# Determinization with Monte Carlo Tree Search for the card game Hearts

Freek Bax
University of Utrecht

*Monte Carlo Tree Search (MCTS)* is a popular algorithm used in AI for games. It is most famous for its implementation in the game Go. *Determinization* is a technique used to extend an algorithm for a game of perfect information to a game of imperfect information. It does this by determinizing the lacking information and calculating the average best move over all the instances with the perfect information algorithm. This paper provides an implementation of MCTS for the card game Hearts and it uses determinization to extend MCTS to games of imperfect information. We analysed the influence of better sampling in the determinization process on the performance of the player. We found that an improvement in the quality of the samples improves the performance of the player. We also show that inference methods could further increase the performance.

**Contents**



## 1 Introduction

Since the beginning of research in AI, games have been a popular topic sparking many innovations that pushed the boundaries of AI. Chess is a prime example of this with influential papers dating back to the 40s (de Groot, 1946) and 50s (Shannon, 1950). Some even call it the drosophila of AI. Although this might be an exaggeration, it shows how influential Chess research has been (Ensmenger, 2012). The goal of AI for board games is to beat human experts and eventually reach superhuman level. A famous breakthrough is IBM's Deep Blue beating Chess world champion Kasparov in 1997 (Campbell, Hoane Jr, & Hsu, 2002). One of the hardest games for AI to crack is the Chinese board game Go, due to the large branching factor of the game tree and lack of a good evaluation function. Monte Carlo Tree Search (MCTS) yielded a breakthrough for Go, with the famous AlphaZero from DeepMind eventually beating human champions (Silver et al., 2016). Most research of AI in board games focuses on games of perfect information, like Chess and Go. However there has also been development in imperfect information games; where some information is hidden. In the card game poker a recent development has pushed AI to superhuman level (Brown & Sandholm, 2018). But improvements can still be made in a lot of other games of imperfect information.

This is why we implemented and analysed an AI for the trick-based card game Hearts, a game of imperfect information. The AI uses MCTS with determinization. MCTS is an algorithm which builds a game tree of the search space and estimates the rewards of the states using simulations. Determinization is a technique to extend an algorithm suitable for perfect information games to imperfect information games. It samples possible instances of the imperfect information and solves the determinized perfect information games from the sample. The average best action is then chosen as the best option for the imperfect information game. For Hearts, determinization is used to sample opponents hands. We analysed different methods of sampling the opponents hands and analysed the influence of the quality of samples on the strength of the player.

Earlier research into determinization and MCTS has been done in a few different games. The first successful implementation of determinization was in the card game bridge, this used determinization in combination with more standard game tree search methods (Ginsberg, 1999). MCTS has been implemented to play the perfect information variant of Hearts successfully (Sturtevant, 2008). MCTS has also been used with Determinization in the chinese Card game Dou Di Zhu (Powley, Whitehouse, & Cowling, 2011) and Magic: The Gathering (Cowling, Ward, & Powley, 2012). Other areas

where determinization in combination with MCTS has been researched are in the stochastic single played game Klondike solitaire (Bjarnason, Fern, & Tadepalli, 2009) and probabilistic planning (Yoon, Fern, & Givan, 2007). However, the effect of the quality of samples in determinization has never been investigated.

MCTS is a versatile algorithm which can easily be be applied in different applications, since it does not need a lot of domain-specific knowledge like an evaluation function. Our research could answer whether domain-specific knowledge is necessary to optimize determinization. Recent state-of-the-art development in AI for games shows promising results for combining reinforcement learning techniques like MCTS and machine learning approaches like neural networks. This approach has beaten humans in computer games like Atari (Guo, Singh, Lee, Lewis, & Wang, 2014), (Mnih et al., 2015) and 3d games like Doom (Dosovitskiy & Koltun, 2016), but also in the board game Go (Silver et al., 2017). We hope to contribute to the broad spectrum of implementations of MCTS (Browne et al., 2012). Understanding and improving MCTS for imperfect information games could also help extend the combination of reinforcement learning and machine learning in environments with less information then current applications.

