Monte Carlo Tree Search Techniques in the Game of Kriegspiel

Paolo Ciancarini
Dipartimento di Scienze dell'Informazione
University of Bologna
ciancarini@cs.unibo.it

Gian Piero Favini
Dipartimento di Scienze dell'Informazione
University of Bologna
favini@cs.unibo.it

Abstract

Monte Carlo tree search has brought significant improvements to the level of computer players in games such as Go, but so far it has not been used very extensively in games of strongly imperfect information with a dynamic board and an emphasis on risk management and decision making under uncertainty. In this paper we explore its application to the game of Kriegspiel (invisible chess), providing three Monte Carlo methods of increasing strength for playing the game with little specific knowledge. We compare these Monte Carlo agents to the strongest known minimax-based Kriegspiel player, obtaining significantly better results with a considerably simpler logic and less domain-specific knowledge.

1 Introduction

Imperfect information games provide a good model and testbed for many real-world problems and situations involving decision making under uncertainty. They typically involve a combination of complex tasks such as heuristic search, belief state reconstruction and opponent modeling, and they can be very difficult for a computer agent to play well. Some games are particularly challenging because at any time, the number of possible, indistinguishable states far exceeds the storage and computational abilities of present-day computers. In this paper, the focus is on one such game, Kriegspiel or invisible chess. It has several features that make it interesting: firstly, its rules are identical to those of a very well-known game and only the players' perception of the board is different, only being able to see their own pieces; secondly, it is a game with a huge number of states and limited means of acquiring information; and finally, the nature of uncertainty is entirely dynamic. This differs from other games such as Phantom Go or Stratego, wherein a newly discovered piece of information remains valid for the rest of the game. Information in Kriegspiel is scarce, precious and ages fast.

In this paper we present the first full application of Monte Carlo tree search to the game of Kriegspiel. Monte Carlo tree search has been imposing itself over the past years as a major tool for games in which traditional minimax techniques do not yield good results due to the size of the state space and the difficulty of crafting an adequate evaluation function. The game of Go is the primary example, albeit not the only one, of a tough environment for minimax where Monte Carlo tree search was able to improve the level of computer players considerably. Since Kriegspiel shares the two traits of being a large game and a difficult one to express with an evaluation function (unlike its perfect information counterpart), it is only natural to test a similar approach. This would also allow to reduce the amount of game-specific knowledge used by current programs by a large amount.

The paper is organized as follows. In Section 2, we introduce the game of Kriegspiel, its rules, and the most significant research results obtained in the field. In Section 3, we provide a few highlights in the history of Monte Carlo tree search methods applied to games, with an emphasis on Phantom Go, which is to Go what Kriegspiel is to chess. We then describe our Monte Carlo approaches in Section 4, showing how we built three Monte Carlo Kriegspiel players of increasing strength. Section 5 contains experimental tests comparing strength and performance of the various programs. Finally, we give our conclusions and furure research directions in Section 6.

2 Kriegspiel

Kriegspiel, named after the 'war game' used by the Prussian army to train its officers, is a chess variant invented at the end of the XIX century by Michael Henry Temple. It is played on three chessboards in different rooms, one for either player and one for the umpire. From the umpire's point of view, a game of Kriegspiel is a game of chess. The players, however, can only see their own pieces and communicate their move attempts to the umpire, so that there is no direct communication between them. If a move is illegal, the umpire will ask the player to choose a different one. If it is legal, the umpire will instead inform both players as to the consequences of that move, if any. This information depends on the Kriegspiel variant being played; on the Internet Chess Club, which has the largest player base for this game, it consists of the following.

- When something is captured: in this case the umpire will say whether the captured chessman is a pawn or another piece and where it was captured, but in the latter case he

will not say what kind of piece.

- When the king of the player to move is in check from one or more directions among the following: rank, file, short diagonal, long diagonal, knight.
- When the player to move has one or more capturing moves using his pawns ("pawn tries").
- When the game is over.

