Universiteit Leiden
The Netherlands

Opleiding Informatica

Agents for the

card game of Hearts

Joris Teunisse

Supervisors:
Walter Kosters, Jeanette de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

23/08/2017

Abstract

The aim of this bachelor's thesis is to create and compare agents for the card game of Hearts, using techniques from the field of Artificial Intelligence. After a short introduction, we explain the rules of the game. We cite some related papers, and discuss their relevance to this thesis. We will then present several handmade agents for this game, and give information about the strategies used. The performance of these agents is visualized using data collected by the program. Finally, we evaluate the performance of the agents and discuss future work.

Contents

Chapter 1

Introduction

The game of Hearts is a trick-taking card game. In this thesis, we use multiple different strategies to try to play the game as good as possible. To achieve this, we use techniques from the field of Artificial Intelligence to create several agents and compare their results.

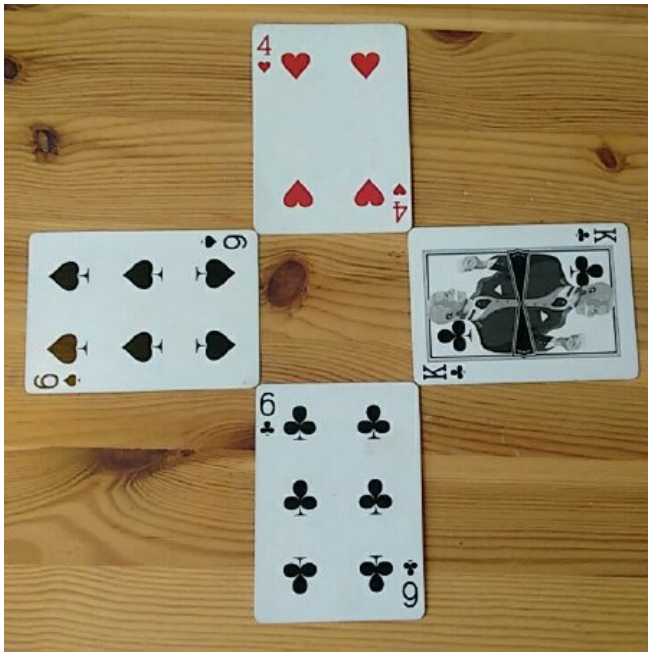

Figure 1.1: A standard trick of Hearts.

A standard game of Hearts is played with four players, using a regular deck of playing cards: 52 in total. These are divided between four *suits*: Clubs, Diamonds, Hearts, and Spades. Every suit in turn contains thirteen *ranks* ranging from two to Ace. Playing these cards in several instances called *tricks* and *rounds*, the players try to obtain as few points as possible before the end of the game is reached. These penalty points are given to the player who takes any Hearts card, or the Queen of Spades. For an example of a trick, see Figure 1.1.

In the aforementioned figure, supposing that one of the Clubs cards was the first to be played, the owner of the King of Clubs takes the trick.

To play this game, we first introduce a simple random agent and a rule-based agent. Afterwards, we discuss two types of agents using the Monte Carlo search method: one with perfect information, and one trying to simulate perfect information using determinization.

This bachelor thesis is supervised by Walter Kosters and Hendrik Jan Hoogeboom of the Leiden Institute of Advanced Computer Science (LIACS), from Leiden University. The structure of the thesis is as follows. This chapter contains the introduction; Chapter 2 explains the rules of the game; Chapter 3 discusses related work; Chapter 4 describes the agents created; Chapter 5 evaluates the performance of said agents; Chapter 6 concludes and discusses future work.

Chapter 2

Concepts and Rules

This chapter discusses the concepts and rules of the game of Hearts. The players of the game are all assumed male to avoid confusion.

2.1 Dividing the cards

The game starts by shuffling a standard deck of playing cards into a random order, and dealing the deck to all players such that every player has the same amount of cards. In this thesis, we will assume the standard values of four players and 52 cards: this amounts to thirteen cards dealt to each player. Players do not start playing the cards immediately afterwards: first, a few of the cards are passed over to one of the opponents. In a standard game, the amount of cards passed is three cards per player. Which cards to pass is not restricted in any way: however, the receiving player is determined by the number of the round. In the first round, every player passes his three cards to the next player in clockwise order. In the next round every player passes the cards to the opponent following the one they had passed to previously, and so on. As passing clockwise in this manner results in the fact that players should pass cards to themselves at some point, no cards are passed around every round divisible by the number of players.

