the rules, some tips on general play, a description of a Yavalath computer player and an annotated sample game.

Rules for Yavalath

1. The board is initially empty.
2. Players take turns placing a piece of their colour at an empty cell.
3. Players win by making a line of four pieces of their colour.
4. Players lose by making a line of three pieces of their colour without also making a line of four.
5. The game is drawn if the board fills up before either player wins or loses.

Tactics and Strategy

The key tactical play in Yavalath is the *forcing move*, as shown in Figure I.1. White move **1** forces Black to lose with the blocking move **2**.

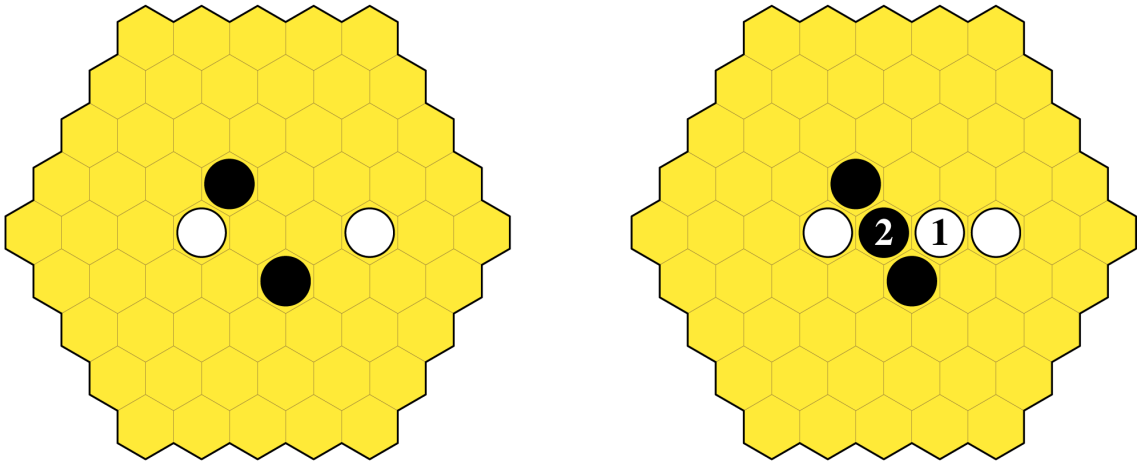


Figure I.1 White's forcing move **1** wins the game.

This example demonstrates two general principles of play:

1. Unblocked lines of size 4 (separated by two empty cells) have good attacking potential.
2. Unblocked lines of size 3 (or adjacent pairs of pieces) are vulnerable to attack.

Figure I.2 (left) shows the extension of principle 1 to a size 4 triangle formed by three unblocked lines of size 4. This is a strong formation that allows its owner to force a win, as shown on the right.

As a general rule of thumb, players should prevent the opponent from forming such unblocked size 4 triangles, and the first step in achieving this is to block lines of size 4 as they occur.

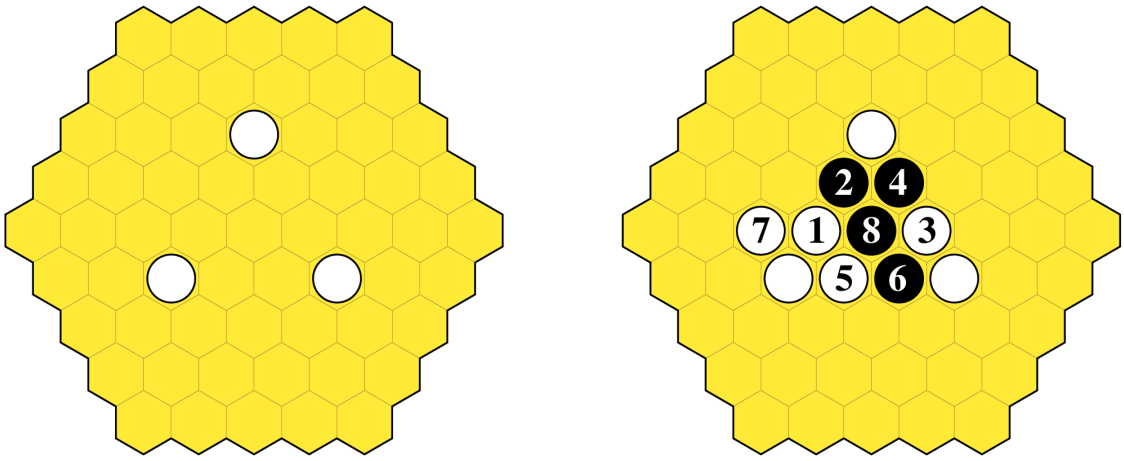


Figure I.2 An unblocked size 4 triangle (left) allows its owner to force a win (right).

Games of Yavalath tend to be short, decisive and dominated by passages of forced play. Moves that offer the current player freedom of choice are therefore precious and having the initiative (move in hand) is more important than in most games. It is easy to create interesting Yavalath puzzles from board positions that emerge during actual play.

Figure I.3 shows a blocking pattern that fills the board without either player winning or losing. However, this pattern requires more than twice as many pieces of one colour than the other, and hence will not occur in actual play except within subregions of the board.