The next section will introduce the game Hearts. It will also provide background information on game theory and it will introduce Monte Carlo Tree Search and Determinization. Section 3 will explain the implementation of the algorithm and the setup of the experiments. After that section 4 gives an overview of the results. In section 5 the results are interpreted. In the conclusion we briefly summarize the results and give recommendations on future research.

## 2 Background

### 2.1 Hearts

Hearts is a trick-based card game, usually played by four players, where the goal is to minimize points taken. There are a lot of different variants out there. Most of the variants change the swapping phase of the game or add additional rules to when a suit can or must be played. We will test a basic variant with no swapping phase and no addition rules. The rewards are from the 'Black Lady' variant of the game, since this is by far the most played reward set. The amount of cards played with can vary: the internationally known version is played with a standard deck of 52 cards, while a Dutch variant is played with a piquet deck of 32 cards. This does not affect the rules, but only the starting hand sizes, which are 13 for a standard deck and 8 for a piquet deck. We will use the piquet variant with 32 cards in the deck. This reduces the search space and therefore also reduces computation time.

Every round of Hearts starts with all 4 players having a full hand. The player with the lowest club leads the first trick. In every trick, all players play one card in a clockwise order. The suit of the card lead (played first) is the suit of the trick. Every player must play the suit of the trick if possible, otherwise they can play any card from their hand. The player with the highest card of the suit of the trick wins the trick. The winner of the trick will lead the next trick. All tricks are played this way until all player hands are empty. After this the points are rewarded, every heart taken is 1 point and the Queen of spades is worth the amount of Hearts in the deck. So for a standard deck 26 points are rewarded, 13 Hearts and 13 for the queen of spades. For a piquet deck 16 points are rewarded, 8 Hearts and 8 for the queen of spades. There is one exception to this. If a player takes all points in a round, he gets awarded zero points and all other players get rewarded the maximum amount of points for the round. This is referred to as "Shooting the moon". When a player reaches a certain amount of points at the end of a round, the game is over. The player with the least amount of points wins the game. The amount of points is 100 for the 52 deck variant of Hearts. For the piquet variant there are different versions with varying amounts of points. We will use 75 points as the ending criteria.

### 2.2 Game Theory

Before we introduce the Monte Carlo Tree Search algorithm, we will first give some background into Game Theory. For detailed information about game theory and other related topics see: Artificial Intelligence: A Modern Approach (Russell & Norvig, 2002). Games are a multiagent environment. When an agent makes decisions in these kinds of environments it needs to consider the actions of other agents and how they affect the outcome of its decisions. Games are a special form of multiagent environments: they are competitive. This means the goals of the agents are in conflict with each other. A game is formally described by the following components:

- $S$: The set of states in the game. $S_0$ is a special kind of state, the initial state. In Hearts, a state can be seen as a combination of hands of players, cards played in the current trick and all the tricks taken by players. A starting state in Hearts is a state where all the players have a full hand.
- $S_T$: The set of terminal states. In Hearts a state is a terminal state when all the players have empty hands.
- $n$: The number of players, in our variant of Hearts 4.
- $A$: The set of actions. In Hearts the actions are a player playing a card.
- $f : S \times A \rightarrow S$: The transition function. This function takes a state and action, which it transforms into the new state. In Hearts this combines a state and a card