2.2 Rounds and tricks

After the cards are dealt, the round starts. One round normally consists of thirteen tricks. At the start of the first trick, the player holding the two of Clubs is identified. This player then *leads* the first trick with this card: it is the first to be played that trick. Afterwards, the remaining players each play a card in clockwise order according to the rules explained in the next section.

When each player has played a single card, the trick concludes and a score is assigned to it. More information on how said scores are calculated will be discussed later on in this chapter. Afterwards, players continue to play cards in similar fashion until every card in the deck has been played. When every card has been played, the round concludes and the points are tallied. If the points of any player exceeds a certain value, 100 in a standard setting, the game session finishes. At this point, the player that has collected the least amount of points is declared the winner. In the case of a tie due to multiple players having the least amount of points, all players tied for first place win the game.

2.3 Rules

The rounds described above adhere to certain rules, which will now be explained:

- Players have to follow suit: they have to play a card of the leading suit if able to. If unable, players are given the opportunity to play any card.
- The Queen of Spades cannot be played on the first turn.
- A player cannot play Hearts as the first card of a trick unless Hearts has been *broken*. This means a Hearts card has been played in a previous trick. The sole exception to this rule is if the player has no suits but Hearts left to play.
- The player that takes a trick is determined by the highest rank of the leading suit. This player also leads the next trick.

2.4 Points

Every trick is assigned an amount of penalty points when it concludes. These penalty points are given to the player who took the trick. In a standard setting, every Hearts card is valued at 1 point, and the Queen of Spades is worth 13. In the event a full round is played and a single player has gathered all of the points, we speak of *Shooting the Moon*. In this case all other players receive the full round penalty, while the penalty for the player that shot the moon is discarded. This amounts to a penalty of 26 points per opponent in a regular setting.

Chapter 3

Related Work

In this chapter, we discuss related work done in this field. We expand on the contents of the papers mentioned, and state their relevance to this thesis.

3.1 Monte Carlo and Determinization

In this thesis, we use the Monte Carlo search method for one of our agents. This method randomly plays out a game state n times, and uses an evaluation function to determine the best move. Browne et al. [BPW+12] discusses that this method relies on two fundamental concepts: the ability to approximate a real solution using random simulations, and the ability to use this approximation to create a strategic approach to a given game. *Determinization* is also covered in this thesis, as a part of one of our Monte Carlo agents. This concept is discussed in [BPW+12] as well. In a designated chapter about the topic, it is argued that a game partially based on chance (such as the one discussed in this thesis) can be researched by transforming all chance-based events into concrete values. By playing several of these determinized games, a strategy can be developed for the game situation actually taking place.

3.2 Hearts

Related work has also been done with regard to the game of Hearts. For example, [LSBF10] uses Hearts among other games to evaluate the performance of Perfect Information Monte Carlo search. Also, [Stu08] discusses the performance of the UCT algorithm in Hearts, paying specific attention to preventing the opponent from shooting the moon. The same paper presents an improvement over one of Sturtevant's older players [SW06], which used linear regression and Temporal Difference learning.

Finally, research has been done on the complexity of trick-taking card games such as Hearts with regard to PSPACE-completeness [BJS13]. The main result of this paper is a proof of PSPACE-completeness for a limited group of hands through reduction for the game class examined, which Hearts is a subset of.

Chapter 4

Approach

In this chapter, we discuss several agents that we created for the game of Hearts. Agents are artificial players for a given game, each using a different strategy to approach the way to victory. By creating these for the game of Hearts in increasing order of difficulty and comparing their results, we try to create an artificial player that plays as good as possible.

4.1 Random agent

The *Random agent* does exactly what the name implies: it plays the game by choosing random cards to pass, and by choosing random cards to play every trick. The only constraint on this agent is, of course, that the cards chosen are subject to the rules of the game. All possible moves are equally likely. Needless to say, this agent performs very poorly and only exists to serve as a baseline for other agents, and also as agent substitutes for the Monte Carlo play-outs.