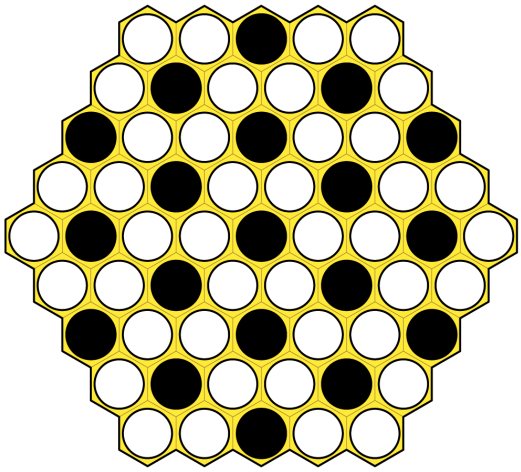


Figure I.3 Optimal blocking pattern.

Draws, although possible, are extremely rare. Players generally tend to make a fatal mistake (long sequences of forced moves can be difficult to predict correctly) or are forced to make a losing move as the number of available choices dwindles in the end game.

General UCT Algorithm

A dedicated Yavalath player was implemented to test the suitability of the UCT algorithm to playing new games in the absence of tactical and strategic knowledge. An initial implementation, based on the standard UCT algorithm with strictly random playouts, played a weak game that was easily beaten by human opponents and the Ludi general game player at a search depth of 3.

A more intelligent implementation, based on *general domain knowledge*, proved much stronger. General domain knowledge refers to information applicable to *all* games rather than any particular game, specifically:

- Always make a winning move (if possible).
- Never make a losing move (if possible).
- Block an opponent win (if possible).

This information may be readily incorporated into UCT playouts to give a General UCT (GUCT) method of move selection as follows:

```
if (winning move(s))
      choose one at random
else if (non-losing blocking move(s))
      choose one at random
   else if (non-losing move(s))
         choose one at random
   else
         choose any move at random  // must be loser
```

Listing I.1 Move selection in GUCT playouts.

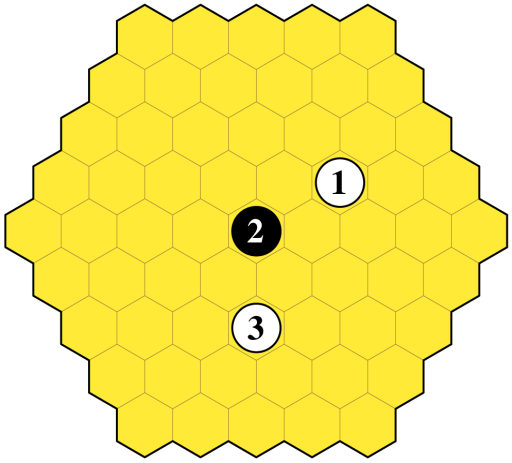
This simple innovation dramatically improved the quality of the player. With a response time of 15 seconds (approximately 45,000 playouts per move) the GUCT player consistently beat the Ludi general game player and most human opponents. It is difficult to gauge the exact strength of the GUCT player due to the lack of an established player base for Yavalath, however even the most experienced opponents found it difficult to beat when moving first and *extremely* difficult to beat when moving second. The player can be downloaded at [Browne, 2008].

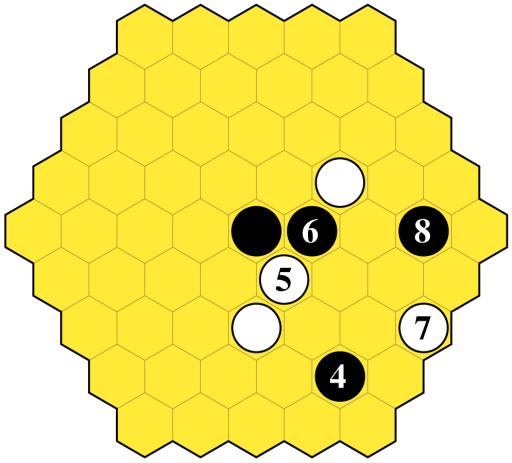
Another attractive property of the UCT approach is that it tends to find good solutions for a given board position but not necessarily the same "best" solution every time (unless that move is critical). The algorithm is hence non-deterministic and will follow different lines of play without the need for nudging it with added noise.

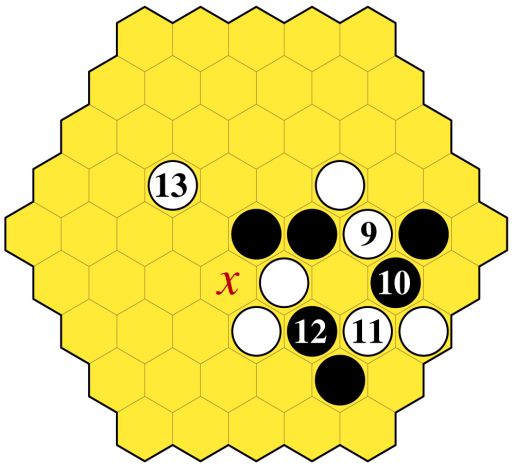
Perhaps the most ringing endorsement of the GUCT approach, however, is not the quality of its play so much as the fact that it can actually teach useful tactics and strategies for games of which it has no tactical or strategic knowledge. For example, the Yavalath player's preferred opening move (one or two cells removed from the centre) initially baffled human testers who had automatically assumed that the central cell would be the most powerful and the obvious choice for first move. However, some practice and analysis soon revealed the usefulness of this off-centre opening, as demonstrated in the following annotated game.

