Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Monte Carlo Algorithms for Playing an Imperfect-Information Chess Variant

Vojtěch Foret

Supervisor: Mgr. Viliam Lisý PhD.
Field of study: Open informatics
Subfield: Informatics and Computer Science
January 2019

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Foret Vojtěch**	Personal ID number:	**456991**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Monte Carlo Algorithms for Playing an Imperfect-Information Chess Variant

Bachelor’s thesis title in Czech:

Monte Carlo algoritmy pro hraní šachů s neúplnou informací

Guidelines:

Games with imperfect information are fundamentally more complicated than games with perfect information, because the players have to reason about large sets of possible states consistent with available information.
The student will:
1) review existing algorithms for playing imperfect information games with focus on Monte Carlo methods;
2) implement a method for representing and maintaining the set of possible states of the game;
3) implement two different algorithms for playing an imperfect-information chess variant (e.g., Kriegspiel);
4) experimentally find suitable parameters for the algorithm and rigorously compare their performance.

Bibliography / sources:

[1] Paolo Ciancarini, Gian Piero Favini, Monte Carlo tree search in Kriegspiel, Artificial Intelligence, Volume 174, Issue 11, Pages 670-684, 2010.
[2] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set monte carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games, 4, no. 2, 120-143, 2012.
[3] Jeffrey R. Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In AAAI 2010, 134-140.

Name and workplace of bachelor’s thesis supervisor:

Mgr. Viliam Lisý, MSc., Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **10.01.2018** Deadline for bachelor thesis submission: **08.01.2019**

Assignment valid until: **30.09.2019**

Mgr. Viliam Lisý, MSc., Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student’s signature

Acknowledgements

I would first like to thank my thesis supervisor Mgr. Viliam Lisý PhD. The door to his office was always open whenever I had a question about my research. He allowed this project to be my own work, but steered me in the right the direction whenever I needed it.

I would also like to thank my parents who provided me with support and encouragement during my student years.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis

In Prague, 1. January 2019

Abstract

Topic of this thesis are algorithms for games with imperfect information. This thesis's goal is to implement some of the existing algorithms used in games with imperfect information and test their usefulness on the game of kriegspiel, a version of a well known game of chess. Handling the imperfect information in this game is a great challenge because the player recieves only very little information about state of the game and the information also ages very quickly.

Keywords: Games with imperfect information, Chess, Kriegspiel, Monte Carlo Tree Search, Perfect Information Monte Carlo

Supervisor: Mgr. Viliam Lisý PhD.

Abstract

Tématem této bakalářské práce jsou algoritmy pro hry s neúplnou informací. Tato práce si bere za cíl implementovat některé z již existujících algoritmů používaných na hry s neúplnou informací a testovat jejich použitelnost na hře Kriegspiel, verzi známe hry šachy. Zvládnutí neúplné informace v této hře je velká výzva, jelikož hráč dostává o stavu hry jen opravdu velmi málo informací a informace se velmi rychle stávají zastaralými.

Keywords: Hry s neúplnou informací, Šachy, Kriegspiel, Monte Carlo Tree Search, Perfect Information Monte Carlo

Title translation: Monte Carlo algoritmy pro hraní šachů s neúplnou informací

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

For ages people have played board games, either for fun or to sharpen their brains. Even in the present day, people often sit down to play a wide variety of these games with their families or friends. Board games have a long tradition and are an important part of human culture. Moreover they are an excellent test bed for studies in the field of artificial intelligence. They provide a simple model with, when compared to the real world, very simple set of rules. Imperfect information is an integral part of the real world and every agent has to deal with decision making under uncertainty which quickly made it a very important part of artificial intelligence. A good test bed for these situations are games with imperfect information. In these games player doesn't know the exact game state, only a set of possible states. This makes games with imperfect information very difficult and challenging for computer programs and humans.

