An earlier version of this paper appears in: H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992.

METAGAME in Symmetric Chess-Like Games

Barney Pell[1]
University of Cambridge
Cambridge, UK
E-mail: bdp@cl.cam.ac.uk

Abstract

I have implemented a game generator that generates games from a wide but still restricted class. This class is general enough to include most aspects of many standard games, including Chess, Shogi, Chinese Chess, Checkers, Draughts, and many variants of Fairy Chess. The generator, implemented in *Prolog*, is transparent and publicly available, and generates games using probability distributions for parameters such as piece complexity, types of movement, board size, and locality.

The generator is illustrated by means of a new game it produced, which is then subjected to a simple strategic analysis. This form of analysis suggests that programs to play Metagame well will either learn or apply very general game-playing principles. But because the class is still restricted, it may be possible to develop a naive but fast program which can outplay more sophisticated opponents. Performance in a tournament between programs is the deciding criterion.

[1]Parts of this work have been supported by RIACS, NASA Ames Research Center [FIA], and a British Marshall Scholarship.

1 Introduction

As discussed in a companion paper ([Pel92]), the idea of Metagame is to develop programs which can take as input the rules of any game within a well-defined class, and play these games against opponents. Since the games are produced by a *game generator*, which has statistical components, the developers of the programs can no longer focus on a specific set of rules, and instead are forced to represent their knowledge to a program in a general fashion. The goal is that the *programs* will perform much of the interesting analysis of particular games, traditionally performed by humans.

Eventually, we would like to see programs which can analyse and play *any* games that humans could play. But for the moment, a smaller step is to generalise a set of games which have already been the subject of much research in computer game-playing, so that we can possibly transfer some of our current game-specific methods to a more general problem.

To this end, I have defined a class, *symmetric chess-like games*, which captures most aspects of many common games such as Chess, Checkers, Chinese Chess, and Shogi,[2] and represents these games in a manner which preserves much of their spatial structure. I have also implemented a *generator* for this class, based on a set of parameters such as piece complexity, the degree of crowding on the initial board (what fraction of the board begins with pieces), and the amount of decisions the players get to make throughout the game. Finally, I have developed a *move grammar*, which can be used by humans and programs to communicate moves for any game within this class.

This paper discusses the various components in detail. Section 2 discusses the class of *symmetric chess-like games*. Section 3 discusses the move grammar for this class. Section 4 discusses the game generator. Section 5 discusses a new game produced by the generator, and provides an analysis of this game by the author. Section 6 concludes the paper by discussing whether Metagame is beyond the state of the art.

2 Symmetric Chess-Like Games

Informally, a *symmetric chess-like game* is a two-player game of perfect information, in which the two players move pieces along specified directions, across rectangular boards. Different pieces have different powers of movement, capture, and promotion, and interact with other pieces based on ownership and piece type. Goals involve eliminating certain types of pieces, driving a player out of moves, or getting certain pieces to occupy specific squares. Most importantly, the games are *symmetric* between the two players, in that all the rules can be presented from the perspective of one player only, and the differences

[2] This paper mentions several games which may be unfamiliar. Descriptions of these games can be found in ([Bel69]).

in goals and movements are solely determined by the direction from which the different players view the board.

In what follows, we describe this class of games formally. For a full grammar in which games in this class are defined, see Appendix A.

At the highest level, a game consists of a *board*, a set of *piece definitions*, a method for determining an *initial setup*, and a set of *goals* or termination conditions. Each of these components is defined from the perspective of the *white* player, who initially places his pieces on the half of the board containing ranks with the lower numbers (call this *white's half* of the board). Unless they can move both forward and backward, white's pieces are generally forced to move toward black's side of the board, and vice-versa, which implies that there must be an inevitable point in the game when these opposing forces come into contact.

2.1 Definitions

Before we can describe the rules, we need a few definitions.

A *board* B is a two-dimensional rectangular array $[1\,..\,X_{max}, 1\,..\,Y_{max}]$, where X_{max} and Y_{max} are the number of *files* and *ranks*, respectively.[3]

Each element of B is a *square*, an ordered pair which is denoted by its position in this array: (x, y).

A *direction-vector* (*d-v*), $\langle dX, dY \rangle$ is a function which maps a square (X, Y) into a square $(X + dX, Y + dY)$. If $dY > 0$, this is a *forward d-v*, and if $dX > 0$, this is a *rightward d-v*.

A *directional symmetry* is a function which maps one *d-v* to another *d-v*. We define three special symmetries:

$$\begin{aligned} forward: &\ \langle dX, dY \rangle &&\mapsto \langle dX, -dY \rangle \\ side: &\ \langle dX, dY \rangle &&\mapsto \langle -dX, dY \rangle \\ rotate: &\ \langle dX, dY \rangle &&\mapsto \langle dY, dX \rangle \end{aligned}$$

A *symmetry set*, SS, is a subset of $\{forward, side, rotate\}$.

The *inversion*, $\mathcal{I}$, maps one square to another square, and maps one *d-v* to another *d-v*:

$$\begin{aligned} \mathcal{I}: &\ (x, y) &&\mapsto (X_{max} - x + 1, Y_{max} - y + 1) \\ &\ \langle x, y \rangle &&\mapsto \langle -x, -y \rangle \end{aligned}$$

Applying the inversion to a square or *d-v* thus produces the corresponding square or *d-v* from the perspective of the other player.

A *symmetric closure*, $SC(SS, D)$ of a *d-v* D under a symmetry set SS, is defined inductively as follows:

1. $D \in SC(SS, D)$

[3] Ranks and files correspond to rows and columns, respectively.

2. $S \in SS \wedge D_1 \in SC(SS, D) \longrightarrow S(D_1) \in SC(SS, D)$

3. Nothing else is in $SC(SS, D)$

Thus, a symmetric closure of a direction vector under a set of symmetries is the set closure obtained by applying these symmetries to the direction vector. If the symmetry set contained all three symmetries, then applying this set to direction $\langle 1, 2 \rangle$ would yield all eight possible directions of a knight move in Chess. Keeping only the side symmetry would yield the two possible moves of a Shogi knight ($\langle 1, 2 \rangle$ and $\langle -1, 2 \rangle$).

2.2 Global Symmetry

As *symmetric chess-like games* are totally symmetric, it is possible to present the entire set of rules (movements, capturing, initial setup, and goals) from the perspective of one player only, which can then be translated to the perspective of the other player by applying the *inversion* $\mathcal{I}$ to every square and *d-v* mentioned in these rules. In what follows, then, we define the rules only for the forward-moving player, illustrated by the *white* player in Chess.[4]

Thus, if the movement of a white piece of a certain type involves a direction vector D, then the corresponding movement of the black piece of that type will involve instead the vector $\mathcal{I}(D)$. Similarly, if white's goal is to have his knight arrive at a given square SQ, then black's goal will be to have his knight arrive at $\mathcal{I}(SQ)$. The same holds for promotion ranks, as will be discussed in the next section.

An interesting effect of this global symmetry is that it allows the generator to produce rules from the perspective of only the white (forward moving) player, and the global symmetry automatically implies that white pieces travel forward and to the right, and black pieces travel backward and to the left (from white's perspective), so that by default, opposing forces tend to move toward each other. If a piece movement has both a forward and a side symmetry, however, then the piece will travel along the same direction vectors for both players, because the inversion of a direction vector is precisely the same as the result of applying a forward and then a side symmetry to this vector.[5]

2.3 Board

The dimensions of a board are declared by the statement: `SIZE` X_{max}`BY` Y_{max}.