Because the information shared with the players is relatively little, the information set for Kriegspiel is huge. If one considers the number of distinct belief states in a game of Kriegspiel, the KRK (king and rook versus king) ending alone is not very far from the whole game of checkers. However, many of these states are in practice equivalent since there is no strategy that allows to distinguish among them in a normal game. This complexity is the primary reason why, for a long time, research only focused on algorithms for specific endings, such as KRK, KQK or KPK. Ferguson showed algorithms for solving KBNK and KBBK under certain circumstances; see, for example, [Ferguson, 1992]. It was only recently that the computer actually started to play Kriegspiel. State reconstruction routines are the main object of [Nance *et al.*, 2006]; [Russell and Wolfe, 2005] focuses on efficient belief state management in order to recognize and find Kriegspiel checkmates; [Parker *et al.*, 2005] is the first Monte Carlo approach to Kriegspiel, using and maintaining a state pool that is sampled and evaluated with a chess function. Finally, [Ciancarini and Favini, 2007] is based on ideas first applied to Shogi in [Sakuta, 2001] with the addition of evaluation functions. A similar program to the one described in [Ciancarini and Favini, 2007], with some improvements, will be used as a benchmark for the players in the present work.

3 Related work

Approaches to imperfect information games can be summarily divided into two broad categories: those which consider individual possible game states, and those which reason on an abstract model that is unable to see each single possible state. As far as Kriegspiel is concerned, [Parker *et al.*, 2005] is an example of the former type, being a Monte Carlo approach (albeit a very different one from Monte Carlo tree search) based on running possible states through a chess engine. On the other hand, [Ciancarini and Favini, 2007] is an example of the latter approach, reducing the game to an abstract model and essentially treating it as a perfect information game. Ever since the introduction of the UCT algorithm in [Kocsis and Szepesvari, 2006], Monte Carlo programs of the first type have been more and more successful over the past few years in facing very complex games such as Go, though they have also been used in games of imperfect information such as poker and Scrabble.

The Phantom Go program described in [Cazenave, 2006] may be one of the most significant examples, since the game is the Go equivalent of Kriegspiel: the player only sees his own stones and a move is declared illegal when he tries to place a stone on an occupied intersection. A Monte Carlo player obtained good results in terms of playing strength against humans on a 9x9 board. A thorough comparison of several Monte Carlo approaches to Phantom Go, with or without tree search, has recently been given in [Borsboom *et al.*, 2007]. Given the success of these methods, it is reasonable to wonder how similar methods would perform in a game with such high uncertainty as Kriegspiel. On the other hand, there are several differences worth mentioning between the two games.

- The nature of Kriegspiel uncertainty is completely dynamic: while Go stones are, if not immutable, at least largely static and once discovered permanently decrease uncertainty by a large factor, information in Kriegspiel ages and quickly becomes old. One needs to consider whether uncertainty means the same thing in the two games, and whether Kriegspiel is a harsher battlefield in this respect.
- There are several dozen combinations of messages that the Kriegspiel umpire can return, compared to just two in Phantom Go. This makes their full representation in the game tree very difficult.
- In Phantom Go there always exists a sequence of illegal moves that will reveal the full state of the game and remove uncertainty altogether; no such thing exists in Kriegspiel, where no sequence of moves can ever reveal the umpire's chessboard except near the end of the game.
- Uncertainty grows faster in Phantom Go, but also decreases automatically in the endgame. By contrast, Kriegspiel uncertainty only decreases permanently when a piece is captured, which is rarely guaranteed to happen.
- In Phantom Go, the player's ability to reduce uncertainty *increases* as the game progresses since there are more enemy stones, but the utility of this additional information often *decreases* because less and less can be done about it. It is exactly the opposite in Kriegspiel: much like in Battleship, since there are fewer enemies on the board and fewer allies to hit them with, the player has a harder time making progress, but any information can give him a major advantage.
- Finally, there are differences carried over from their perfect information counterparts, most notably the victory conditions. Kriegspiel is about causing an event that can happen suddenly and at almost any time, whereas Go games are concerned with the accumulation of score.