4.2 Rule-based agent

The *Rule-based agent* operates using several hard-coded rules, which at first glance may appear to follow an optimal strategy. However, just like the Random agent, there is no room for any emerging strategies: the rules are followed at all times. This results in the agent not being able to overcome limitations set by the simple rules by which it is guided.

The rules we have chosen for situations in which the agent does not lead the trick are as follows:

1. Play the Queen of Spades if the agent does not have to follow suit.
2. Play any Hearts card if the agent does not have to follow suit.
3. Play the highest card that still assures the agent does not take the trick.
4. Play any card that can be played.

Alternatively, if the agent does lead the trick, the lowest valid card is played. In the case of ties, the agent randomly plays one from the subset of tied cards. By following these rules, as many points as possible are avoided without making the rules too complex.
The final rule used by our rule-based agent comes into effect if the player has obtained more than a set amount of points thus far this round. When this limit is reached and no other players have any penalty points, the agent tries to obtain as many points as possible by inverting the rules. This way, a strategy is pursued to obtain the maximum amount of points possible and hopefully shoot the moon. In the case any other player does incur a penalty, the strategy is reversed once again to limit the damage.

4.3 Monte Carlo agent

The *Monte Carlo agent* uses the Monte Carlo strategy to determine the best move. For a visualization of how the algorithm works, see Figure 4.1.

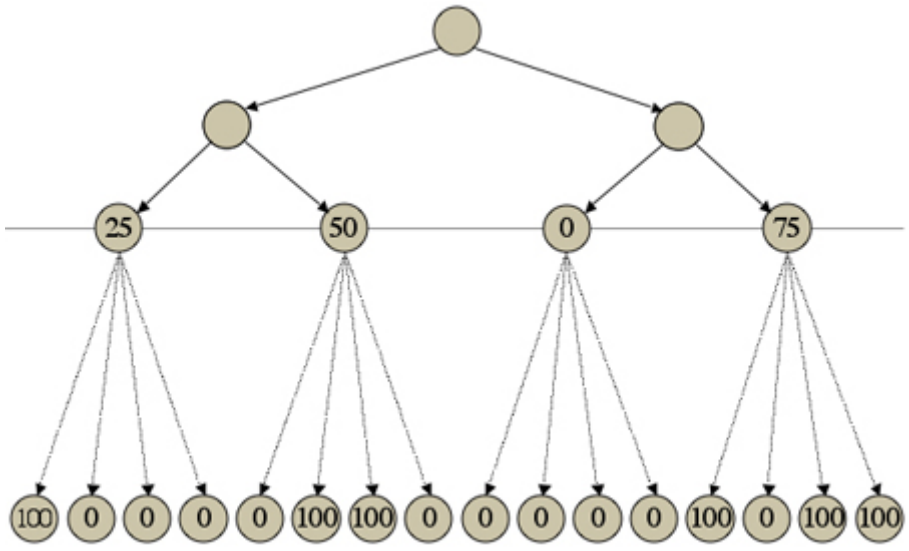

Figure 4.1: The Monte Carlo algorithm (`ggp.stanford.edu`).

The algorithm randomly plays out each valid move a set amount of times. These play-outs are given a value based on performance of the Monte Carlo player. By selecting the move with maximum performance in these play-outs, the move for the current game state is determined.

This agent is divided into two different versions: a version which uses perfect information which we call *clairvoyant*, and a version which uses determinization to extract the maximum results from the information publicly available. Both versions use the same evaluation method: the current round is randomly played out either l tricks in advance, or until the round finishes if fewer than l tricks remain. These play-outs occur m times for every move. The reason for not always fully playing out the round is linked to the performance of the agent: this will be shown in more detail in Chapter 5. The amount of points the agent possesses after the play-out is then compared to the current score of the agent, and the difference is added to the point total of that move. The move with the least average amount of points gained is determined by selecting the move with the lowest point total. If multiple moves share the lowest total, a random move from that subset is selected. Note that this strategy automatically includes shooting the moon: thus, the agent will gather points if it concludes that shooting the moon is the optimal strategy for the situation.