[4] Epstein ([Eps89]) uses the terms *player* and *opponent* to refer to the two players in a game. We shall here use these terms indexically, such that *player* refers to whichever player is to move.

[5] It is thus possible to characterise the degree of symmetry in a particular game by the extent to which movements are invariant under inversion. For example, Chess is almost totally symmetric (every piece except pawns has all three symmetries), while Shogi is much less so.

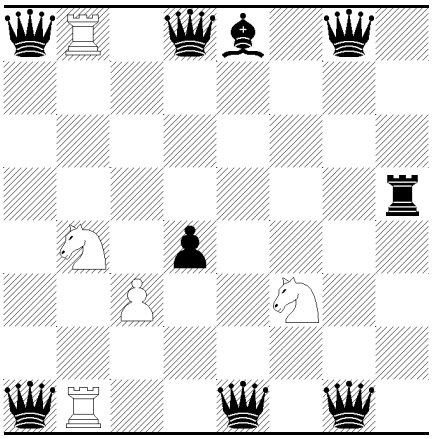

Figure 1: A *vertical-cylinder* board and capture movements

A board has one of two *types*: *planar* and *vertical-cylinder*. The planar board is the standard one used in almost all board games. The vertical-cylinder board is like the planar, but the left and right sides of the board are connected to each other so that pieces can *wrap-around* the side of the board.[6] Formally, for vertical-cylinders,

$$d\text{-}v\langle dx, dy\rangle : \quad (x, y) \quad \mapsto ((x + dx - 1) \bmod (X_{max}) + 1, y)$$

For example, in Figure 1, the ♘b4 can capture the ♜h5.

A board also has a privileged rank, called the *promotion rank*, such that any piece which, as a result of movement, arrives at or past this rank *at the end of a turn* must then exercise its *promotion power*, if it has one (see Section 2.4.4). If this rank had value 6 on a board consisting of 8 ranks, then white pieces would promote on reaching any rank numbered 6 or greater, while by inversion, black pieces would promote on reaching rank 3 or less ($Y_{max} + 1 - Y$). The set of squares at which a player can promote pieces is that player's *promotion territory*.

2.4 Pieces

A *piece* is defined by a power of *moving*, *capturing*, and *promoting*, and by an optional set of *constraints* on the use of these powers.

2.4.1 Movements

A *basic movement* consists of a *movement type*, which may have associated *movement restrictions*, a *direction vector*, D, and a *symmetry set*, SS. A piece with a given movement can move to any square reachable from its current

[6] The rules for vertical-cylinder boards are the same as for normal boards except for modular addition. It is thus legal for a piece to wrap around a vertical-cylinder board back to its original square, effectively passing the move to the next player.

square, according to its movement type, along D or any *d-v* in the symmetric closure $SC(SS, D)$.

Movement Types The simplest type of movement, called a *leap*, is that of taking a piece from a square S *directly* to the next square along a particular direction vector D, without regard for intervening squares. A piece which moves in this way is called a *leaper*. Thus, a movement which takes a piece only one square forward (for white) would be a $\langle 0, 1\rangle$-$leap$, which is the basic movement of pawns in Chess. Similarly, if a Chess knight were restricted to moving one square to the right, and two squares forward, this would be a $\langle 1, 2\rangle$-$leap$.

The next type of movement, called a *ride*, allows a piece to continue for some number of leaps along the same direction vector, as long as the squares on intermediate leaps are empty. So a pawn which is allowed to continue indefinitely forward through a line of empty squares (a *pawn rider*) would be a $\langle 0, 1\rangle$-$rider$. This is the basic movement of a *lance* in Shogi. This piece can be converted to a Chess *rook* by adding *rotation* and one of *side* and *forward* symmetries.[7]

The final type of movement, called a *hop*, is that in which we relax the constraint on a rider that intervening (leap) squares must be empty, and insist instead that some of these squares must be occupied by pieces.This type of movement is exemplified by the capturing power of a *man* in Draughts or a *cannon* in Chinese Chess.

Movement Restrictions Since a leap is a direct movement from an initial square to a final square, no other squares are considered. However, rides and hops pass through a set of intermediate squares, and additional restrictions may apply to those squares, as part of the rules for a particular piece's movement. For example, the *cannon* in Chinese Chess hops over any one piece *owned by either player*, with any number of empty squares before and after it, and captures the first enemy piece it lands on thereafter. However it is possible to restrict this piece further, by allowing hops over certain pieces only (e.g., black knights), constraining the number of empty squares before or after the hopped-over piece (called the *cannon-support*) to be within some interval (e.g., less than 3, at least 2), and requiring a piece to hop over a specified number of pieces matching a certain description (e.g., 2 pawns of either player).

To illustrate these restrictions, a constrained hopping movement might be defined as follows:

[7]Note that a $\langle 0, 2\rangle$-$rider$ could move from $(4, 1)$ to $(4, 5)$, as long as $(4, 3)$ was empty, without regard for squares $(4, 2)$ and $(4, 4)$, as each leap along the way moves directly to the second square forward.

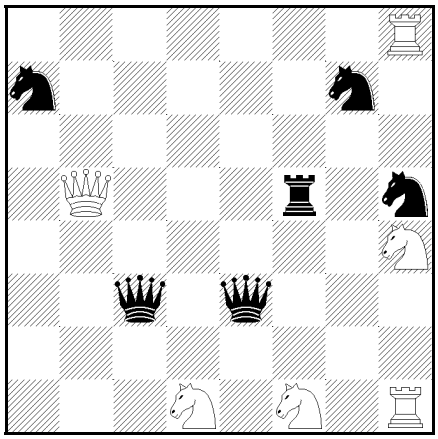

Figure 2: Example piece movements.

```
MOVEMENT
  HOP BEFORE [X >= 0]
      OVER   [X = 2]
      AFTER  [X <= 2]
  HOP_OVER [opponent any_piece]
  ⟨1,2⟩ SYMMETRY {side}
END MOVEMENT
```

A piece with this movement would move in one of the directions $\langle 1,2\rangle$ or $\langle -1,2\rangle$ (by side symmetry). In a given direction, the piece would first leap zero or more times, so long as each leap lands on an empty square. Then the piece would make two more leaps along the same direction, with the condition that each square be occupied by the opponent's pieces. Then the piece would make 0, 1 or 2 further leaps (still along the same direction), through empty squares. Finally the piece would make one last leap along the same direction, landing on its final square. If any of these conditions fail, the move is not legal. For example, in Figure 2, if ♘d1 were a piece with a capturing power whose movement was as a hopper so defined, it could move along direction $\langle 1,2\rangle$ to leap through 0 empty squares, then hop over the two enemy pieces ♛e3 and ♜f5, then leap through 0 empty squares, and then make a final leap to land on ♞g7. However, ♘d1 could not make a similar movement along direction $\langle -1,2\rangle$ to land on ♞a7, as it cannot hop over the friendly piece ♕b5.