Because of these differences, the resemblance with Phantom Go might be more superficial than it can appear at a first glance and adds further motivation to see if analogous methods can work on both games despite their differences.

There is comparatively less research done on Monte Carlo approaches to games with a mutable, dynamic board. We recall, for example, [Chung *et al.*, 2005], which deals with real time strategy games, because it implements high-level planning with a Monte Carlo method. This is of particular interest to us because it better reflects the way a human Kriegspiel player thinks, giving himself strategic goals such as attacking a square and then considering which goal offers the best tradeoff between risks and rewards. In the quoted paper, games are simulated with the AI following different

strategies and then choosing the one with the highest victory ratio. This is an example of simulations run through a model of the game rather than the game itself, and as such the approach seems compatible with a Kriegspiel model like the one described in [Ciancarini and Favini, 2007].

4 Our approach

In this section, we provide three Monte Carlo methods for playing Kriegspiel, which we label A, B and C. These approaches are quickly summarized in Figure 1. Initially, we investigated an approach that was as close as possible to the Monte Carlo techniques developed for Go and other games; the first version of our program, approach A, was a more or less verbatim translation of established Monte Carlo tree search for Go. We developed the other two methods after performing unsuccessful tests, though these only differ in the simulation step, which is one of four phases in Monte Carlo tree search. MCTS is an iterative method that performs the following four steps until its available time runs out.

- **Selection.** The algorithm selects a leaf node from the tree based on the number of visits and their average value. Selection is a problem reminding of the n-armed bandit, where the agent needs to keep a balance between exploration and exploitation.
- **Expansion.** The algorithm optionally adds new nodes to the tree.
- **Simulation.** The algorithm somehow simulates the rest of the game one or more times, and returns the value of the final state (or their average, if simulated multiple times).
- **Backpropagation.** The value is propagated to the node's ancestors up to the root, and new average values are computed for these nodes.

Our approach A would implement the four steps of Monte Carlo tree search as follows.

- For Selection, our implementation used the standard UCT algorithm (Upper Confidence bound applied to Trees) first given in [Kocsis and Szepesvari, 2006]. This algorithm chooses at each step the child node maximizing the quantity

$$U_i = v_i + c\sqrt{\frac{\ln N}{n_i}},$$

  where v_i is the value of node i, N is the number of times the parent node was visited, n_i is the number of times node i was visited, and c is a constant that favors exploitation if low, and exploration if high. At each step the algorithm moves down the tree until it finds a leaf.
- For Expansion, our implementation would expand a new node with each iteration, which we considered to be a reasonable solution.
- Simulation raises a number of questions in a game of imperfect information, such as whether and how to generate the missing information, and how to handle subsequent moves. Existing research is of limited help, since to the best of our knowledge this is the first time MCTS is applied to a game with such high uncertainty - a game in which knowledge barely survives the next move or two. Go is relatively straightforward in that move handling can be as simple as playing a random move anywhere except in one's own eyes. It is also easier to estimate the length of a simulated Go game, which is generally related to the number of intersections left on the board. Kriegspiel simulations are necessarily heavier to compute due to the rules of the game.

  Our program A simulated games as follows: first, a random position was generated for the opponent's pieces. These positions were approximated using several heuristics such as remembering how many pieces are left on the board, and how many pawn can be on each file. Then, both players would play random moves until they drew by the fifty move rule or they reached a standard endgame position (such as king and rook versus king), in which case the game would be adjudicated. In order to make simulation more accurate, both players would almost always try to capture back or exploit a pawn try when possible - this is basic and almost universal human behavior when playing the game.
- We implemented standard Backpropagation, using the average node value as backup operator.