Annotated Game

The following Yavalath game #162 was played on the Gamerz.net server [Rognlie, 1996] in January, 2008, between the two highest-ranked players at the time.





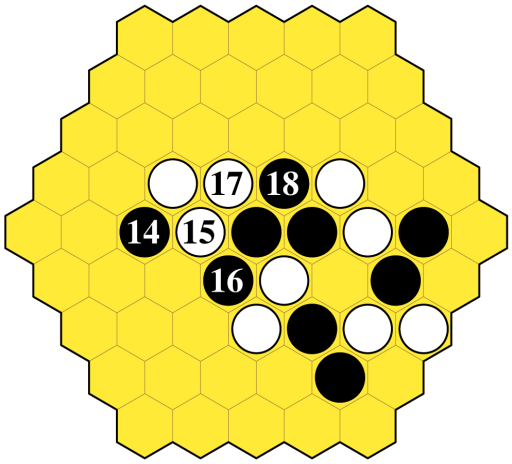


White	**Black**
1 Opening suggested by the GUCT player.	
	2? Black takes the proffered central cell.
3! Strong reply that establishes a size 4 line threat.	
	4 Black blocks with their own line threat.
5 Forcing move that blocks Black's line threat.	
	6 Forced reply.
7 White completes a size 4 triangle (blocked one side).	
	8? An apparently good move that only delays the inevitable.

Move **8** appears at first to be quite strong; it completes a size 4 triangle (partially blocked) while forcing an apparently innocuous reply **9** from White. However, the ensuing forced sequence eventually yields the initiative to White.

9 Forced reply.	
	10 Forced reply.
11 Forced reply.	
	12 Forced reply.
13!! Killer move.	

Move **13** establishes a size 4 triangle for White, which although blocked on one side presents a win next move at cell *x*. Black must immediately address this threat.



White	Black
	14 Black foils the immedaite win.
15 Forced reply.	
17 White lines up the win.	
	16 Forced reply.
	18 Black is forced to make a line of three.
	Black loses.

Black's defeat was certain after move **13** (no other reply **14** would avoid defeat) and almost as certain after move **8** since moves **9**, **10**, **11** and **12** were then forced. In fact, the decline of Black's position can be traced back to their very first move **2** which allowed White to establish a strong opening formation with move **3**. This opening play was first devised by the GUCT player, which puts it to good use against unwary opponents.

Bibliography

Abbott, R. (1975) "Under the Strategy Tree," *Games & Puzzles*, 36.

Abbott, R. (1988) "What's Wrong with Ultima," *World Game Review*, 8, 12-13.

Alden, S. (2000) "BoardGameGeek," http://www.boardgamegeek.com/.

Alexander, S. (2005) "Managing the game experience," *The Games Journal*, http://www.thegamesjournal.com/articles/ManagingTheExperience.shtml.

Allis, L., Van den Herik, H. & Herschberg, I. (1991) "Which Games Will Survive?" *Heuristic Programming in Artificial Intelligence 2: the second computer olympiad,* Levy, D. & Beal, D. (eds.), Ellis Horwood, Chichester, 232-243.

Allis, L. (1994) "Searching for Solutions in Games and Artificial Intelligence," PhD Thesis, University of Limburg, Maastricht.

Alpern, S. and Beck, A. (1991) "Hex Games and Twist Maps on the Annulus," *American Mathematical Monthly*, 98, 803-811.

Althöfer, I. (2003) "Computer-Aided Game Inventing," Technical Report, http://www.minet.uni-jena.de.preprints/althoefer_03/CAGI.pdf.

Althöfer, I. (2005) Personal correspondence.

Althöfer, I. & Heuser, S. (2005) "Randomised Evaluations in Single Agent Search," Technical Report, http://www.minet.uni-jena.de.preprints/althoefer_05/altheuser.pdf.

Angeline, P. & Pollack, J. (1992) "Learning With A Competitive Population," LAIR Technical Report #92-pa-compete, Ohio State University, Columbus.

Anshelevich, V. (2002) "A Hierarchical Approach to Computer Hex," *Artificial Intelligence*, 134, 101-120.

Arnheim, R. (1971) *Entropy and Art: An Essay on Disorder and Order*, University of California Press, Berkeley.

Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002) "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, 47, 235-236.

Bäck, T., Hoffmeister, F. & Schwefel, H. (1991) "A Survey of Evolution Strategies," *Genetic Algorithms*, Belew, R. & Booker, L. (eds.), Morgan Kaufman, San Mateo, 2-9.

Bäck, T. & Hoffmeister, F. & Schwefel, H. (1994) "Basic aspect of evolution strategies," *Statistics and Computing,* 4:2, 51-63.