In a similar manner, a ride can be restricted to *at least* L leaps and/or *at most* M leaps. A ride can also be restricted to the maximum number of available leaps (by the presence of the keyword `LONGEST` in the riding movement definition). For example, in Figure 2, if ♖h8 were a rook constrained to the longest ride in each direction for purposes of moving, it could only move to a8 or h6. This constraint thus limits a piece to one move in each legal direction. This restriction to *longest* ride is not applied on vertical-cylinder boards, as it is not well defined for this case.

Disjunctive Movements In addition to the basic movements, we also allow *disjunctive movements*, which are the union of several basic movements. Thus, if we have movements corresponding to a bishop and rook in Chess, then we can define the movement of a queen as the disjunction of these two simpler movements. This is also the method of definition of certain promoted pieces in Shogi.

2.4.2 Capturing

Capturing Movements The movements discussed above are used in defining both the *moving* and *capturing* powers of pieces. While a normal movement is used simply to move a piece from one square to another, with no effects on other pieces, a capturing movement always results in some change to the status of other pieces. It is possible (and even common) that pieces in chess-like games *move* in one way, and *capture* in another. Examples of this are pawns in Chess, cannons in Chinese Chess, and all pieces in Draughts.[8]

How To Capture In addition to special capturing movements, different pieces have different methods of capture. The most common capturing method, called *clobbering*, is when a piece ends its movement on a square occupied by another piece, and thus captures it. A second method, applicable only to pieces which *hop* as part of their capture movement, allows certain hopped-over pieces (see below) to be captured. The final capturing method is *retrieval*, in which a piece moves directly away from another piece, and thus captures it. A particular capture definition may allow different types of capture at once, so a piece might hop over one piece and land on another, capturing both.[9] Examples of these capturing methods are presented shortly.

What To Capture As in the case of restrictions on movements, there can be restrictions on what a piece can capture (e.g., any piece, opponent rook or queen), using a particular capturing power. This allows some pieces to be capable of capturing *anything but* a particular piece, for example. To be a legal use of a particular movement for a capturing power, at least one piece must actually wind up being captured.[10]

Effects of Capture Now, given that a piece can move in a certain way to capture a piece (or set of pieces), and that this piece is of the kind that it can legally capture, there are a number of possible *capture effects*, all of which remove the captured piece from its present square. The possible effects are:

[8] As in the case of movements, a piece can also have multiple (disjunctive) capture powers.

[9] A fourth common method of capture is *coordination*, in which some relationship between two or more pieces determines an additional set of pieces to be captured. Examples are bracketing in *Hasami Shogi* and *Othello*. This would be an interesting extension to the class presented here.

[10] Note that a piece can be restricted to capturing only friendly pieces.

- *Remove* a piece from the game altogether.
- *Player Possesses* the piece, converts it to his own side (if necessary) and can place it on any empty square, instead of making a normal piece movement, on one of his turns later in the game (i.e., starting with his next turn). This is the capture effect used in Shogi.[11]
- *Opponent Possesses* the piece, converts it to his own side (if necessary), and can place it on any empty square later in the game.

Here *player* and *opponent* are relative to whichever player has performed the capture. So if *white* captures a black piece, the *opponent possesses* effect means that *black* is then free to place this piece (still black) on any empty square later in the game, while the *player possesses* effect means that *white* would be able to place a white piece of that type later in the game. Examples of each type of capture effect, with the corresponding notation, are presented in Section 3.

Examples of Capturing We can illustrate the capture methods and restrictions using Figure 1. First, if pieces captured by clobbering, as in normal Chess, then ♖b1 could capture ♕a1 or ♕e1 by landing on them. Second, if pieces captured by hopping, then ♙c3 could capture ♟d4 by hopping over it to the empty square e5, and ♖b1 could capture ♕e1 by hopping over it to f1. Third, if pieces moved as in Chess but captured enemy pieces by *retrieval*, then ♖b1 could capture ♕a1 by moving away from it to c1 or d1, and ♘f3 could capture ♕g1 by moving directly away from it, for example to the square e5.

Finally, suppose ♖b8 is a piece with a *capture movement* of hopping on straight lines over any number and type of pieces, a *capture restriction* that it can capture only enemy ♕'s, and all three *capture methods* (i.e., clobbering, retrieval, and hopping capture). Then in one move, it could move directly away from ♕a8, hopping over ♕d8 and ♗e8, to land on ♕g8. All the enemy queens (♕a8, ♕d8, and ♕g8) would be captured and removed from the board, but ♗e8 would not be captured as this piece can only capture enemy ♕s.[12]

2.4.3 Compulsory and Continued Captures

The above sections describe the conditions and effects of capturing moves. In addition, there are two additional types of rules affecting the use of capture movements. The first type of rule *requires* a capture move to be made in preference to an ordinary piece movement. This is indicated by the presence of a

[11] Although Shogi restricts placement squares for some pieces (pawns cannot be placed on files where the player has a pawn already), the class defined here makes no such restrictions.

[12] Note that if there had been a ♗ at g8 instead of a ♕, this move would not have been legal, as a piece can never land on an occupied square unless it does so using a clobbering capture power which is restricted to pieces of a type consistent with the occupant.

must_capture constraint, which can appear as both a *global* and a *local* constraint (attached to the game definition or to the piece definition, respectively). As a *global* constraint, this indicates that if a player is to move *any* piece (as opposed to *placing* a piece from his hand), and *some piece* has a capture move available, he must play it. As a *local* constraint, this indicates that if a player is to move a *particular* piece, and *this piece* has a capture move available, he must play it. If the global version is present, any local versions are irrelevant. In both cases, when multiple such captures are available, the player is free to choose any one of them.

The second type of rule *allows* a player to make multiple capture movements within a single logical move, and is indicated as a continue_captures constraint. This occurs only in a local version, which allows multiple capture movements with the *same* piece. Unlike the game 10x10 Draughts, captured pieces are removed immediately, not at the end of a turn. Thus, a continued capture sequence is logically equivalent to one player making a sequence of capture movements with a particular piece, while the other player passes.

Finally, these two rules interact as follows: if at any point, both the must-capture and continue-captures rules are in effect, then the player *must* continue capturing if it is legal to do so. As the continue captures is only a local rule, only the piece which just captured is constrained to continue capturing.

The game of Checkers (Figure 3) illustrates the use of these rules. In this game, the must_capture rule is *global*, meaning that a player must make a capturing move if any of his pieces can capture, and the continue_captures rule is local to each piece, meaning that a player is allowed to continue capturing with a piece which has just made a capture movement. The interaction of these two rules means that a player must capture if he can, and once he has done so, he must continue capturing with the same piece until it cannot make any more captures.

2.4.4 Promotion

In addition to the normal moving and capturing powers attached to a piece, there is a special power, called *promotion*, which allows the piece to be changed while remaining on its final square. The rule applies when a player has *moved* a piece (possibly several times if this piece made a sequence of captures), which finishes its movements on a square which is in *promotion territory* for the player who moved it (see Section 2.3).[13]

In this case, one of the players (as specified in the definition of the piece) gets to replace the promoting piece with any piece of his choice matching a certain *description*. Thus, while in Draughts and Shogi pieces promote to a specific piece of the same colour, pawns in Chess promote to any of a set of pieces of the same colour, as chosen by *player*. Under the generalisation here,

[13]Note that moving from one square in promotion territory to another still qualifies a piece for promotion, and also that promotion applies only to a piece which actually used a moving or capturing power, as opposed to one which was placed on the board by one of the players.

this choice could also be made by *opponent*. If so, the opponent performs the promotion at the start of his next turn, before proceeding to make his ordinary placement or transfer move.