Approach A failed, performing little better than the random player and losing badly and on every time setting to a more traditional player based on minimax search. Program A's victory ratio was below 2%, and its victories were essentially random and unintentional mid-game checkmates. Investigating the reasons of the failure showed three main ones in addition to the obvious slowness of the search. First, the positions for the opponent's pieces as generated by the program were not realistic. The generation algorithm used probability distributions for pieces, pawns and king that were updated after each umpire message. While the probabilities were quite accurate, this did not account for the high correlation between different pieces, that is, pieces protecting other pieces. Kriegspiel players generally protect their pieces quite heavily, in order to maximize their chances of successfully repelling an attack. As a result, the program tended to underestimate the protection level of the opponent's pieces. Secondly, because moves were chosen randomly, it also underestimated the opponent's ability to coordinate an attack and hardly paid attention to its own defense.

Lastly, but perhaps most importantly, there is the subtler issue of *progress*. Games where Monte Carlo approaches have been tested most thoroughly have a built-in notion of progress. In Go, adding a stone changes the board permanently. The same happens in Scrabble. In Poker, the player is always given the same options. Kriegspiel, on the other hand, like real-time strategy games has no such notion; if the players do nothing significant, nothing happens. In fact, it can be argued that many states have similar values and a program failing to find a good long-term plan will either rush a very dangerous plan or just choose to minimize the risk by moving the same piece back and forth. When a Monte Carlo method does not perform enough simulations to find a stable

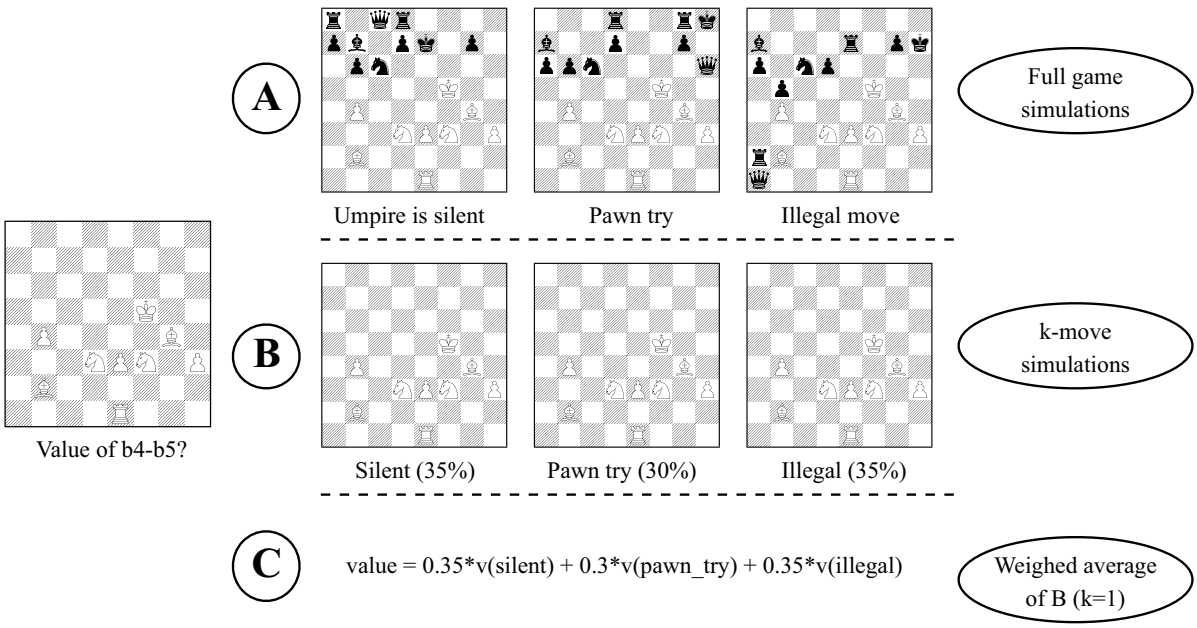

Figure 1: Comparison of three simulation methods. Approach A is standard Monte Carlo tree search, approach B simulates umpire messages only and for k-move runs, approach C immediately computes the value of a node in approach B for $k = 1$.

maximum, it can do either.