This thesis focuses on the game kriegspiel. Kriegspiel is a very interesting game because it is an imperfect information variant of chess. It combines rules identical to chess, one of the most popular board games in the world, and very strongly imperfect information. Chess is transformed into one with imperfect information by letting players see only their own pieces. Player receives only very little information about enemy pieces which makes the number of possible states particularly large and impossible to store in any present-day computer. The information in kriegspiel also ages fast and rarely lasts longer than a few moves. This makes kriegspiel a particularly challenging and very difficult for a computer program to play well.

A lot of work and research has been put into solving games with perfect information and computer programs have defeated the best human players in many of these games (e.g. chess). After a long time of research, computer programs have begun to be succesfull even in games with imperfect information. Perfect information Monte Carlo is a technique that deals with the problem of perfect information by avoiding it. Even though it is very often criticized for problems arising from this approach, it's success in a variety of games is undeniable. Another very interesting approach to solving games with imperfect information is relatively new AI technique called information set Monte Carlo tree search (ISMCTS). Monte Carlo Tree Search (MCTS) has been succesfull in many games with perfect information and it requires minimal to no domain knowledge even though it can improve it's performance. ISMCTS is a way to apply MCTS to games with perfect information and it avoids some of the downsides of Perfect information Monte Carlo.

1.2 Related work

Perfect information Monte Carlo (PIMC) as described in [CPW12] was for a long time the aproach to go for when dealing with games with imperfect information. Despite it's flaws, in many games like Bridge and Skat it plays on expert level. In [CPW12] a new version of Monte Carlo Tree Search for games with imperfect information is introduced - Information Set Monte Carlo Tree Search. It is shown that it overcomes some of the often criticized problems of PIMC and in some domains performs better.

Not too much work has been done on kriegspiel itself. In [CF07] an aproach called metaposition is introduced. Metaposition stores a superset of the current information set on a single virtual chessboard and allows to create evaluation function and use minimax methods like it was a game with perfect information. It performed very well among other kriegspiel playing computer programs but was not strong enough to face human players. In [CPW12] it is shown that Information Set Monte Carlo Tree Search can be with some domain-specific knowledge very strong and even outperform kriegspiel. It can even face human players although it does not play on the expert level just yet.

1.3 Kriegspiel

Kriegspiel is a chess variant deriving from a German war game (Kriegsspiel) used to train military officers. It's a variant with regular chess pieces in which player has exact information about position of only his own pieces. This nature of the game is also reason why it is sometimes reffered to as Blind Chess. All the rules remain the same as in the regular game of chess. The game of chess can end for several reasons:

- Checkmate - The player to move has his king in check (threatened with capture) and there is no legal move that can remove the threat. Chcekmating the opponent is the only way to win the game of chess. If the game is terminated for any other reason, it ends in a draw.
- Stalemate - The player to move has no legal moves available and he is not in check.
- 3-fold repetition - The same position occurs three times (all the pieces are at the same position, it is the same player's turn and he has the same legal moves).
- 50-move rule - No capture or pawn move has occured in the last 50 moves.
- Insufficient material - The game reaches position where there is no possible series of legal moves that would cause a checkmate.

The only way players can obtain information about opponent's pieces is through a referee (umpire) who is the only one with information about positions of both players' pieces. After a player attempts a move, umpire decides whether the move is legal or not. If it is illegal, player can choose a different one. If the move is legal, umpire announces consequences of the move, if there are any. From the view of umpire it is normal game of chess, only the information shared with the players depends on the Kriegspiel variant being played. Probably the most widely spread ruleset is the one usen on the Internet Chess Club. The umpire messages there are following:

- No or Illegal - If the move is illegal, umpire will ask the player to choose a different move.
- White to move/ Black to move - If the move is legal and it's now opponent's turn

- Pawn captured - When a pawn is captured. Umpire will also specify the position of the captured pawn.
- Piece captured - When a piece that is not a pawn is captured. Umpire will also specify the position of the captured pawn.
- Check - In the case that a King is in check

Chapter 2

Background

2.1 Perfect Information Monte Carlo

2.1.1 Algorithm

Perfect Information Monte Carlo (PIMC)(sometimes also called determinization) is a relatively simple approach to playing games with imperfect information. Rather than dealing with the imperfect information, it applies classic algorithms on an equivalent game with perfect information. The term determinization refers either to the process of converting a game with imperfect information to an instance of a game with perfect information or to the instance itself. Determinization is chosen from the current information set. For example, a determinization of a game of Kriegspiel is a chessboard with all pieces visible.