2.5 Initial Setup

Given a board and a set of pieces, it is necessary to determine a method for setting up an initial configuration of pieces. While some chess-like games begin with an arbitrary, fixed initial state, others have the players alternate assigning either their own piece, or their opponent's piece, to any of a set of squares.[14] A final possibility is that each contest of a particular game could begin with a randomised assignment of a known set of pieces to a known set of squares. Since these games are symmetric, both the fixed and random configurations are guaranteed to be symmetric. When players place their own pieces, however, there is no constraint that such placement be symmetric.[15] Finally, it should be noted that not all piece-types are necessarily present at the start of the game, as some can only be obtained through promotion (as in Checkers and Shogi).

2.6 Goals

So far we have described the method of determining the initial state, and the set of operators, which characterise this class of games. The final component necessary to describe any problem is the *goal*. As these games are symmetric, the goals, like the initial setup and piece movements, are defined from the perspective of the forward player. Thus, a goal definition simply defines those positions in which a player has achieved a goal, and we define a position as a *win* for *player* if only *player* has achieved a goal, a *draw* if *both* players have achieved a goal, and a *loss* if only *opponent* has achieved a goal. Goals are evaluated, from the perspectives of *both players*, at the *start* of each turn, when control is transferred from one player to another.[16] Thus either player might win at the start of each turn, if a goal is true from his perspective.

This class of games has three types of goals. First, a player achieves a *stalemate goal* in a position in which a specified player (player or opponent) cannot legally make any complete moves at the start of his own turn.[17] Thus,

[14] This corresponds to the clause for `assignment_decision` in the grammar in Appendix A. Note that `piece_names` may contain duplicates, as a player may have multiple pieces of the same type (e.g., players have two knights each in Chess).

[15] White places the first piece, and players alternate thereafter. During this phase there are no captures or promotions. Also, the initial squares upon which pieces are placed comprise the first R ranks for each player, so that the pieces always wind up assembled facing each other across the board.

[16] Note that under the *opponent-promotes* promotion method, a player begins a turn by promoting his opponent's piece (see Section 2.4.4), so goals are evaluated before he does this.

[17] Note again that promoting an opponent's piece does not constitute a complete move. In

white is stale-mated if he begins a turn with no legal moves, and a player is not stale-mated if it is the other player's turn to move. Every game in this class must have a defined stalemate goal, as the rules must cover positions in which a player cannot move. However each game decides whether such an outcome is a win, draw, or loss for the stale-mated player.

Second, a player achieves an *eradicate goal* if, at the start of any turn after the initial assignment stage, there are no pieces on the board which match a certain description. Examples are goals to eliminate the opponent king (Chess, Chinese Chess, Shogi), to eliminate all the opponent's pieces (Checkers), or to eliminate all your own pieces (Giveaway Chess). Note that a description might be complex, allowing goals to eliminate the opponent's knights and pawns. Note also that the description might be of the form: `[any_player king]`. Since *any_player* is symmetric for both players, this implies that both players achieve a goal if there are no more kings on the board. In other words, this outcome would be a draw.

Third, a player achieves an *arrival goal* if, at the start of a turn, a piece matching a certain description occupies a certain square on the board. This allows goals such as player getting his own knight to the square $(4,5)$, or player getting opponent's queen to the square $(2,2)$.[18]

2.6.1 Disjunctive Goals

An additional source of complexity in the rules of games in this class is that players can have disjunctive, or multiple, goals, in which a player achieves a goal if any of a number of conditions arise. For example, white may win the game if *either* someone eradicates black's knights, *or* white loses all his pieces. Such complex goals are especially interesting when the separate goals interfere with each other.[19]

2.6.2 General Termination

The goals of a game define the primary ways in which a game can end. However, it is possible that a game reaches a state in which neither player can (or knows how to) win. To stop such games from continuing forever, two additional rules are assumed for all games within this class. The first is an N-move rule, which says that the game automatically ends in a draw after some number of moves have been played. Since it is difficult to determine just how many moves any game in this class may require, the choice of N is rather arbitrary.

---

order to be legal the player must also be able to move or place a piece on the board.

[18] In the absence of certain compulsions, like *must capture* rules, this effectively means that a rational player will never move his piece to such a square, thus effectively adding a constraint instead of a goal to the game. However, it is certainly *legal* for a program to play such a move, thus losing the game instantly.

[19] It is difficult to imagine a naive evaluation function which could automatically handle these disjunctive goals.

For now we shall leave N at 500 moves (i.e., after black plays his 250th move, if neither player has won, the game is a draw).

A second rule is included to disallow endless cycles. Although this is not strictly necessary (since games terminate after N moves anyway), we adopt a rule similar to the *triple repetition rule* in Chess, which says that a game is a draw if the same position has been reached a third time with the same player to move. By *position*, we mean the contents of the board and *hands* of the players (i.e., they possess the same set of pieces).[20]

2.7 Coverage of Existing Games

Now that we have described the class of games in detail, we can discuss the general coverage of this class.

As is discussed more thoroughly in Section 4.1, the class of symmetric chess-like games was deliberately designed to be a generalisation which was restricted enough to preserve the structure of some real games, while general enough to allow complex interactions and a variety of games. This goal was assisted by drawing on research from the field of *Fairy Chess*, as developed by T.R. Dawson ([Dic71]). This field specialises in developing new variants and generalisations of Chess. Dawson's *Theory of Movements* formed the basis for the movement types (leap, ride, and hop) discussed in Section 2.4.1. This allows the class defined here to capture the basic forms of movement encountered not only in existing standard chess-like games, like Chess, Shogi, and Checkers, but also to handle many of the variants developed in Fairy Chess.

Although this allows most of the basic movements to be represented in this class, there are several aspects of common games which seemed too idiosyncratic to generalise. For example, it is difficult to find a natural generalisation of the *en passant* or *castling* rules in Chess, or of the rule in Shogi which prohibits a player from placing a pawn on a file on which he already has a pawn. Thus, these rules cannot easily be represented in the class defined here.

Another point about empirical coverage of this class is that players are allowed to make moves which would lose the game instantly, since piece movements are separate from goal criteria. For example, in Chess it is illegal to leave your king in check, and the game ends if a player can make no legal moves. In the class defined here, it is legal to move into check, but doing so would cause a loss of the game against any opposition.[21]

As an illustration of how chess-like games are defined in this class, Figure 3 presents a grammatical representation of the complete rules for American Checkers as a *symmetric chess-like game*.

[20] As no rules in games in this class make use of history, there is no need to discuss history in determining repetition of position, as is done in Chess.