4.3.1 Clairvoyant

The clairvoyant agent, as stated above, uses perfect information to cheat its way to victory. By using the real distribution of cards for the random play-outs, the agent decides on the strategy that is as good as possible for the situation actually taking place. While not feasible to be used for actual game sessions due to accessing forbidden information, this agent serves as skill measurement for the agent using determinization: as determinization is trying to mimic perfect information as good as possible, it is worthwhile to compare with the strength of an agent using the real values.

4.3.2 Determinization

The other Monte Carlo agent uses determinization to avoid having to cheat with perfect information. First of all, all publicly available information about the card distribution is gathered. This includes the cards the agent has passed at the start of the round: the agent knows exactly who holds these individual cards. Also, there is information to be gained in the case another player cannot follow suit: this means no cards of the current leading suit are held by that player.
After obtaining this information, a subset of the unknown cards is distributed preliminarily. This subset includes all cards only one player can own, which can be derived in multiple ways. First, if all other players do not own the suit of the card that is to be distributed, the card is assigned to the single player that can be the owner by process of elimination. Also, if a single player can only have one possible hand, this hand is distributed to the player. Afterwards, the cards still unknown are randomly distributed between the players: ideally such that every distribution is equally likely to take place. However, we have chosen a slightly different approach: the reasoning behind this is explained in the next chapter.
Also, there is a small chance the cards will be distributed incorrectly: in this case, the distribution is reset and the algorithm of the agent continues until a permutation is found that adheres correctly to suit ownership. These distributions are then used in the play-outs to approximate the best possible strategy for the actual situation.

Chapter 5

Evaluation

In this chapter, we conduct several experiments and analyse the results thereof. Multiple types of experiments will be done, including analysis of single agents by tuning parameters and combined analysis of multiple agents by comparing different types against each other. These experiments will be conducted by using the Hearts program created in this thesis. The agents are always sorted in the same manner (if applicable to the table): the first column represents the first agent playing the game, and next columns represent the other agents in clockwise order. Also, in all figures n will denote the amount of game sessions the data was extracted from. This value will differ based on the complexity of the problem at hand: all tests were run on the same machine using an Intel Core i5 3570 processor in tests taking a few hours to a day to complete.

5.1 Rule-based agent

First, we will look at the performance of the rule-based (RB) agent. Although the random (RD) agent might seem like the logical start to these experiments, it delivers next to nothing in terms of information on its own. To start off, we will test the RB agent by tuning several parameters to ascertain that we are working with the best possible version in the later experiments.

5.1.1 Parameter tuning

In this section, we look at game sessions featuring RB agents with differently tuned parameters and compare their results. The parameter that is to be tuned is the threshold before the strategy is flipped to shoot the moon, which will be tested in the following section.

Tuning the threshold

The threshold ranges from zero (always shoot the moon) to 26 (never shoot the moon). We will now decide on the optimal threshold to flip the strategy by using self-play against three RB agents that do not aim to shoot the moon at all: a threshold of 26. This test is conducted for every other threshold possible. The results of these experiments are shown in Figure 5.1.