Back, L. (2001) "Three Hex Variants," *Abstract Games*, 5, 24-25.

Baker, J. (1987) "Reducing bias and inefficiency in the selection algorithm," *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, Lawrence Erlbaum Associates, Mahwah, 14-21.

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982) *Winning Ways for your Mathematical Plays*, Volumes 1: *Games in General* and 2: *Games in Particular*, Academic Press, London.

Birkhoff, G. D. (1933) *Aesthetic Measure*, Harvard University Press, Cambridge.

Birmingham, J. & Kent, P. (1977) "Tree-Searching and Tree-Pruning," *Advances in Computer Chess 1*, Clarke, M. (ed.), Edinburgh University Press, Edinburgh, 89-97.

Bjork, S., Lundgren, S. & Holopainen, J. (2003) "Game Design Patterns," Game Developers Conference 2003, San Jose.

Böhm, N., Kókai, G. & Mandl, S. (2005) "An Evolutionary Approach to Tetris," *MIC2005: The Sixth Metaheuristics International Conference*, Vienna, 1-6.

Bordelon, P. (2004) "Further Notes on Game Design Chapter," Personal correspondence.

Borel, E. (1913) "Mécanique Statistique et Irréversibilité", *J. Phys. 5e série* 3, 189–196.

Borvo, A. (1977) *Anatomie d'un jeu de cartes: L'Aluette ou le Jeu de Vache*, Librarie Nantaise Yves Vachon, Nantes.

Bouzy, B. (2003) "On Meta-game Approaches to Computer Go," Technical Report, Universite Paris, Paris.

Browne, C. (2000) *Hex Strategy: Making the Right Connections*, A K Peters, Natick, Massachusetts.
Browne, C. (2005) *Connection Games: Variations on a Theme*, A K Peters, Natick, Massachusetts.
Browne, C. (2007) "Mambo", http://www.cameronius.com/games/mambo/.
Browne, C. (2008) "Yavalath", http://www.cameronius.com/games/yavalath/.
Cazenave, T. (1999) "Metaprogramming domain specific metaprograms," *Reflection 99*, LNCS 1616, 235-243.
Chellapilla, K. & Fogel, D. (1999) "Evolution, Neural Networks, Games, and Intelligence," *Proceedings of the IEEE*, 87:9, 1471-1496.
Coates, A. (2005) "Search space complexity," *AI Depot*, http://ai-depot.com/LogicGames/Go-Complexity.html.
Colton, S., Bundy, A. & Walsh, T. (2000) "On the Notion of Interestingness and Automated Mathematical Discovery," *International Journal of Human-Computer Studies*, 53:3, 351-375.
Conway, J. H. (1976) *On Numbers and Games*, Academic Press, London.
Costikyan, G. (2005) "Don't Be a Vidiot: What Computer Game Designers Can Learn From Non-Electronic Games", http://www.costik.com/vidiot.html.
Cron, D. H. (1999) "Directing Search in Metagame", Masters Thesis, Monash University, Melbourne.
Davey, J. (1934) *Art As Experience*, Perigree Books, New York.
Dawkins, R. (1976) *The Selfish Gene*, Oxford University Press, Oxford.
Dawkins, R. (1989) "The Evolution of Evolvability," *Artificial Life*, Langton, C. (ed.), Addison-Wesley, 210-220.
De Boer, P., Kroese, D., Mannor, S. & Rubinstein, R. (2004) "A Tutorial on the Cross-Entropy Method," *Annals of Operations Research*, 134:1, 19-67.
De Jong, K. & Spears, W. (1991) "An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms," *Parallel Problem Solving from Nature – Proceedings of 1st Workshop*, PPSN 1, Springer Verlag, Berlin, 38-47.
De Voogt, A. (1995) "A classification of board games," *New Approaches to Board Game Research: Asian Origins and Future Perspectives*, Van de Velde, P. (ed.), Working Paper Series 3, IIAS, Leiden, 9-15.
Drescher, M. (1961) *Games of Strategy: Theory and Applications*, Prentice-Hall, Englewood Cliffs.
Ellington, H., Addinall, E. & Percival, F. (1982) *A Handbook of Game Design*, Kogan Page, London.
Epstein, S. (1989) "The Intelligent Novice: Learning to Play Better," *Heuristic Programming in Artificial Intelligence: the First Computer Olympiad,* Levy, D. & Beal, D. (eds.), Ellis Horwood, Chichester, 273-284.
Epstein, S. (1991) "Deep Forks in Strategic Maps: Playing to Win," *Heuristic Programming in Artificial Intelligence 2: the Second Computer Olympiad,* Levy, D. & Beal, D. (eds.), Ellis Horwood, Chichester, 189-203.
Eschelman, L. & Schaffer, J. (1991) "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest," *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, 115-122.
Evans, R. (1974) "A Winning Opening in Reverse Hex," *Journal of Recreational Mathematics*, 7:3, 189-192.
Evans, R. (1975) "Some Variants of Hex," *Journal of Recreational Mathematics*, 8:2, 120-122.
Eysenck, H. (1968) "An Experimental Study of Aesthetic Preference for Polygonal Figures," *Journal of General Psychology*, 79, 3-17.
Franke, H. (1989) "Mathematics As an Artistic-Generative Principle," *Leonardo,* Computer Art in Context Supplemental Issue, 25-26.
Friedl, M. (2003) *Online Game Interactivity Theory*, Charles River Media, Hingham.