[21] Thus the distinction between *checkmate* and *stalemate* in Chess cannot easily be fully represented.

```
GAME              american_checkers
GOALS             stalemate opponent
BOARD_SIZE        8 BY 8
BOARD_TYPE        planar
PROMOTE_RANK      8
SETUP             man AT {(1,1) (3,1) (5,1) (7,1) (2,2) (4,2)
                          (6,2) (8,2) (1,3) (3,3) (5,3) (7,3)}
CONSTRAINTS       must_capture

DEFINE man                                DEFINE king
  MOVING                                    MOVING
    MOVEMENT                                  MOVEMENT
      LEAP                                      LEAP
      ⟨1,1⟩ SYMMETRY {side}                     ⟨1,1⟩ SYMMETRY {forward side}
    END MOVEMENT                              END MOVEMENT
  END MOVING                                END MOVING
  CAPTURING                                 CAPTURING
    CAPTURE                                   CAPTURE
      BY {hop}                                  BY {hop}
      TYPE [{opponent} any_piece]               TYPE [{opponent} any_piece]
      EFFECT remove                             EFFECT remove
      MOVEMENT                                  MOVEMENT
        HOP BEFORE [X = 0]                        HOP BEFORE [X = 0]
            OVER   [X = 1]                            OVER   [X = 1]
            AFTER  [X = 0]                            AFTER  [X = 0]
        HOP_OVER [{opponent} any_piece]           HOP_OVER [{opponent} any_piece]
        ⟨1,1⟩ SYMMETRY {side}                     ⟨1,1⟩ SYMMETRY {forward side}
      END MOVEMENT                              END MOVEMENT
    END CAPTURE                               END CAPTURE
  END CAPTURING                             END CAPTURING
  PROMOTING                                 PROMOTING
    PROMOTE_TO king                           PROMOTE_TO king
  END PROMOTING                             END PROMOTING
  CONSTRAINTS continue_captures             CONSTRAINTS continue_captures
END DEFINE                                END DEFINE

END GAME.
```

Figure 3: Definition of *American Checkers* as a symmetric chess-like game.

3 Move Grammar

As discussed in a companion paper ([Pel92]) , for humans and programs actually to play Metagame there needs to be a language through which players can communicate their moves. The move grammar for *symmetric chess-like games* appears as Appendix B. The grammar is based on the standard notations for moves used in Chess and Shogi, but extended to describe unambiguously all the changes which can happen as part of a move in this class. As the move grammar should be clear, we will only provide a few example descriptions of different types of moves.

Basic Movements and Captures The basic movement of a piece P from square $(x1, y1)$ to square $(x2, y2)$ is written: `P` $(x1, y1) \rightarrow (x2, y2)$. If this move had the *capture effect* of *removing* a piece Q at square $(x3, y3)$, the full move would be (replacing symbols for squares): `P sq1` $\rightarrow$ `sq2 X Q sq3`.

Possession If the effect of a given capture were *player possesses* instead of removal, the captured piece Q would then be in the possession of the player who moved. If white had just moved, this would be denoted: `P sq1` $\rightarrow$ `sq2 X Q sq3 / (white)`. If the effect were instead *opponent possesses*, the captured piece would go to the opponent. The above move thus would be: `P sq1` $\rightarrow$ `sq2 X Q sq3 / (black)`.

Multiple Captures A single capture movement can result in the capture of several pieces. For example, a piece may hop over one piece to land on another, thus capturing both. Such multiple captures are denoted by listing each piece and square captured: `P sq1` $\rightarrow$ `sq2 X Q sq3 R sq4`. If the effect were *player possesses* instead of *remove*, such a move would be denoted: `P sq1` $\rightarrow$ `sq2 X Q sq3 R sq4 (white)`.

Placing a Possessed Piece A player in possession of a piece can at any later move *place* this piece on any empty square, instead of making a normal piece movement. So if the piece captured and possessed on $sq3$ in the last move above was Q, a later move for *white*, placing this piece on square sq, would be: `Q(white)` $\rightarrow$ `sq`.

Promotion by Player The notation for a move which promotes a piece includes the square the piece is on, the player who will now own the piece,[22] and the piece-type being promoted to. If white moves piece P to $sq2$ in his promotion territory, P has a `fixed` or `player_promotes` promotion power, and white decides to promote it to a king, this would be denoted: `P sq1` $\rightarrow$ `sq2; promote sq2 white king`.

[22] Recall that promotion may involve a change of ownership.

Promotion by Opponent When a player must begin his turn by promoting a piece which has just been moved by the opponent, the notation for this promotion precedes the notation for the rest of the move. For example, suppose a move by *white* moves a piece P from $sq1$ to $sq2$, capturing some piece Q on $sq3$, with the `opponent_possesses` capture effect, and also that $sq2$ is in promotion territory for white, and that piece P has the `opponent_promotes` promotion power. As this promotion is not denoted in the player's move (he makes no choice here, so it can be inferred), the notation for white's move would be: `P sq1 → sq2 X Q sq3 (black).`

White's turn would then end, and black would then have to promote white's moved piece into some piece $P2$ consistent with its promotion power, and then make a normal move (suppose he places the captured piece Q from his hand on square $sq4$). Black's move would then be denoted:

`promote sq2 P2; Q(black) → sq4.`

Continued Captures This sequential notation (with the semicolon) is also used when a player *continues capturing* (see section 2.4.3). For example, a continued captures move in Checkers might be written: `White Man (a 1) → (c 3) X (b 2) ; White Man (c 3) → (e 5) X (d 4).`

This notation is complete and unambigous, and thus allows humans and programs to communicate their moves in any game in the class of symmetric chess-like games.

4 Game Generator

The previous section discussed the grammar for this class of games, which shows what kind of games are possible. This section discusses the specifics of the generator, which actually produces new games from the set of those possible. Section 5 discusses a new game produced by the generator.

4.1 Generality vs. Structure

Defining a class of games for Metagameis almost as difficult as developing programs to play Metagame, as it involves making a tradeoff between *generality*, where we prefer classes which can describe the widest variety of games, and *structure*, where we prefer a class where the individual games appear to have similar underlying structure. On the one hand, very expressive class definitions (such as: those games definable in a particular programming language) require more intelligent game generators in order to produce any interesting games, as the lack of inherent structure decreases the chances that a given game within that class will be interesting, or even playable. On the other hand, very restricted generalisations (such as: variants of Chess where the pieces start in different locations) could probably be fully analysed by the hu-

man using extremely small generalisations of existing methods, thus defeating our goal that the programs should do more of the game analysis.

The compromise adopted here was to produce a class which is a moderate generalisation over a set of games which are actually played by people, but which still allows for enough interactions and complexities to generate a diverse and interesting set of games. We allow the components of the game-generator to be non-modular, in that they explicitly refer to generated aspects of other components, which means that the rules for the pieces can be highly complex and interactive. Yet, since the rules for these games are fully symmetric, we increase the chances that a given game is actually balanced, regardless of the complexities involved.[23]

4.2 Non-modularity

Non-modularity is achieved by generating a set of piece names first. Each piece name is just a symbol, to which the generator will attach a set of properties in order to define a particular piece. Each game component generator, like the movement generator, then has access to this entire set of piece names, from which it can then choose a subset in order to produce interactions with other pieces. For example, the movement generator might decide to generate a hopping movement. These movements have a restriction component, which describes the set of pieces over which a particular hopper can hop. The restriction component then chooses a subset from the set of piece names. Although the specific details of the pieces in this subset are inaccessible to the generator, the inclusion of the names alone is enough to develop an extremely complex pattern of interactions between the various pieces in a game.

4.3 Generator Parameters

Except for this element of non-modularity, the other components in a game are effectively generated by statistically choosing from tables of possibilities. Specifically, for each point in the grammar at which there is a nondeterministic choice point, there is an associated probability distribution indicating the probability of making one choice out of those possible.