Using this approach, we choose a number of determinizations from the current information set and use algorithms for games with perfect information for every determinization. Then we combine results of these algorithms to get a decision for the original game. Determinizations could be chosen using a probability distribution over the information set but in this thesis i chose to chose them uniformly at random. Advantage of determinization is that it allows us to choose the method used to evaluate particular determinizations. Because of the nature of kriegspiel determinization, I am able to use sophisticated minmax based algorithms that are very strong at playing chess.

2.1.2 Characteristics

Allthough determinization shows success in many games, it is not without flaws. Since determinization uses perfect information game playing algorithms on particular determinizations, it doesn't "realize" that it is playing a game with imperfect information. Therefore it will never make the decision to make a move that reveals some information or a move that makes sure that as much information as possible stays hidden from the opponent. That brings us to another problem, some determinizations might be extremely unlikely because the opponent can simply avoid them - that would make results of these determinizations useless. The last significant problem with determinization is strategy fusion. Player can make different decisions in different determinizations but he cannot make different decisions in different states of the same information set because he cannot distinguish them. For example, there can be a move after which the player can in every determinization make a decision that leads him to victory. However, in every determinization the winning decision can be different. So it is even possible that determinization will say that the said move is a sure victory while in reality the probability that the player will choose the right move after that is very low.

2.2 Monte Carlo Tree Search

This chapter focuses on describing Monte Carlo Tree Seaarch (MCTS) for games with perfect information as summarised in [CPW12].

2.2.1 Algorithm

MCTS starts with only the root node being the current state of the game. With each iteration the algorithm builds bigger and bigger part of the search tree focusing more on the more promising areas. Each iteration has four following steps:

- Selection - Algorithm starts in the root and descends the already existing tree. Descending stops when the algorithm reaches a node that is either a terminal state or reaches a node that has children that have not been addded to the tree yet. There are several existing selection strategies.

The simplest one is to select randomly. It can be also aproached as a multiarmed bandit problem and UCT as described in 2.4 can be used. Ideal selection strategy finds balance between exploration (descending to parts of the tree that have been visited less often) and exploitation (choosing the more promising parts of the tree).

- Expansion - In this step, a new node is added to the tree. A simple and popular expansion strategy is to add one node per iteration but it could also be a small subtree. This node can be chosen at random from the children of the node that we find ourselves in that is not yet in the tree.
- Simulation - A simulation is run from the newly added node. There are different simulation strategies. The most popular one is to choose moves randomly until the end of the game is reached. Using domain-specific information can improve reliability of the result of this simulation. Even a knowledge-based evaluation function can be used.
- Backpropagation - The algorithm returns back to the root and statistics stored in all the visited nodes are updated using the newly gained results from the simulation.

After a given number of iterations or when time runs out, the most favourable action from the root is chosen using the statistics that the algorithm gathered.

2.2.2 Characteristics

MCTS is an aheuristic algorithm. In games where reliable heuristics have been found, minimax based algorithms perform very well (e.g. Chess). Since MCTS requires little to no domain-specific knowledge, it can be very usefull in games where it is difficult to formulate reliable and useful heuristics (e.g. Go). However, using some domain-specific knowledge can signifficantly improve performance of MCTS.

After adding each node to the tree in each iteration, MCTS immediately backpropagates the outcome up in the tree so the statistics stored in the nodes are always up-to-date. This means that the algorithm can return an action at any moment making it an anytime algorithm. This is an advantage over minimax algorithm although iterative deepening can also make minimax into an anytime algorithm.

MCTS builds an asymmetric tree and therefore performs an asymmetric search. Depending on selection strategy, the algorithm can put more computation time into the more promising parts of the tree and find balance between exploration and exploitation.