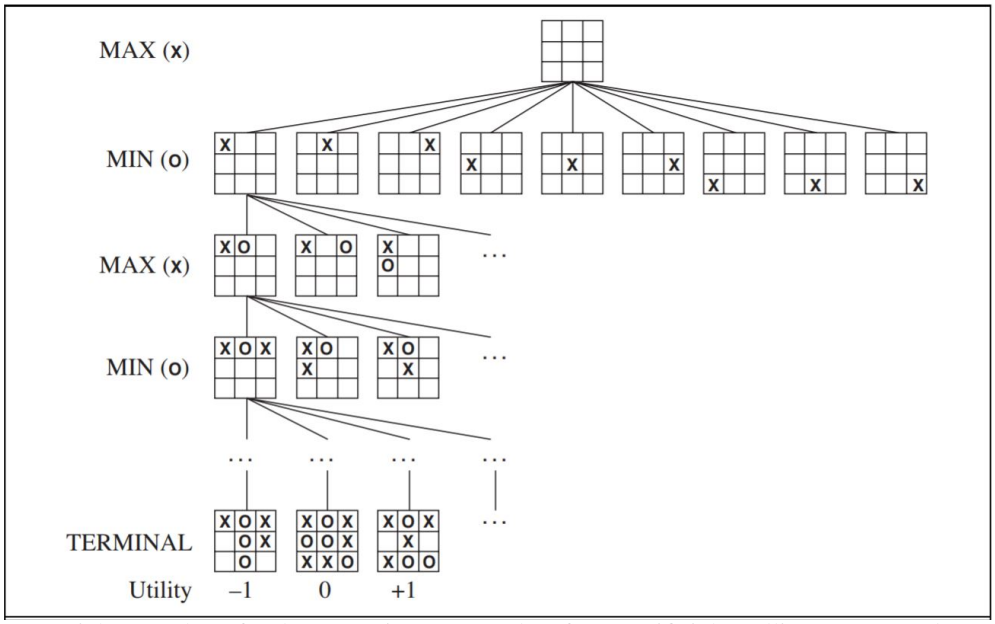


*Figure 1.* A partial game-three for the game tic-tac-toe, taken from Artificial Intelligence: A Modern Approach.

played by a player, which it transforms into the state where the card is played.

- $R : S_T \rightarrow \mathbb{R}$: The utility function. A function which gives the rewards for a given terminal state. In Hearts it looks at the cards the players have taken and gives the players the points the cards are worth.

With these components a search space can be searched through, usually by creating a game tree. In this tree the nodes are game states and actions connect two nodes together. The root of the tree is a node corresponding to an initial state. When a game tree is complete, the leaves of the tree are terminal states. Figure 1 is an example of a game tree for the game Tic-Tac-Toe. At the top is the starting state with no actions taken. All the 9 possible actions are represented on the second level as children of the starting node. The next levels are shown for only a few of the children of the first action. At the bottom are 3 terminal states with the outcome of the utility function.

For most games constructing a complete tree is not possible. For example, Chess has a branching factor of around 35 and games can easily reach 50 moves. This would mean a game tree would consist of $10^{154}$ nodes, which is way more than the amount of atoms estimated in the universe and therefore not possible. So to search a game tree more effectively algorithms like minimax are used to limit the search space.

Games have different properties. These properties say something about the nature of the game and the way algorithms need to be designed to solve them.

- Zero-sum: The sum of rewards is always equal and normally equal to one. For example: Chess is a zero-sum game where the winner gets 1 and the loser 0 or both $\frac{1}{2}$ when the game is drawn.
- Information: A game can be fully or partially observable to the players. In Chess both players have all the information available, but in Hearts each player can only see his own cards.
- Determinism: Whether the game is influenced by chance. Chess is a deterministic game. Hearts is not a deterministic game, as the starting hands are random. But the actions of player are deterministic.

Hearts is a partially observable, non-deterministic game. It can also be seen as zero-sum, because the total rewards each round are the same and distributed between all players.

### 2.3 Monte Carlo Tree Search

In this section we will introduce the Monte Carlo Tree Search algorithm used. MCTS is a family of algorithms from which we will use the version proposed by Kocsis and Szepesvári called Upper Confidence Bound (UCB) extended for Trees, mostly referred to as UTC (Kocsis & Szepesvári, 2006). The breakthrough of MCTS is mainly due to UTC and most research involving MCTS uses a version or extension of the UTC algorithm. To get an overview of the algorithm and a lot of the variants and enhancements see: A Survey of Monte Carlo Tree Search Method by Browne et al. (Browne et al., 2012).

MCTS builds a game tree and approximates rewards by simulating games from the leaves of the tree it is building. MCTS does not have a specific terminal criteria so it is given

a computational budget, after which it returns the best action estimated at that point. This budget is usually a set amount of iterations. Each iteration has 4 distinct steps:

- Selection: From the starting node, a child is selected through a tree policy. This policy is used to select the child best fit of exploration. The existing tree is traversed through a leaf node with this policy.
- Expansion: When the selection process halts at a leaf node, the node is expanded. One of the children defined by the actions possible in the node is added to the tree.
- Simulation: From the state of this new node, a simulation is done with a default policy. This policy defines the actions taken by all the players. The most used default policy lets all players make random actions, but other policies can be defined by simple heuristics.
- Backpropagation: The results from the simulation are updated for the new leaf node and all its parents.