For example, the clause in the grammar defining *movement types* is as follows:

```
movement_type --> leaper | rider | hopper
```

This states that a movement type can be either a leaper, a rider, or a hopper. The probability distribution corresponding to these possibilities is defined as follows:

[23]The issue of designing interesting games is discussed more generally in the preceding paper ([Pel92]).

```
parameter(movement_type,
    distribution([leaper=0.4, rider=0.4, hopper=0.2])).
```

This states the probability that the generator will choose each of these options, when they are available.

One interesting consequence of generating games in this fashion, by statistically sampling among possible rewrite rules, is that it provides a natural method of generating games with certain statistical properties, in that we can modify parameters corresponding to the probability of making different significant choices.

Rule Complexity One property of interest is the complexity (length) of the rules in a game. This can be controlled by means of a small set of parameters in the generator which are consulted in order to choose between making a game component more complex, or leaving it as it is.[24] This allows components to be generated with arbitrarily long descriptions, though longer descriptions are exponentially less probable than shorter ones. By varying these parameters, we can thus change the overall expected complexity of the components to which they are associated. Examples of such parameters are those attached to the `movement_def` and `capture_def` clauses, which control the probability of adding another disjunct to these definitions.

Decision Complexity Another statistical property of a game which can be determined in this way is the degree to which a game allows players to make choices, instead of assigning arbitrary values to these choices as part of the game definition. For example, pieces in Shogi promote to exactly one type of new piece each, whereas pawns in Chess promote to any one of a set of choices, to be decided by the player at the time of promotion. This property can easily be varied to produce different types of games, by modifying the distribution attached to the rule which decides, for example, whether a promotion or initial-setup decision should be arbitrary or not.

Search Complexity A related statistical property of generated games is that of search complexity, essentially the size of the search space in a particular game. This can be adjusted, without affecting the rule or decision complexities discussed above, by altering the probability distribution on *board size*, as larger board sizes will tend to allow more possible movements for each piece, and thus more possible moves in each position in the game. Of course, the parameters mentioned above (such as capture complexity) also affect the size of the search space, such as increasing increasing the probability that a piece has different types of movement available.

[24] More precisely, several rules choose between two possibilities, one of which is tail-recursive. Assigning a probability p to choosing the non-recursive case means that the recursion will continue with probability $1 - p$.

Certain parameters, in fact, have dramatic consequences on the search space. For example, the presence of an `opponent_promotes` rule, which allows the opponent to make a promotion decision before starting his move (see section 2.4.4), multiplies the number of possible moves available to him in a such a position: if a player had n ordinary move movements in a position, but he first has to promote an opponent's piece to one of p other pieces, then the total branching factor for that position is pn. If he had also to promote whichever piece he moved to one of M pieces, the branching factor would rise to Mpn.

At the opposite extreme of affecting search complexity, the presence of *must_capture* constraints has the effect of dramatically *reducing* the size of the search space.

Locality A final property of interest is *locality*, which determines the fraction of a board which can be traversed by a piece in one leap, without regard for the other squares on the board. The less locality, the more pieces on one side of the board can directly affect the status of pieces on another side of the board. It is possible that this affects the degree to which a program could reason about separate aspects of the board individually. Locality is affected by the modules constraining the restrictions on riders and hoppers, the module which generates direction vectors, and the `board_type` parameter, as a cylindrical board allows pieces to move from one side to the other with a direct leap.

4.4 Consistency Checking

Deciding whether a generated game can possibly be won generally requires a level of analysis beyond that implemented in the generator (in fact, the general problem is NP-Complete, as proved in [Pel93b]). However, the current generator does perform a simple analysis to avoid some of the common problems which would otherwise produce a high proportion of trivial games. For example, the generator does not produce goals to arrive a piece on a square where it will already be placed in the initial position.[25] In general, though, it is up to the programs to decide whether or not a game is trivial or even winable, which is indeed an aspect of game analysis traditionally left to humans.

5 A Worked Example

A recurrent point in work on Metagame has been that existing methods of computer game-playing have left much of the interesting game analysis to the human researcher, and that existing methods like minimax do not offer much advice on developing programs to play a new game. Thus, we developed a class of new games, and a generator for it, to highlight these issues and

[25] More details on this type of analysis are found in [Pel93b].

provide a test bed for addressing them. In this section we provide an example game actually produced by the generator, and a quick analysis of this game performed by the author. We then draw two conclusions from this example. First, although generated games often look silly at first, the complexity of the rules and symmetric structure offer chances for interesting strategic analysis. Second, the kind of game-analysis used to analyse these games is not easily amenable to a naive and general-purpose evaluation function.

5.1 *Turncoat-Chess*

I generated a random game using the generator with parameters set to prefer small boards and moderate complexity of movements, captures, and goals. The resulting game, as actually output from the generator, is presented in Figure 4 and Figure 5. I have replaced some internal symbols with more mnemonic names, and named this game *turncoat-chess*.

As the rules of this game are fairly complex, I shall attempt to summarise them in a more comprehensible form. For a full explanation of the meaning of particular rules, such as movement powers, see Section 2.

5.1.1 Summary of Rules

Turncoat-Chess is played on a 5-by-5 planar board. There are three types of pieces: *slug*, *termite*, and *firefly*. The initial setup is fixed, with pieces placed on the first rank of each player, symmetrically across the board. Each player starts with one slug, two termites, and two fireflies. Figure 6 shows a representation of the initial position for *turncoat-chess*. Fireflies are represented by the symbols ♘and ♞, termites by ♕and ♛, and slugs by ♖and ♜, for white and black pieces, respectively.

The pieces move and capture in different ways, discussed below, but all pieces can capture *any* type or colour of piece, by landing on it, and the captured piece is then permanently removed from the game. All pieces *promote* upon reaching the last rank, at which point the player who owns the piece can replace it with any type of piece, although for two of the pieces he must transfer ownership of it to the enemy after promoting.[26] A player wins if he has no legal moves at the start of his turn.

The descriptions of pieces are broken into powers of moving, capturing, and promoting.[27]

Slug The first type of piece is a *slug*. The slug moves by continually leaping (i.e., riding) to every second square along a particular rank or file, with the constraint that for each direction, it must ride as far as it can.[28] A slug's power

[26] Hence the name, *turncoat-chess*.

[27] It should be remembered that a capturing power can only be used if it results in a piece being captured.