Hence, performing Monte Carlo on single states proved far too unstable for a generic program, though the approach may work in specific situations with a small information set. In order to overcome this problem, we define a second approach, called B. Approach B removes the randomness involved in generating single states and instead only simulates umpire messages, without worrying about the enemy layout that generated them. The same abstract model used to create states in approach A is now used to estimate the likelyhood of a given umpire message in response to a certain move. For example, there is a chance of the enemy retaliating on a capture one or more times and a chance of a move being illegal. These are all approximated with probability distributions on the various squares, just like in approach A. In particular, the model makes the following assumptions:

- The probability for a move to be illegal is equal to the sum of the probabilities of each square on the piece's path being occupied; for pawns, this includes the destination square unless the move is a capture.
- When a capture takes place, there is a 99% chance of the capturing piece being, in turn, captured by the opponent. If the player can immediately retaliate once more, this chance halves with each iteration.
- There is a 30% chance of the player's piece being captured when a check message is heard.
- The probability distribution for enemy pieces is updated with a simple algorithm after each move, which basically brings all values closer to the average and normalizes them. There is a 10% chance of the player suffering a capture, with more exposed pieces being more likely targets.

The second point of interest about method B is that it does not play full games as that proved to be too detrimental to performance. Instead, it simulates a portion of the game that is at most k moves long. The algorithm also accounts for quiescence, and allows simulations to run past the limit of k moves after its starting point in the event of a string of captures. The first move is considered to be the one leading to the tree node where simulation begins; as such, when $k = 1$, there is basically no exploration past the current node except for quiescence. Intuitively, a low value of k gives the program less foresight but increases the number of simulations and as such its short term accuracy; a high value of k should do the opposite. At the end of the simulated snippet, the resulting chessboard is evaluated using the only real notion of Kriegspiel theory in this method; that basically reduces to counting how many pieces the player has left, minus the number of enemy pieces left.

The third and final approach, called C, is approach B taken to the extreme for $k = 1$; it was developed after noticing the success of that value of k in the first tests. Since the percentages for each umpire message are known in the model, it is easy to calculate the results for each and average them. This operation, which builds implicit chance nodes (whose number would be enormous if actually created), makes it so each node needs only be evaluated once. Because simulations are assumed to instantly converge in this fashion, the backup operator is also changed from the average to the maximum node value. Of course, this is the fastest simulation strategy, blurring the line between simulation and a UCT-driven evaluation function (or, more accurately, a cost function in a pathfinding

algorithm), and it can be very discontinuous from one node to the next. If approach C is successful, it means that information in Kriegspiel is so scarce and of such a transient nature, as outlined in the previous section, that the benefits of global exploration by simulating longer games are quite limited compared to the loss of accuracy in the short run, thus emphasizing selection strategies over simulation strategies. Another way to think of approach C is as if simulations happened entirely on the tree itself rather than in separate trials, at the rate of one simulation per node. This is based on the assumption that good nodes are more likely to have good children, and the best node usually lies at the end of a series of good or decent nodes.

5 Tests

We test our approaches, with the exception of A which is not strong enough to be interesting, against a similar program to the one described in [Ciancarini and Favini, 2007]. The main feature of the Kriegspiel player in the cited paper is the use of *metapositions* as representations of the game's belief state that can be evaluated in a minimax-like fashion. Tests against humans on the Internet Chess Club showed that the minimax program's playing strength is reasonable by human standards, ranking above average at around 1700 Elo points; specifically, it possesses good defense but is often unable to find a good attack strategy unless the opponent is in a weaker position, which limits its strength as Kriegspiel favors a wise attacker. The program used in our tests is probably slightly stronger than the aforementioned one, since it performs a series of hard-coded checks that prevent the agent from making obvious blunders. It should be noted that our Monte Carlo players do not include these checks. The evaluation function of the minimax player is rather complex, consisting of several components including material, positional and information bonuses. By contrast, our Monte Carlo programs know very little about Kriegspiel: approaches B and C only know that the more pieces they have, the better. They know nothing about protection, promoting pawns, securing the center or gathering information.