Fyfe, C. (1999) "Structured population-based incremental learning," *Soft Computing*, 2:4, 191-198.
Gale, D. (1979) "The Games of Hex and the Brouwer Fixed-Point Theorem," *The American Mathematical Monthly*, 86:10, 818-827.
Gao, X., Iida, H., Uiterwijk, W. & Van den Herik, J. (1999) "A Speculative Strategy," *Computers and Games: First International Conference (CG ,98)*, Van den Herik, J. (ed.), Springer, Berlin, 74-92.
Garcia, D. (1995) "GAMESMAN: A finite, two-person, perfect-information game generator", Masters Thesis, University of California, Berkeley.
Gardner, M. (1957) "Concerning the Game of Hex, which May be Played on the Tiles of the Bathroom Floor," *Scientific American*, 197:1, 145-150.
Gardner, M. (2004) "Theory of Everything," *The New Criterion*, 23:2.
Gelly, S. (2006) "Exploration exploitation in Go: UCT for Monte-Carlo Go", Video Lecture, http://videolectures.net/otee06_gelly_umc/.
Gelly, S., Wang, Y., Munos, R. & Teyaud, O. (2006) "Modification of UCT with Patterns in Monte-Carlo Go," Technical Report 6062, Institut National de Recherche en Informatique et en Automatique.
Genesereth, M. & Love, N. (2005) "General Game Playing: Game Description Language Specification," http://games.stanford.edu/gdl_spec.pdf.
Gering, R. (2005) Personal communication.
Ghory, I. (2004) "Reinforcement learning in board games," Technical Report CSTR-04-004, Department of Computer Science, University of Bristol, Bristol.
Gobet, F., de Voogt, A. & Retschitzki, J. (2004) *Moves in Mind: The Psychology of Board Games*, Psychology Press, Hove.
Gómez, J. A., Ragnarsson, C. J., and Seierstad, T. G. (2002) "Unlur: Winner of the Unequal Forces Game Design Competition," *Abstract Games*, 12, 17-21.
Gonzalez, R. & Woods, R. (2008) *Digital Image Processing*, Third Edition, Prentice Hall, Upper Saddle River.
Greene, J. (2007) "Downloads," http://members.cox.net/nppangband/download.htm.
Grimbergen, R. (2000) "Plausible Move Generation Using Move Merit Analysis with Cut-Off Thresholds in Shogi," *Computers and Games, Second International Conference, CG 2000*, Marsland, T. & Frank, I. (eds.), Hamamatsu, 315-332.
Grimbergen, R. & Matsubara, H. (2001) "Plausible Move Generation in Two-Player Complete Information Games Using Static Evaluation," *Transactions of the Japanese Society for Artificial Intelligence*, 16, 55-62.
Grunbaum, B. and Shephard, G. (1987) *Tilings and Patterns*, W. H. Freeman, San Francisco.
Gtkboard [2003] "Gtkboard: Board games system," http://gtkboard.sourceforge.net/indexold.html.
Guillion, D. & Guillion, O. (1996) "Awale", http://www.myriad-online.com/en/products/awale.htm.
Guy, R. (1991) "What is a Game?" *Proceedings of Symposia in Applied Mathematics*, 43, 1-21.
Hale-Evans, R. (2001) "Game Systems – Part I," *The Games Journal*, http://www.thegamesjournal.com/articles/GameSystems1.shtml.
Handscomb, K. (2000) Afterword to "An Introduction to Twixt," *Abstract Games*, 2, 12.
Handscomb, K. (2001) "Octagons: Another Perspective on this Unusual Connection Game," *Abstract Games*, 7, 12-13.
Hayward, R. & van Rijswijck, J. (2006) "Hex and Combinatorics," *Discrete Math*, 306, 2515-2528.
Henson, K. (1987) "Memetics: The Science of Information Viruses," *Whole Earth Review* #57, 50-55, http://findarticles.com/p/articles/mi_m1510/is_n57/ai_6203733.
Hesser, J. & Männer, R. (1991) "Towards an Optimal Mutation Probability for Genetic Algorithms," *Parallel Problem Solving from Nature – Proceedings of 1st Workshop*, PPSN 1, Springer Verlag, Berlin, 23-32.