[28] A way to think of this is that it *can't stop* riding along a line, until it is blocked.

```
GAME            turncoat_chess
GOALS           stalemate player
BOARD_SIZE      5 BY 5
BOARD_TYPE      planar
PROMOTE_RANK 5
SETUP           termite AT { (1,1) (2,1) }
                slug AT { (3,1) }
                firefly AT { (4,1) (5,1) }


DEFINE slug
  MOVING
    MOVEMENT
      RIDE LONGEST
      ⟨2,0⟩ SYMMETRY all_symmetry
    END MOVEMENT
  END MOVING
  CAPTURING
    CAPTURE
      BY {clobber}
      TYPE [any_player any_piece]
      EFFECT remove
      MOVEMENT
        HOP BEFORE [X >= 0]
            OVER   [X = 2]
            AFTER  [X >= 0]
        HOP_OVER [any_player {firefly}]
        ⟨0,1⟩ SYMMETRY {forward side}
      END MOVEMENT
    END CAPTURE
  END CAPTURING
  PROMOTING
    DECISION player
    OPTIONS [{player} any_piece]
  END PROMOTING
END DEFINE

DEFINE termite
  MOVING
    MOVEMENT
      HOP BEFORE [X >= 0]
            OVER   [X = 1]
            AFTER  [X >= 0]
      HOP_OVER [any_player {termite}]
      ⟨0,1⟩ SYMMETRY {side rotation}
    END MOVEMENT
    MOVEMENT
      RIDE LONGEST
      ⟨0,1⟩ SYMMETRY all_symmetry
    END MOVEMENT
  END MOVING
  CAPTURING
    CAPTURE
      BY {clobber}
      TYPE [any_player any_piece]
      EFFECT remove
      MOVEMENT
        LEAP
        ⟨2,3⟩ SYMMETRY {forward side}
      END MOVEMENT
    END CAPTURE
  END CAPTURING
  PROMOTING
    DECISION player
    OPTIONS [{opponent} any_piece]
  END PROMOTING
END DEFINE
```

Figure 4: *Turncoat-Chess*, a new game produced by the game generator.

```
DEFINE firefly
  MOVING
    MOVEMENT
      LEAP
      ⟨1,2⟩ SYMMETRY all_symmetry
    END MOVEMENT
    MOVEMENT
      HOP BEFORE [X >= 0]
           OVER   [X = 1]
           AFTER  [X >= 0]
      HOP_OVER [any_player {termite}]
      ⟨2,1⟩ SYMMETRY {side rotation}
    END MOVEMENT
    MOVEMENT
      LEAP
      ⟨2,3⟩ SYMMETRY all_symmetry
    END MOVEMENT
    MOVEMENT
      LEAP
      ⟨0,1⟩ SYMMETRY all_symmetry
    END MOVEMENT
  END MOVING
  CAPTURING
    CAPTURE
      BY {clobber}
      TYPE [any_player any_piece]
      EFFECT remove
      MOVEMENT
        LEAP
        ⟨0,1⟩ SYMMETRY all_symmetry
      END MOVEMENT
      MOVEMENT
        RIDE
        ⟨2,3⟩ SYMMETRY {forward side}
      END MOVEMENT
    END CAPTURE
  END CAPTURING
  PROMOTING
    DECISION player
    OPTIONS [{opponent} any_piece]
  END PROMOTING
END DEFINE
END GAME.
```

Figure 5: *Turncoat-Chess* (continued).

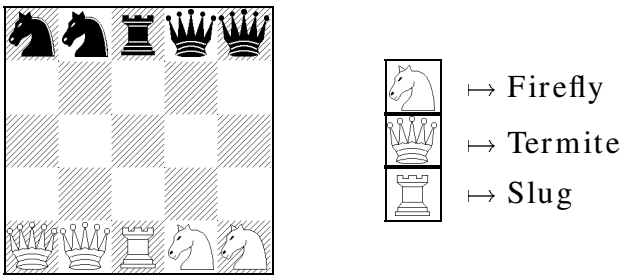

Figure 6: Initial board for *turncoat chess*.

to *capture* is very restricted: if there are two consecutive *fireflies* (of any colour) along a file, it can hop over any number of empty squares, then over the two fireflies, then over any number of empty squares, and finally capture *any* piece it lands on. So in Figure 2, if ♖h1 were a slug and ♘h4 and ♞h5 were fireflies, then ♖h1 could capture ♖h8, which is the first piece beyond the fireflies on the h-file. Finally, a slug can promote to any other piece, and does not change colour on promotion.

Termite The second type of piece is a *termite*, which moves in one of two ways. First, it can *hop* along a line forward, backward, or to either side, but must hop over a single *termite* of any colour, though it can pass over any number of empty squares before and after it. Second, it can move like a chess rook, in which case it makes the *longest ride* in a given direction until it is blocked. A termite captures any piece at relative coordinates $\langle 2,3\rangle$, forward and backward, left and right (but not $\langle 3,2\rangle$, which requires *rotational* symmetry). Finally, a termite promotes to any type of piece, though it then changes ownership (so a white termite promotes to any type of black piece).

Firefly The third type of piece is a *firefly*, which has many forms of movement and capture (see Figure 5). Its simple forms of movement are leaping as a chess knight, leaping 1 square orthogonally, or leaping to any square at relative coordinates $\langle 2,3\rangle$ or $\langle 3,2\rangle$, in any directions. Its more complicated form of movement is as a knight-hopper, in which case it must hop over a single termite. For example, in Figure 2, a firefly ♘f1 could hop over a termite ♛e3, and then land on either of the empty squares d5 and c7.

A firefly captures either by leaping to an orthogonally adjacent square, or leaping to a square at relative coordinates $\langle 2,3\rangle$, $\langle -2,3\rangle$, $\langle 2,-3\rangle$, or $\langle -2,-3\rangle$.[29] Finally, a firefly promotes the same way as a termite.

5.2 A Quick Analysis

As the rules look extremely complex, it can be difficult for a human to remember them, much less play a game using them. However, to illustrate the kind of simple analysis which is typical of humans analysing games, I will give an example of my own analysis of this game.

5.2.1 Strategy of Turncoat-Chess

Envisioning a Win In order to win, a player must begin a turn having no legal moves. Thus either he must have no remaining pieces, or they must have no moves. The first case seems easier to achieve. A player can remove his own

[29] According to the piece definition, it rides along these vectors, but on a 5 by 5 board there is enough room for only 1 leap.

pieces either by capturing them, or, in the case of fireflies and termites, by giving them to the opponent via promotion. As a player cannot give away a slug, he must either capture it with one of his own pieces, or first promote it into a termite or firefly, and then promote that piece to give it to the opponent. As the latter takes more moves, capturing a slug to start with seems the simplest option.

A Naive Winning Plan Thus, the simplest plan to win, ignoring opposition, is as follows: first, capture the rest of one's own pieces using one of the fireflies, then promote the final firefly, which will take at least 2 more moves.

Two Counter Plans However, this plan can easily be defeated with any opposition. First, it is not enough for a player to get rid of his last piece, as the opponent might be able to give him a piece back, and it is only stalemate if a player *begins* his turn without any moves. Second, while a player captures all his pieces with a firefly, the opponent can advance his own pieces to promotion range (after capturing his own slug first). Then when the first player has only 1 firefly left, the opponent can promote each piece to give away several slugs. Slugs are hard to promote, and have limited mobility, so the first player should be so busy trying to promote the slugs back to fireflies, that the opponent can capture or give his pieces away by promotion.

5.3 Discussion

So, this simple analysis reveals that it is at least possible to win this game, and there are a set of straightforward plans and counter plans which must be traded off. In the end, it is likely that one player will be overloaded with slugs, giving the other player time to win, but the means by which this happens are far from trivial. Thus, while the rules are strange and complex, the game could prove to be interesting, and certainly does present some elements of strategic complexity.[30]

While this analysis has only touched on the basic strategy of this new game, it does illustrate the kind of analysis humans perform when they are presented with a new game, and similar analyses for other games can be found in specialist books on these games, or in almost any paper on computer game-playing, where the human begins by analysing the game for significant features which could form the basis of an evaluation function. Thus far, though, this type of analysis has been considered a prerequisite for computer game-playing, and not a subject of research in its own right.