The results of the tests are summarized in Figure 2. Programs are evaluated through comparison with the minimax player, with the value of k on the x axis and the difference in Elo points on the y axis. Thus, programs that perform worse than the minimax player are below 0 in the graph, and better programs are above 0. We recall that in the Elo rating system a difference of 200 points corresponds to an expected result of about 0.75 (with 1 being a win and 0.5 being a draw), and a difference of 400 points has an expected result of about 0.9. The minimax player itself always runs at the same, optimal setting for its evaluation function, requiring between 1 and 2 seconds to run. The program hits a performance plateau after this point, preventing it from further increases in performance.

After witnessing the failure of approach A, we limit our tests to approaches B and C. The Monte Carlo programs do not have particular optimizations and their parameters have not been thoroughly fine-tuned. The programs are all identical outside the simulation task, with the single exception of

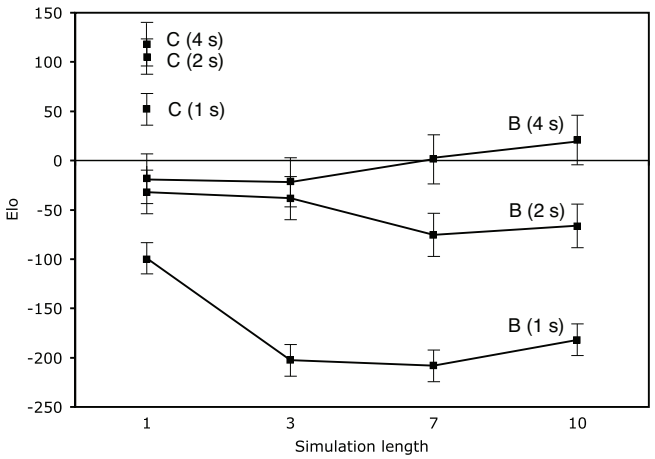

Figure 2: Comparison of Monte Carlo approaches B and C with a fixed-depth minimax player at different time settings and simulation depths, with error intervals.

the UCT exploration parameter c. Approach C uses a lower value of c leaning towards exploitation more on the basis that each node is only evaluated once. However, this different value of c has only a small beneficial value on the program: most of the advantage of the weighed average method lies in its speed, which allows it to visit many more nodes than the corresponding B program for $k = 1$ - the speedup factor ranges from 10 to 20 on average, everything else being equal. In fact, approach C lacks randomness altogether and could be considered a degeneration of a Monte Carlo technique.

Experimental findings more or less confirm our expectation, that is, lower values of k should be more effective under faster time settings, and higher values of k should eventually gain the upper hand as the program is given more time to reason. When k is low, the program can execute more simulations which make it more accurate in the short term, thus reducing the number of tactical blunders. On the other hand, given enough time the broader horizon of a higher k finds more strategic possibilities and longer plans through simulation that the lower k cannot see until they are encountered through selection.

At 1 second per move, $k = 1$ has a large advantage over the other B programs. Doubling the time reduces the gap among all programs, and at 4 seconds per move the longer simulations have a small but significant edge, actually outperforming the minimax player by a slight margin. The only disappointment came from the $k = 3$ players, which did not really shine under any time setting. It is possible that three moves is just not enough to consistently generate good plans out of random tries. Since Kriegspiel plans can be interleaved with basically useless moves that more or less maintain the status quo on the board, a ten-move sequence can contain a good three-move sequence with higher likelyhood.

Given the simplicity of the approach and the lack of specialized knowledge compared to the minimax player's trained parameters and pruning techniques, B programs are quite remarkable, though not as much as the performance of C type

programs. These can defeat the benchmark program consistently, ranking over 100 Elo points above it and winning about three times more games than they lose to it. Since approach C has basically no lookahead past the node being explored, we can infer that UCT selection is the major responsible for its performance, favoring the paths of least danger and highest reward under similar time settings to the minimax player's. The UCT exploration-exploitation method beats the hard pruning algorithms used by the minimax player, showing that in such a game as Kriegspiel totally pruning a node can often remove an interesting, underestimated line of play: there are relatively few bad nodes that can be safely ignored. It appears more profitable to allocate different amounts of time and resources to different moves, like in Monte Carlo tree search and the related n-armed bandit problem.