Higashiuchi, Y. & Grimbergen, R. (2005) "Enhancing Search Efficiency by Using Move Categorization Based on Game Progress in Amazons," *Proceedings of the 11th Advances in Computer Games conference*, Taiwan, 97-103.
Ho, M. (1991) "Reanimating Nature: The Integration of Science with Human Experience," *Leonardo*, 24:5, 607-615.
Hofstadter, D. (1985) , *Mathematical Themas: Questing for the Essence of Mind and Pattern*, Penguin, Harmondsworth.
Holland, J. (1998) *Emergence: From Chaos to Order*, Addison-Wesley, Redwood City.
Hollosi, A. (2002) "XGF – An XML Game Format," http://www.red-bean.com/sgf/xml/.
Hollosi, A. (2006) "SGF File Format," http://www.red-bean.com/sgf/.
How, M. (1984) "Beyond Hex," *GAMES*, July, 57-58.
Howard, N. (1971) *Paradoxes of Rationality: Games, Metagames, and Political Behavior*, MIT Press, Cambridge.
Hughes, J., Michell, P. & Ramson, W. (1992) *The Australian Concise Oxford Dictionary*, Oxford University Press, Oxford.
Huizinga, J. (1955) *Homo Ludens: The Study of the Play Element in Culture*, Beacon Press, Boston.
Iida, H., Yosjimura, J., Morita, K. & Uiterwijk, W. (1999) "Retrograde Analysis of the KGK Endgame in Shogi: Its Implication for Ancient Heian Shogi," *Computers and Games: First International Conference (CG ,98)*, Van den Herik, J. (ed.), Springer, Berlin, 318-335.
Iida, H., Takeshita, N. & Yoshimura, J. (2003) "A Metric for Entertainment of Boardgames: Its Implication for Evolution of Chess Variants," *Entertainment Computing: Technology and Application, IFIP First International Workshop on Entertainment Computing (IWEC 2002)*, Nakatsu, R. & Hoshino, J. (eds.), Kluwer, Dordrecht, 65-72.
Iida, H., Takahara, K., Nagashima, J., Kajihara & Hashimoto, T. (2004) "An Application of Game-Refinement Theory to Mah Jong," *Entertainment Computing – ICEC 2004: Third International Conference (LNCS 3116)*, Rauterberg, M. (ed.), Springer, Berlin, 445-450.
Joris, W. (2002) *100 Strategic Games for Pen and Paper*, Carlton, Dubai.
Keller, M. (1983) "A Taxonomy of Games," *World Game Review*, 1, 21.
Kernighan, B. & Pike, R. (1999) *The Practice of Programming*, Addison-Wesley, Reading.
Kierulf, A. (1990) "Smart Game Board and Go Explorer: A study in software and knowledge engineering," *Communications of the ACM*, 33:2, 152-166.
Kocsis, L. & Szepesvári, C. (2006) "Bandit based Monte-Carlo Planning," *15th European Conference on Machine Learning (ECML)*, 282-293.
Korf, R. (1991) "Multi-player alpha-beta pruning," Artificial Intelligence, 48, 99-111.
Koster, R. (2004) "A Grammar of Gameplay," Game Developer's Conference 2005, http://www.theoryoffun.com/grammar/gdc2005.htm.
Koster, R. (2005) *A Theory of Fun*, Paraglyph, Scottsdale.
Koza, J. (1994) Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, 4, 87-112.
Kramer, W. (2000) "What Makes a Game Good?" *The Games Journal*, http://www.thegamesjournal.com/articles/WhatMakesaGame.shtml.
Lagarias, J. and Sleator, D. (1999) "Who Wins Misère Hex?" *The Mathemagician and Pied Puzzler: A Collection in Tribute to Martin Gardner*, Berlekamp, E. & Rodgers, T. (eds.), A K Peters, Natick, 237-240.
Langdon, W. (1996) "Data structures and genetic programming," *Advances in Genetic Programming 2*, MIT Press, Cambridge, 395-414.
Lehman, D. (1964) "A Solution of the Shannon Switching Game," *Journal of the Society for Industrial and Applied Mathematics*, 12, 687-725.
Levy, D. (1983) *Computer Gamesmanship: The Complete Guide to Creating and Structuring Games Programs*, Century, London.
Levy, L. (2002) "Strategy & Tactics," *The Games Journal*,