2.3 Information Set Monte Carlo Tree Search

Information Set Monte Carlo Tree Search (ISMCTS) should help us overcome some of the problems with determinization because it embraces the imperfect information nature of the game.

In MCTS the nodes represent game states and edges represent moves. In ISMCTS they represent the same thing from the perspective of the root player and so taking into account that it is a game of imperfect information. Therefore nodes represent information sets and edges represent moves from the point of view of the root player - actions that are indistinguishable for the root player share a single edge. In the case of kriegspiel edges representing opponent's moves correspond to umpire messages retrieved by the root player.

2.3.1 Algorithm

At the beginning of each iteration the algorithm chooses a determinization at random just like the Perfect Information Monte Carlo. The whole iteration then works with this determinization and has 4 steps that are only slightly different from the MCTS's :

- Selection - Algorithm descends the tree using only the edges compatible with the chosen determinization. When choosing the action performed by the root player, it can be viewed as subset-armed bandit problem and we can use UCT. Actions of opponent are chosen at random because nodes in the tree do not store any information that would help us make a better choice.
- Expansion - There has to be action compatible with the chosen determinization leading to the newly added node.
- Simulation and Backpropagation are the same as in MCTS

This algorithm solves the strategy fusion. Unfortunately it assumes that the opponent choses his moves at random so the opponent model is very weak. This can be solved by maintaining a separate tree for the opponent that corresponds to the opponents point of view. But this algorithm is not part of this thesis.

2.4 Upper Confidence Bounds Applied to Trees

As described in [CPW12] UCT is a variant of UCB1 used in trees. It is an algorithm used in multi-armed bandit problem. It is used to balance exploration and exploitation. I use UCT to choose to which node to descend to in selection step of ISMCTS iteration. The node to descend to is chosen as the one with the highest $\overline{X}_j + c\sqrt{\frac{\ln n}{n_j}}$, where $\overline{X}_j$ is value of node j, n is the number of times the parent node was visited, n_j is the number of times the node j was visited and c is an exploration constant. Finding the optimal value of c for my algorithms will be one of the many goals of my bachelor thesis.

In ISMCTS instead of multi-armed bandit problem we have to solve subset multi-armed problem. We deal with that by counting into n only times when the parent node was visited and the current node was available in chosen determination.

Chapter 3

Implementation

3.1 GNU-chess

GNU-chess is mainly written in C. I sometimes used C++ here and there. GNU-chess engine offers many efficient functions and data strucures that I used in my parts of the code, for example finding legal moves, board data structure, list of moves. I implemented negamax player, random player, possibility to play from the terminal, different sizes of chessboards and completely changed the game loop and implemented the umpire.

3.2 Chessboards

This program offers the game of Chess or Kriegspiel on several chessboards of different sizes. The main focus is put on chessboard 8x8 and 4x4. Chessboard 4x4 has 2 variants. One of them is Silverman 4x4 and the second one is a reduced form of Silverman with a Bishop instead of a Queen and no Pawns. Other chessboard sizes from 4x4 to 8x8 are also available.

3.3 Players

A class Player is an interface for agents. There are several implementations:

- Terminal Player

  Allows user to play from terminal.

- Random Player

  Random player can play Kriegspiel and Chess on any chessboard from 4x4 to 8x8. It chooses a random move from all legal moves. The only intelligence, it possesses, is that when playing Kriegspiel and umpire says that it's move was illegal, it will choose from the rest of the pseudolegal moves.

- Negamax Player

  It's a simple minimax player with alpha-beta pruning. The evaluation function uses only material value. It cannot play Kriegspiel. It plays only chess but it can play on any chessboard from 4x4 to 8x8.

- Determinization Player

  Player using described determinization algorithm. Each determinization is evaluated by gnu chess engine.

- ISMCTS Player

  Player using described ISMCTS algorithm. It has two different implemented simulation versions.

- Gnu-chess engine

  GNU-chess player uses Principal variation search with iterative deepening. It also uses search enhancements like transposition table and null-move pruning. Evaluation function is also more complex than the one that my negamax uses. It includes for example material value, mobility, king safety and more. It cannot play Kriegspiel. It plays only chess on chessboard 8x8.