[30] When I first drafted this section, I generated a game (the first produced after a few system errors) and analysed it for 20 minutes, and had not yet played against an opponent. Thus, this strategy is very basic, and the reader will probably have thought of better strategies already.

6 Conclusion

Now that we have instantiated Metagame sufficiently to have developed a concrete problem that can actually be addressed, an important concern is whether this whole project is beyond the state of the art in games, learning, and even AI in general. It might be argued that researchers have tended to specialise on particular games, and leave the difficult aspects of game-analysis to humans, precisely because these problems are too hard to tackle at the present time.

While it is true that playing Metagame *well* raises some difficult issues, it is a straightforward process to build a program to play it *legally* (see [Pel93a]). Since the representation of any game produced by a generator is basically a definition of the legal moves of the game, a conversion program can be written which takes a new game and produces a legal move generator for it. From that point, we at least have an entrant in the competition, a *random* player, as a baseline.

Given that we can develop a legal move generator, it is then straightforward to create a program which plays based on some general heuristics. Examples of these might be using *expected outcome* ([Abr90]), *mobility* ([Don92, Har87]) or *material* as features in an evaluation function to conduct a minimax search (see [Pel93a]).

Thus it is easy to create some obvious baseline programs to play Metagame. From that point, any programs which actually do *anything* more clever are likely to defeat these simple programs. And this was precisely the motivation for Metagame: to change the problem such that once again, programs which address the interesting issues are expected to win more games. Whether they do or not in practice is an empirical question which can also be addressed in the context of competition, which in itself is an exciting prospect.

7 Acknowledgements

Thanks to Susan Epstein, Nick Flann, Mike Frank, Robert Levinson, Bert Menting, Douwe Osinga, Dan Pehoushek, Prasad Tadepalli, Mark Torrance, and David Wilkins for interesting discussions. Thanks especially to my supervisors, Steve Pulman and Manny Rayner, to Victor Allis for careful reading of drafts and for many useful suggestions on this class of games, and to my mentor-in-absentia, Peter Cheeseman.

References

[Abr90] Bruce Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2), February 1990.

[Bel69] R.C. Bell. *Board and Table Games from Many Civilizations*. Oxford University Press, 1969.

[Dic71] Anthony Dickins. *A Guide to Fairy Chess*. Dover, 1971.

[Don92] Ch. Donninger. The relation of mobility, strategy and the mean dead rabbit in chess. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992.

[Eps89] Susan Epstein. The Intelligent Novice - Learning to Play Better. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*. Ellis Horwood, 1989.

[Har87] D. Hartmann. How to Extract Relevant Knowledge from Grand Master Games, part 1. *ICCA-Journal*, 10(1), March 1987.

[Pel92] Barney Pell. Metagame: A New Challenge for Games and Learning. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992. Also appears as University of Cambridge Computer Laboratory Technical Report No. 276.

[Pel93a] Barney Pell. Metagame Realized: A Player to Beat. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 4 – The Fourth Computer Olympiad*. Ellis Horwood, 1993. In preparation.

[Pel93b] Barney Pell. *Strategy Generation and Evaluation for Meta Game-Playing*. PhD thesis, Computer Laboratory, University of Cambridge, 1993. Forthcoming.

A Class Definition

This grammar for *symmetric chess-like games* is presented in an extended-*BNF* notation. Capitalised and quoted words are terminal symbols, except for `IDENTIFIER` and `NUMBER`, which stand for any identifier and any number, respectively. Text in unquoted braces is optional. The grammar is not case-sensitive, so that game definitions may use uncapitalised words for clarity.

Comments can appear within game definitions. Comments begin with percent symbols (`'%'`) and end with the start of a new line.

```
game --> GAME IDENTIFIER
         goal_defs
         board
         SETUP assignment_list
```

```
          {CONSTRAINTS MUST_CAPTURE}
          piece_defs
          END GAME '.'

goal_defs --> GOALS goals

goals -->  goal | goal goals

goal --> ARRIVE description AT square_list
       | ERADICATE description
       | STALEMATE player

description --> '[' player_gen piece_names ']'

player_gen --> '{' player '}' | ANY_PLAYER

player -->  PLAYER | OPPONENT

piece_names --> '{' identifiers '}'
              | ANY_PIECE

identifiers --> IDENTIFIER | IDENTIFIER identifiers

square_list --> '{' squares '}'

squares --> square | square squares

square --> '(' NUMBER ',' NUMBER ')'

board --> BOARD_SIZE NUMBER BY NUMBER
          BOARD_TYPE board_type
          PROMOTE_RANK NUMBER

board_type --> PLANAR | VERTICAL_CYLINDER

assignment_list --> assignment_decision
                  | assignments

assignment_decision --> DECISION assigner ASSIGNS
                        piece_names TO square_list
                        END DECISION

assigner --> player | RANDOM

assignments --> assignment | assignment assignments
```

```
assignment --> IDENTIFIER  AT square_list

piece_defs --> piece_def | piece_def piece_defs

piece_def --> DEFINE IDENTIFIER
              MOVING movement_def END MOVING
              CAPTURING capture_def END CAPTURING
              PROMOTING promote_def END PROMOTING
              {CONSTRAINTS constraint_def}
              END DEFINE

movement_def --> movement | movement movement_def

movement --> MOVEMENT
             movement_type
             direction
             symmetries
             END MOVEMENT

movement_type --> leaper | rider | hopper

leaper --> LEAP

rider --> RIDE  {MIN NUMBER} {MAX NUMBER} {LONGEST}

hopper --> HOP BEFORE compare_eq
               OVER compare_eq
               AFTER compare_eq
           HOP_OVER description

compare_eq --> '[' X comparative NUMBER ']'

comparative --> '>=' | '=' | '<='

direction --> '<'  NUMBER ',' NUMBER '>'

symmetries --> SYMMETRY symmetry_set

symmetry_set --> ALL_SYMMETRY
               | '{' {FORWARD} {SIDE} {ROTATION} '}'

capture_def --> capture | capture capture_def

capture --> CAPTURE
```

```
            BY capture_methods
            TYPE description
            EFFECT effect
            movement_def
            END CAPTURE

capture_methods --> '{' {RETRIEVE} {CLOBBER} {HOP} '}'

effect --> REMOVE | player POSSESSES | player DISPLACES

promote_def --> PROMOTE_TO IDENTIFIER
              | promotion_decision

promotion_decision -->  DECISION player
                        OPTIONS description

constraint_def --> {MUST_CAPTURE} {CONTINUE_CAPTURES}
```

B Move Grammar

```
move --> {promote ';'}
         main_move '.'

promote --> PROMOTE square piece

main_move --> placement
            | transfers {';' promote}

placement --> piece '(' color ')' '->' square

transfers --> transfer ';' {transfers}

transfer --> moving {capture}

moving --> piece square '->' square

capture --> X piece square effect {capture}

effect --> remove
         | possess
```

```
remove --> {}

possess --> '/' '(' color ')'

piece --> color piece_name

color --> WHITE
        | BLACK

piece_name --> IDENTIFIER

square --> '(' NUMBER ',' NUMBER ')'
```