http://www.thegamesjournal.com/articles/StrategyTactics.shtml.
Loeb, D. (1996) "Stable winning coalitions," *Games of No Chance*, MSRA Publications, 29, Cambridge University Press, Cambridge, 451-471.
Lubberts, A. & Miikkulainen, R. (2001) "Co-Evolving a Go-Playing Neural Network," *Coevolution: Turning Adaptive Algorithms Upon Themselves*, Belew, R. & Juill, H. (eds.), San Francisco, 14-19.
Luckhardt, C. & Irani, K. (1986) "An Algorithmic Solution of *N*-Person Games," *Proceedings of AAAI-86, Volume 1: Science*, Morgan Kaufmann, Los Altos, 158-162.
Lundgren, S. (2002) "Joining Bits and Pieces," MSc Thesis in Interaction Design, Department of Computing Science, University of Goteberg, Goteberg.
MacDonald, D. & Fyfe, C. (2004) "Strategy Selection in Games Using Co-evolution Between Artificial Immune Systems," *Entertainment Computing – ICEC 2004: Third International Conference (LNCS 3116)*, Rauterberg, M. (ed.), Springer, Berlin, 445-450.
Maire, F. & Rasmussen, R. (2004) "An Accurate Induction Method of Player Ratings from Tournament Results," *Proceedings: Simulated Evolution And Learning (SEAL04)*, BEXCO, Busan.
Májek, P. & Iida, H. (2004) "Uncertainty of Game Outcome," *Proceedings of Inter Academia 2004*, Hungary, 171-180.
Mallett, J. & Lefler, M. (1998) "Zillions of Games: Unlimited Board Games & Puzzles," http://www.zillions-of-games.com/.
Markley, R. (2003) "A Game Rating System," *The Games Journal*, http://www.thegamesjournal.com/articles/RatingGames.shtml.
Markovitch, S. & Scott, P. (1988) "The Role of Forgetting in Learning," *Proceedings of the Fifth International Conference on Machine Learning*, Morgan Kaufmann, Ann Arbor.
Marsland, T. (1986) "A Review of Game-Tree Pruning," *Journal of the International Computer Chess Association*, 9:1, 3-19.
Matsubara, H., Iida, H. & Uiterwijk, J. (1996) "A Shogi-Computer Test Set," *CSC '96*, Philadelphia, 139-146.
McWorter, W. A. (1981) "Kriegspiel Hex," *Mathematics Magazine*, 54:2, 85-86.
Mendelson, E. (2004) *Introducing Game Theory and Its Applications*, Chapman & Hall, Boca Raton.
Meyers, S. (2001a) "New Connection Games: Part One," *GAMES*, October, 67.
Meyers, S. (2001b) "New Connection Games: Part Two," *GAMES*, November, 68, 77.
Miller, J. (2003) *Game Theory at Work: How to Use Game Theory to Outthink and Outmaneuver Your Competition*, McGraw-Hill, New York.
Müller, M. (1991) "The Smart Game Board as a Tool for Game Programmers," *Heuristic Programming in Artificial intelligence – The Second Computer Olympiad*, Levy, D. & Beal, D. (eds.), Ellis Horwood, Chichester.
Nahm, M. (1933) *The Aesthetic Response: an Antinomy and its Resolution*, University of Pennsylvania Press, Philadelphia.
Neto, J. (2000) "Mutators," *The Chess Variant Pages*, http://www.chessvariants.com/newideas.dir/mutators.html.
Neto, J. (2004) "The LUDAE Project," *LUDAE*, http://www.di.fc.ul.pt/~jpn/ludae/.
Neto, J. & Silva, J. (2005) *Jogos Matematicos, Jogos Abstractos*, Gradiva, Portugal.
Orwant, J. (2000) "EGGG: Automated programming for game generation," *IBM Systems Journal*, 39:3/4, 782-794.
Parlett, D. (1999) *The Oxford History of Board Games*, Oxford University Press, Oxford.
Parlett, D. (2007) "What is a ludeme?" http://www.davidparlett.co.uk/gamester/ludemes.html.
Patist, J. & Wiering, M. (2004) "Learning to Play Draughts Using Temporal Difference Learning with Neural Networks and Databases," *Benelearn '04: Proceedings of the Thirteenth Belgian-Dutch Conference on Machine Learning*, Nowe, A. & Steenhout, K. (eds.), 87-94.

Pell, B. (1992) “METAGAME in Symmetric Chess-Like Games,” *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*, Van den Kerik, H. & Allis, L. (eds.), Ellis Horwood, Chichester.
Pell, B. (1993a) “Logic Programming for General Game-Playing,” *Proceedings of the Workshop on Knowledge Computation and Speedup Learning: Machine Learning Conference*, Amherst.
Pell, B. (1993b) *Strategy Generation and Evaluation for Meta-Game Playing*, PhD Thesis, Trinity College, University of Cambridge, Cambridge.
Pell, B. (1996) “A Strategic Metagame Player for General Chess-Like Games,” *Computational Intelligence*, 12, 177-198.
Pitrat, J. (1968) “Realization of a general game-playing program,” *IFIP Congress*, 2, 1570–1574.
Polczynski, J. (2001) “Lightning: A Connection Game from the 1890s,” *Abstract Games*, 5, 8-9.
Pollack, J. & Blair, A. (1998) “Co-Evolution in the Successful Learning of Backgammon Strategy,” *Machine Learning*, 32, 225-240.
Raiko, T. (2006) “Higher Order Statistics in Play-out Analysis,” http://www.cis.hut.fi/praiko/papers/mlg07.pdf.
Reich, Y. (1993) “A model of aesthetic judgment in design,” *AI in Engineering*, 8:2, 141-153.
Reilly, R. (2008) Personal correspondence.
Rognlie, R. (1996) “Gamerz.net,” http://www.gamerz.net/pbmserv/.
Rolle, T. (2003) *Development of a Multi-Game Engine*, Diploma Thesis, Friedrich-Schiller-University Jena, Faculty of Mathematics and Computer Science, Jena.
Sackson, S. (1969) *A Gamut of Games*, Random House, New York.
Sadikov, A., Bratko, I. & Kononenko, I. (2005) “Bias and pathology in minimax search,” *Theoretical Computer Science*, 349, 268-281.
Salen, K. & Zimmerman, E. (2004) *Rules of Play*, MIT Press, Cambridge.
Samuel, A. (1959) “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, 3, 210-229.
Schensted, C. and Titus, C. (1975) *Mudcrack Y and Poly-Y*, NEO Press, Peaks Island.
Schmidt, G. (2007) “Axiom Universal Game Engine – Release 1.0”, http://www.gamesforum.ca/showthread.php?t=229341.
Schmittberger, R. W. (1983) “Star: A Game is Born,” *GAMES*, September, 51-54.
Schmittberger, R. W. (1992) *New Rules for Classic Games*, John Wiley & Sons, New York.
Schmittberger, R. W. (2000) “Making Connections,” *GAMES*, June, 10-13, 44, 58-61.
Schraudolph, N., Dayan, P. & Sejnowski, T. (2000) “Learning to Evaluate Go Positions Via Temporal Difference Methods,” *Computational Intelligence in Games*, Springer Verlag, Berlin, 74-96.
Shannon, C. E. (1950) “Programming a computer for playing Chess,” *Philosophical Magazine*, 41:4, 256-275.
Shannon, C. E. (1955) “Game playing machines,” *Journal of the Franklin Institute*, 206: December, 447-453.
Siegel, A. (2004) “Combinatorial Game Suite”, http://cgsuite.sourceforge.net/.
Sims, K. (1991) “Artificial Evolution for Computer Graphics,” *Computer Graphics*, 25, 319-328.
Smith, J. (1991) “Designing biomorphs with an interactive genetic algorithm,” *Genetic Algorithms*, Belew, R. & Booker, L. (eds.), Morgan Kaufman, San Mateo, 535-538.
Spears, W., De Jong, K., Back, T. Fogel, D. & De Garis, H. (1993) “An Overview of Evolutionary Computation,” *Machine Learning: EC ML-93*, Brazdil, P. (ed.), Springer Verlag, Berlin, 442-459.
Sperner, E. (1928) “Neuer Beweis fur die Invarianz der Dimensionszahl und des Gebietes,” Abhandlungen aus dem Mathematischen Seminar der Hamburgischen Universitat, 6:3/4, 265-272.
Staudek, T. (1999) “On Birkhoff’s Aesthetic Measure of Vases,” Technical Report FIMU-RS-99-06, Masaryk University, Masaryk.