This gives us an updated tree with one new node created and rewards updated for the path the tree policy took this iteration. Every new iteration starts over with the selection process starting at the root node. Because every iteration goes trough the expansion and simulation stage, with more iterations the algorithm can better estimate the rewards and search trough more of the search space, as more nodes are created and simulations are done with each iteration.

The UTC algorithm adds a tree policy to this general definition of MCTS. This policy balances the exploitation versus exploration dilemma. This dilemma is the tradeoff between exploring new strategies or exploiting already explored good performing strategies. The policy selects a child using the Upper Confidence Bound. It chooses the child with the maximum value of the following function:

$$UCB1 = \frac{Q(v')}{N(v')} + c * \sqrt{\frac{2 * ln * N(v)}{N(v')}}$$

In this formula $v$ is the node a child is selected from and $v'$ is the child the UCB value is calculated for. $Q(x)$ is the reward for a given node $x$ and $N(x)$ is the number of explorations for a given node x. $c$ is a parameter chosen beforehand. This parameter determines the amount of explorations versus the amount of exploitation. When the $c$ value is higher there will be more exploration, because the second term of the formula will be bigger. The second term of the formula has a higher value the less a child is explored compared to the other children of the node. $c$ needs to be greater than 0 and can be selected trough experiment or theoretically. Kocsis and Szepesvári showed that for rewards between 0 and 1, $c = \frac{1}{\sqrt{2}}$ satisfied the Hoeffding inequality (Kocsis & Szepesvári, 2006).

### 2.4 Determinization

To make UTC suitable for a game of imperfect information we need to extend the algorithm, since we can not create a game tree when we do not know the actions other players can take. In Hearts we do not know the hands of the other players and therefore we do not know the possible actions other players can take. One of the techniques to use MCTS in games of imperfect information is *determinization*. Determinization samples different states from the set of possible states, making different states of perfection information which can be solved by perfect information algorithms, like minimax or MCTS. Hence for Hearts we sample different possible hands for the opponents.

For each of these determinized games, we can use the standard UTC algorithm to calculate the best move possible for that game. The combination of all these results is an approximation of the best action to take for the imperfect information state. This method is also referred to as Perfect Information Monte Carlo (PIMC). The first breakthrough of PIMC was in the game of Bridge. Ginsberg created the GIB player, which became the strongest bridge AI and on par with human experts (Ginsberg, 1999). It combines the idea of determinization with domain knowledge about the bidding process of bridge to infer information about the unknown cards of the opponents. With this information the sample of possible states represents the actual state of the game better. These samples were searched with standard game tree search methods.

## 3 Method

We have implemented the UTC algorithm as described in the previous section for the perfect information variant of Hearts. The code can be found on `https://github.com/freekbax/UCTHearts`. To reduce the computation time we will use the Dutch version of Hearts, which plays with a 32 card deck. Every game ends when a player reaches 75 points. In all our tests, we played with 4 players: 2 of each algorithm we were comparing. We first tested the UTC algorithm to see which value of $c$ performs best. This parameter determines the ratio between exploration exploitation done. $c = 0.1$ performed best, so we used this value for all versions of our algorithm. This is lower than conventional values of $c$. This means our algorithm does not explore the search space as much as in other implementations. An explanation for this is that Hearts is a game with a small branching factor and our search space is also limited because we use a 32 card deck. We have analysed the influence of the computational budget on the performance of the algorithm by varying the amount of iterations the algorithm performs.

To analyse determinization, we implemented three different methods of determinizing opponents hands. A first

version where all three opponents were given random cards. This version does not take into account which cards are already played and therefore does not use all the information available. This is similar to a human player who does not remember the cards already played in the game. We will refer to this method as the Random Sample. The second version does use this information. It gives all the opponents random cards from the cards that were not played already. We will refer to this method as the Cards Played Sample. This still does not use all the information available. In the case where a player does not play the suit a trick is lead with, it can be inferred that the player does not have the suit the trick was lead with. Our third version also takes this information into account. It determinizes opponents hands with cards not played and with the suits the player has, based on which the player has played. We will refer to this method as the Suits and Cards Played Sample.