3.4 Information Set Representation

Because the information set is almost always too large, it is a good idea to remember only a resonable amount of possible states and throw away the rest. Both determinization and ISMCTS use only a subset of the information set so they don't mind. Unfortunately in case that the true state of the game is one of the states thrown out, it is possible that the information set will get thinner and thinner and, unless the true game state is added to the information set because one of the possible enemy moves lead there, it will eventually be empty. To avoid that, it is needed to generate new samples and maintain a pool of samples - they are generated in random.

Chapter 4
Experiments and Evaluation

In this chapter, I present experiments that I performed. I compared 5 algorithms:

1. Random player choosing uniformly among all legal moves. In this chapter I will call him Dummy1.
2. Random player choosing uniformly among all legal moves. Only if his piece gets captured and he can recapture, then he prefers to recapture. In this chapter I will call him Dummy2.
3. ISMCTS player where simulation is comparing number and quality of pieces and assigning one of values 0; 0.5; 1 depending on who is better off. In this chapter I will call him ISMCTS1.
4. ISMCTS player where simulation is performing random moves till the end of the game. In this chapter I will call him ISMCTS2.
5. Determinization player using gnu chess engine to evaluate particular determinizations.

Exploration constant also has to be experimentally found for ISMCTS players.

In all these experiments, algorithms have only one thread availabe and they are given 2 second for every move.

4.1 Experiments

4.1.1 Dummy1 vs. Dummy2

The first experiment is to see whether slightly more sophisticated random player has the edge over a plain random player. When his piece is captured, player Dummy2 will recapture if possible. In the game of kriegspiel, it is a usual behaviour because when piece gets captured, it is one of the few situations when a player has exact information about where one of the enemy's pieces is and therefore it is very often clever to recapture if possible.

In the Table 5.1 we can see that Dummy2 has indeed slightly higher win rate than Dummy1 even though the difference is really small. Large portion of games ended as a draw because for random players it is not easy to checkmate the opponent and the game gets very often terminated due to the 50-move rule.

Even though the difference is insignificant, I will use Dummy2 to find suitable exploration constant for ISMCTS algorithms.

4.1.2 Finding Exploration Constant

I performed an experiment where players ISMCTS1 and ISMCTS2 with exploration constant 0.25, 0.5, 0.75,...,1.75, 2.0 played against player Dummy2. These values for exploration constant were suggested in [CPW12]. Each ISMCTS player played 80 games per constant with Dummy2; 40 as white and 40 as black.

Tables 5.2 and 5.3 show that none of the versions of ISMCTS is particularly sensitive to changes in the exploration constant. However best performed players ISMCTS1 with constant equal to 1.5 and ISMCTS2 with constant equal to 0.75 and therefore I will use these constants in following experiments.

White	Dummy1	Dummy2	Average
Dummy1 wins	2	2	1.0%
Dummy2 wins	6	12	4.5%
Draw	192	186	84.5%

Table 4.1:

	c = 0.25	c = 0.5	c = 0.75	c = 1.0	c = 1.25	c = 1.5	c = 1.75	2.0
ISMCTS1 wins	6	5	4	4	5	7	4	3
Dummy2 wins	2	1	1	1	1	1	3	1
Draw	72	74	75	75	74	72	73	77

Table 4.2:

	c = 0.25	c = 0.5	c = 0.75	c = 1.0	c = 1.25	c = 1.5	c = 1.75	2.0
ISMCTS2 wins	8	10	12	8	5	7	11	4
Dummy2 wins	1	2	1	2	3	5	4	1
Draw	71	68	67	72	68	65	75	

Table 4.3:

4.1.3 ISMCTS1 vs. Dummy2

In this experiment ISMCTS1 with exploration constant equal 1.5 played 400 games against Dummy2; 200 of them as white and 200 as black. We can see the results in the Table 5.4. As we can see ISMCTS1 has slightly higher win rate then random player but given how much more sophisticated he is, it is really not by a lot.