Stiny, G. & Gips, J. (1978) *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts*, University of California Press, Berkeley.
Straffin, P. D. (1985) "Three Person Winner-Take-All Games with McCarthy's Revenge Rule," *The College Mathematics Journal*, 16:5, 386-394.
Sturtevant, N. & Korf, R. (2000) "On Pruning Techniques for Multi-Player Games," *AAAI/IAAI* , 2000:49, 201-207.
Sutton, R. (1988) "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, 3, 9-44.
Szita, I. & Lörincz, A. (2006) "Learning Tetris Using the Noisy Cross-Entropy Method," *Neural Computation*, 18:12, 2936-2941.
Tavener, S. (2007) Personal correspondence.
Tesauro, G. (1992) "Practical Issues in Temporal Difference Learning," *Machine Learning*, 8, 257-277.
Tesauro, G. (1995) "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, 38:3, 58-68.
Thompson, J. M. (2000) "Defining the Abstract," *The Games Journal*, http://www.thegamesjournal.com/articles/DefiningtheAbstract.shtml.
Thrun, S. (1995) "Learning to Play the Game of Chess," *Advances in Neural Information Systems 7*, Tesauro, G., Touretzky, D. & Leen, T. (eds.), MIT Press, Cambridge.
Todd, S. & Latham, W. (1994) *Evolutionary Art and Computers*, Academic Press, Orlando.
Van den Herik, H. (1983) "Strategy in chess endgames," *Computer Game-Playing: Theory and Practice*, Bramer, M., (ed.), Ellis Horwood, Chichester, 87-105.
Van Rijswijck, J. (2000a) "Are Bees Better Than Fruitflies? Experiments with a Hex Playing Program," *Advances in Artificial Intelligence: 13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2000*, Springer, Verlag, Berlin, 13-25.
Van Rijswijck, J. (2000b) *Computer Hex: Are Bees Better than Fruitflies?* Master's Thesis, University of Alberta, Alberta.
Van Rijswijck, J. (2002) "Search and evaluation in Hex," Technical Report, Department of Computer Science, University of Alberta, Alberta, http://www.cs.ualberta/~javhar/research/y-hex.pdf.
Van Rijswijck, J. (2004) "Hex," http://www.cs.ualberta.ca/~javhar/hex/.
Walker, S., Lister, R. & Downs, T. (1993) "On Self-Learning Patterns in the 'Othello' Board Game by the Method of Temporal Differences," *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence: AI '93*, Rowles, C., Liu, H. & Foo, N. (eds.), World Scientific, Singapore, 328-333.
Weisstein, E. (1999) *CRC Concise Encyclopedia of Mathematics*, CRC Press, Boca Raton, Florida.
Weisstein, E. (2005) "Connection Game," *Wolfram Mathworld*, http://mathworld.wolfram.com/ConnectionGame.html.
West, D. B. (1996) *Introduction to Graph Theory*, Prentice-Hall, Englewood Cliffs, New Jersey.
Wolfe, D. (1996) "The Gamesman's Toolkit," *Games of No Chance: Combinatorial Games at MSRI, 1994,* Cambridge University Press, Cambridge, 93-98.