6 Conclusions and future work

There are several conclusions to be drawn from these experiments. First, they show that a Monte Carlo tree search algorithm can converge to good results in a reasonable amount of time even in a very difficult environment like Kriegspiel, whose lengthy simulations might at first appear to be a significant disadvantage of the method. However, precautions need to be taken so that the program does not end up sampling data that is too noisy to be useful; in this case, such a requirement is met by abstracting the game with a model in which single states are not important, and only their perception matters.

Secondly, we can explain the success of approach C, which is basically UCT with a node evaluation function, with its accuracy in simulating the very next move. Also, it can be argued that variable-depth UCT selection can outperform fixed-depth minimax in high-uncertainty situations even under these unorthodox premises. Still, approach B - the more traditional Monte Carlo method - seems to have the largest room for improvement. While experimental data indicates that stable evaluation of the first move is the largest factor towards improving performance and avoiding tactical blunders when time is short, a longer simulation with higher values of k provides better strategic returns under higher time settings. In particular, $k = 10$ shows great improvement at 4 seconds per move. It is possible that, with more effective simulation strategies and more processing power, approach B will be able to outperform approach C. It is too early to reach a conclusion on this point. A hybrid approach, treating the first move like C and the following moves like B, is also worth investigating.

The program as a whole can still be improved by a large factor. In the game of Go, Monte Carlo tree search is more and more often combined with game-specific heuristics that help the artificial player in the selection and simulation tasks. Since Monte Carlo methods are weak when they are short on time, these algorithms drive exploration through young nodes when there is little sampling data available on them. An example of such algorithms is the two progressive strategies described in [Chaslot *et al.*, 2008]. Since Kriegspiel is often objective-driven when played by humans, objective-based heuristics are the most likely candidates to make good progressive strategies, and research is already underway in that direction. There are several other optimizations borrowed from Go that might be useful under imperfect information, such as the all-moves-as-first heuristic.

References

[Borsboom *et al.*, 2007] J. Borsboom, J. Saito, G. Chaslot, and J. Uiterwijk. A comparison of Monte-Carlo methods for Phantom Go. 2007.

[Cazenave, 2006] T. Cazenave. A Phantom-Go program. *Lecture Notes in Computer Science*, 4250:120–125, 2006.

[Chaslot *et al.*, 2008] G. Chaslot, M. Winands, J. van der Herik, J. Uiterwijk, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(3):343–352, 2008.

[Chung *et al.*, 2005] M. Chung, M. Buro, and J. Schaeffer. Monte Carlo planning in RTS games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005.

[Ciancarini and Favini, 2007] P. Ciancarini and G. Favini. Representing Kriegspiel states with metapositions. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI 07)*, pages 2450–2455, Hyderabad, India, 2007.

[Ferguson, 1992] T. Ferguson. Mate with bishop and knight in Kriegspiel. *Theoretical Computer Science*, 96:389–403, 1992.

[Kocsis and Szepesvari, 2006] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. *Lecture Notes in Computer Science*, 4212:282–293, 2006.

[Nance *et al.*, 2006] M. Nance, A. Vogel, and E. Amir. Reasoning about partially observed actions. In *Proc. 21st National Conference on AI*, Boston, USA, 2006.

[Parker *et al.*, 2005] A. Parker, D. Nau, and VS. Subrahmanian. Game-tree search with combinatorially large belief states. In *Int. Joint Conf. on Artificial Intelligence (IJCAI05)*, pages 254–259, Edinburgh, Scotland, 2005.

[Russell and Wolfe, 2005] S. Russell and J. Wolfe. Efficient belief-state AND-OR search, with application to Kriegspiel. In *Int. Joint Conf. on Artificial Intelligence (IJCAI05)*, pages 278–283, Edinburgh, Scotland, 2005.

[Sakuta, 2001] M. Sakuta. *Deterministic solving of problems with uncertainty*. PhD thesis, Shizuoka University, Japan, 2001.