4.1.4 ISMCTS2 vs. Dummy2

I performed the same experiment with ISMCTS2 against Dummy2. This ISMCTS player uses very common and simple simulation strategy and as we can see in the Table 5.5 he did a little bit better against a random player than ISMCTS1.

4.1.5 ISMCTS1 vs. ISMCTS2

Results of games between ISMCTS1 and ISMCTS2 are in the Table 5.6. Just as ISMCTS2 did slightly better against Dummy2 than ISMCTS1, he also slightly beated ISMCTS1. However, the difference between these two versions of the same algorithm are so close to each other that given the number of games, it is not statistically signifficant.

White	ISMCTS1	Dummy2	Average
ISMCTS1 wins	21	8	7.25%
Dummy2 wins	4	4	2%
Draw	175	188	90.75

Table 4.4:

White	ISMCTS2	Dummy2	Average
ISMCTS2 wins	23	27	12.5%
Dummy2 wins	9	2	2.75%
Draw	168	171	84.75%

Table 4.5:

White	ISMCTS1	ISMCTS2	Average
ISMCTS1 wins	3	33	9%
ISMCTS2 wins	41	4	11.25%
Draw	136	163	74.75

Table 4.6:

4.1.6 Determinization vs. Dummy2

As we can see in the Table 5.7, determinization player has win rate 46% against Dummy2 while Dummy2 wasn't able to win a single game.

4.1.7 Determinization vs. ISMCTS

in the Tables 5.8 and 5.9 we can see that Determinization player wins 40.75% games against ISMCTS1 and 50.75% games against ISMCTS2. Determinization player has a strong gnu chess engine behind him and despite the flaws of the determinization algorithm, he is still stronger than IMCTS as implemented in this thesis.

ISMCTS2 once again performed slightly better than ISMCTS1.

White	Determinization	Dummy2	Average
Determinization wins	91	93	46%
Dummy2 wins	0	0	0%
Draw	109	107	54%

Table 4.7:

White	Determinization	ISMCTS1	Average
Determinization wins	85	78	40.75%
ISMCTS1 wins	0	1	0.25%
Draw	115	121	59%

Table 4.8:

White	Determinization	ISMCTS2	Average
Determinization wins	116	87	50.75%
ISMCTS2 wins	12	6	4.5%
Draw	72	107	44.75%

Table 4.9:

Chapter 5
Conclusion

First, we described some of the algorithms that are used to play games with imperfect information, namely Perfect Information Monte Carlo (or determinization) and Information Set Monte Carlo tree search. I Implemented these algorithms and tested them on the game of Kriegspiel.

PIMC performed very well against random players and also against ISMCTS players. It had a very strong gnu chess engine behind it. ISMCTS did not perform well. Only very simple simulation was implemented and as mentioned in [CPW12] this implementation of ISMCTS very often cannot be distuinguished from random player. Although it had higher win rate than random player, the difference was so small that it has no statistical signifficance. It appears that more domain-specific knowledge has to be implemented into ISMCTS for it to play kriegspiel decently.

Appendix A

Bibliography

[CPW12] Peter I Cowling, Edward Jack Powley and Daniel Whitehouse, *Information Set Monte Carlo Tree Search*, IEEE Transactions on Computational Intelligence and AI in Games, **4** (2012), 120–143.

[CF07] Paolo Ciancarini and Gian Piero Favini, *Representing Kriegspiel States with Metapositions*, IJCAI, (2007).

[CPW12] C. B. Browne et al., *A survey of Monte Carlo tree search methods*, IJCAI, **4** (2012), 1–43.

[CPW12] Richard Long, Jeffrey and R. Sturtevant, Nathan and Buro, Michael and Furtak, Timothy, *Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search*, Proceedings of the National Conference on Artificial Intelligence, **4212** (2010), 1–43.

[CPW12] Kocsis L., Szepesvári C., *Bandit Based Monte-Carlo Planning*, Lecture Notes in Computer Science, **1** (2006).

[CPW12] Paolo Ciancarini, Gian Piero Favini, *Monte Carlo tree search in Kriegspiel*, Artificial Intelligence, **174** (2010), 670–684.