To analyse the strength of these different sampling methods, we played them against a perfect information player. The three versions all played with the same amount of determinizations of opponents hands ($d$) and iterations ($i$), $d = 25$ and $i = 250$. Earlier research showed that the amount of determinizations needed for a strong player in the card game Dou Di Zhu was around 20 and increasing the amount of determinizations after this did not have a large impact on performance (Powley et al., 2011). Increasing the amount of iterations will probably increase the performance, but we see no obvious reason this should affect any of the sampling methods more than the others. The perfect information player played with 125 iterations. To show how good a player would be when it infers opponents cards perfectly we also tested a perfect information player with 250 iterations against the player with 125 iterations. As a benchmark, we also included a player which actions are completely random within the rules of the game.

## 4 Results

Table 1 shows the performance of different versions of perfect information UTC against itself with different computational budgets. Each column is a version with an $i$ amount of iterations and played against a version with double the amount of iterations. The first row shows the amount of points the player got, the second row shows the amount of points the opponent with double the computational budget got. The third row shows the win rates of the player, the win rate of the opponent is 100% minus the win rate of the player. The sample size is a 1000 games. All results are statistically significant with over 99% confidence. Within the brackets after the amount of points is the standard deviation of the sampled points. We can see that doubling the iterations improves the performance. The opponents get about 5 points less on average. Remember that less points is better. The win rate of the player with double the amount of iterations is about 55% to 60%.

Table 2 shows the performance of the different versions of determinized UTC explained in the method section. It also displays the performance of a random player and a perfect information variant with the same amount of iterations. The opponent was a perfect information UTC player with 125 iterations. The first row shows the amount of points the player got, the second row shows the amount of points the opponent. The third row shows the win rates of the player, the win rate of the opponent is 100% minus the win rate of the player. The sample size is a 1000 games. All results are statistically significant with over 99% confidence, except the difference between the Card Played Sample and Suits and Cards Played Sample. These were statistically significant with 95% confidence. We see the performance of the determinization players improving with each improvement in the sample of opponents hands. The difference between the Random Sample and the Cards Played Sample is larger than the difference between the Cards Played Sample and the Suits and Cards Played Sample.

## 5 Discussion

The perfect information UTC algorithm performs as we expected. When the algorithm gets more computation time, the performance of the algorithm increases. When the amount of iterations is doubled, the player achieves an average score of 6 points lower and a win rate of about 60%. This is also in line with earlier research (Sturtevant, 2008). This is notable, because in Sturtevants research the algorithms were tested on the 52 card version of Hearts, while we tested on the 32 card version of hearts. So even though the search space was a lot smaller in our implementation, the influence of the amount of iterations remains the same.

For the different determinization methods we see an effect on performance. When more information is used to sample opponents hands, the performance increases. The effect is very strong between the Random Sample and the Cards Played Sample. This suggests that adding domain specific knowledge to the determinization process greatly increases the performance of the player. As an addition of cards played increases the performance of our player drastically.

For the second improvement of sampling; the Suits and Cards Played Sample, the effect is noticeable but weaker. This is because making an inference from the cards that have been played limits the possible sample a lot more than making an inference from a player not following suit. In the former case, every action reduces the sample size by one card, but players not following suit is a much rarer occurrence and may not happen at all in a given game. The performance of UTC with perfect information shows that there is still room for improvement with better inference techniques. However Hearts is a card game where information is not inferred easily. This is especially true in the beginning of the game, be-

| i | 50 | 100 | 125 | 250 | 500 |
|---|---|---|---|---|---|
| points player | 59,23(18,66) | 58,97(18,62) | 59,28(18,75) | 58,26(18,89) | 58,83(18,75) |
| points opponent | 53,29(19,14) | 53,63(19,06) | 52,77(19,23) | 54,18(19,06) | 53,53(19,23) |
| win rate player | 42,3% | 40.8% | 41.1% | 44,0% | 42,5% |

Table 1

*Performance of Perfect information UTC against UTC with double the iterations*

| | Random Sample | Cards Played Sample | Suits and Cards Played Sample | Random player | UTC250 |
|---|---|---|---|---|---|
| points player | 64,38(18,05) | 60,32 | 59,32(18,53) | 66,14(17,63) | 52,77(19,23) |
| points opponent | 38,53(17,73) | 51,86 | 53.76(19,18) | 28,60(15,11) | 59,28(18,75) |
| win rate player | 14,3% | 37,3% | **42,1**% | 3,4% | 59.1% |

Table 2

*Performance of determinization players against UTC with 125 iterations*

cause there is no bidding phase in which players reveal information about their hand, unlike other card games as Bridge or Spades. So at the start of the game there is no information available, until cards are played.

## 6 Conclusion

Our experiment shows that improvement of sample quality for determinization improves the quality of play. The better the hands of opponents can be estimated, the better the agent performs. Especially when the amount of possible hands is reduced by a lot, as shown by the difference between the Random Sample and Cards Played Sample. We do not see any clear reason why this effect should not be present in other areas. This indicates that efforts to improve sample quality will likely improve the performance of a player. The performance of the perfect information player shows that inference techniques can also improve a player significantly. For card games with a bidding phase like bridge or spades, inference can improve a player more than a card game like Hearts, because bidding reveals information about the strength of a hand. In future research, the combination of inference with determinization is a topic worth looking at, especially in games where more information can be inferred. One other interesting avenue for broadening this research is in exploring its applicability to less obviously related problems such as probabilistic planning.

The biggest limitation of our research was computation time, because determinization is an expense technique. It would be interesting to investigate the effect of better samples with more determinizations and more iterations. It is not clear yet if more samples could compensate for the quality of samples or if incorrect samples influence the algorithm as much or more with more samples. This could be done by varying the amount of determinization and looking if the effect we measured is larger or smaller for different amounts of determinizations.

Hearts is a game with a small branching factor. One of the strengths MCTS compared to other tree search methods is that it can handle a large branching factor. This is why the algorithm helped the breakthrough of AI for Go, a game with a very large branching factor. In future research determinization for games with a higher branching factor could use this strength of MCTS.

New approaches to AI for games have combined reinforcement learning techniques like MCTS with machine learning. This approach has shown promising results in both computer games and perfect information board games. This approach could also be used for board games with imperfect information. however exact applications of this are a matter of future research.

## 7 References

Bjarnason, R., Fern, A., & Tadepalli, P. (2009). Lower bounding Klondike solitaire with Monte-Carlo planning. In *Nineteenth international conference on automated planning and scheduling.*

Brown, N., & Sandholm, T. (2018). Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, *359*(6374), 418–424.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, *134*(1-2), 57–83.

Cowling, P. I., Ward, C. D., & Powley, E. J. (2012). Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(4), 241–257.

de Groot, A. D. (1946). *Het denken van den schaker: een experimenteel-psychologische studie*. Noord-Hollandsche Uitgevers Maatschappij Amsterdam.

Dosovitskiy, A., & Koltun, V. (2016). Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*.

Ensmenger, N. (2012). Is chess the drosophila of artificial intelligence? A social history of an algorithm. *Social Studies of Science*, *42*(1), 5–30.

Ginsberg, M. L. (1999). GIB: Steps toward an expert-level bridge-playing program. In *Ijcai* (pp. 584–593).

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in neural information processing systems* (pp. 3338–3346).

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European conference on machine learning* (pp. 282–293).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Powley, E. J., Whitehouse, D., & Cowling, P. I. (2011). Determinization in Monte-Carlo tree search for the card game Dou Di Zhu. *Proc. Artif. Intell. Simul. Behav*, 17–24.

Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

Shannon, C. E. (1950). Xxii. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(314), 256–275.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... others (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Sturtevant, N. (2008). An analysis of UCT in multi-player games. *ICGA Journal*, *31*(4), 195–208.

Yoon, S. W., Fern, A., & Givan, R. (2007). FF-replan: A baseline for probabilistic planning. In *Icaps* (Vol. 7, pp. 352–359).