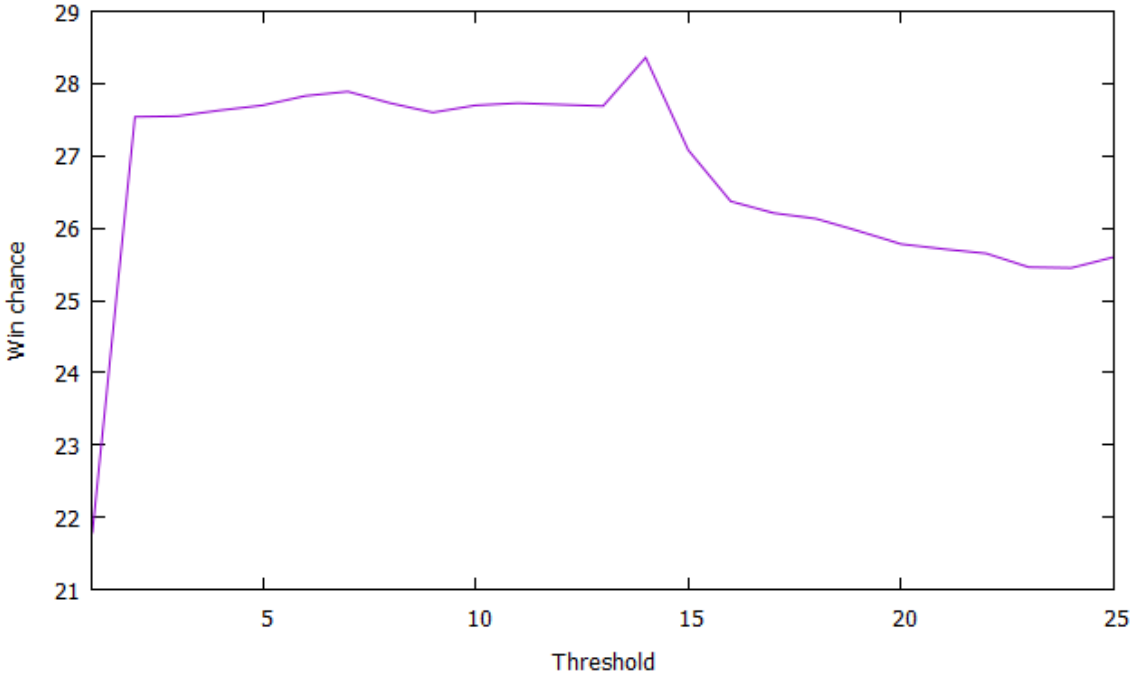

Figure 5.1: Win chance of a threshold RB agent against those without ($n = 200000$).

Studying this figure, it is clear that aiming to shoot the moon leads to an improvement for every threshold above one: the agent examined has a win chance well above 25%. Also, it is worth noting that the performance spikes when shooting the moon at fourteen points. A possible explanation is as follows: at fourteen points the agent has definitely obtained the Queen of Spades, therefore not having too much extra to lose if shooting the moon fails (due to another agent taking a Hearts card). Also, Hearts has been broken already: the agent has obtained one Hearts card thus far, since the only combination for fourteen penalty points is the Queen of Spades and a single Hearts card. This opens up more options to play according to a point-gathering strategy, since owned high Hearts cards can be used on a leading turn to gather large amounts of points. Taking these results into account, we will conduct further tests using a threshold of fourteen.

5.1.2 Performance against others

To start off, our first test will simply consist of putting up the RB agent against three RD agents: the results are shown in Table 5.1.

	RB	RD 1	RD 2	RD 3
Win %	73.28	10.35	9.38	8.39
Average points	42.63	79.17	80.75	82.74

Table 5.1: Performance of a RB agent against three RD agents ($n = 1000000$).

This table presents a few interesting observations. First of all, note that the combined percentages of games won exceeds 100%. This is the case because there are games in which multiple players win the game due to having a shared amount of points (see Section 2.2).
A far more interesting development is that the RD agents differ in their behaviour based on their relative position to the RB agent: the percentage of games won decreases and the average of points gained increases. This seems to hint at a relation between the relative position between two agents with different strategies and their performance. Continuing that train of thought, we tested game sessions with three RB agents against one RD agent, expecting some sort of relation between position and performance: see Table 5.2.

	RB 1	RB 2	RB 3	RD
Win %	31.58	32.83	35.71	2.05
Average points	55.27	54.53	52.93	102.31

Table 5.2: Performance of three RB agents against a RD agent ($n = 1000000$).

Again, we see a correlation between the position of the agents and their performance. This time, the RB agents increase in effectiveness the further they are from the RD agent. The average points follow the same trend. A possible explanation for this behaviour is the difference in position is strategically important. If a RB agent plays a card after a RD agent, the card already played by the RD agent has little value: it has, after all, been randomly selected. However, if a RB agent plays a card after the other RB agents, the strategic importance might be much greater. After all, the card to play has been selected according to a selection procedure and therefore reveals some inside information about the agent in question. Also, this advantage greatly outweighs revealing the own strategy to the RD agent that plays the next card: it is randomly selected regardless, and therefore there is no immediate punishment from the next agent for the card that has just been played.

5.2 Monte Carlo agents

In this section, we evaluate the performance of the Monte Carlo agents created. As has been mentioned before, there are two variants: one with perfect information which we call clairvoyant (CV), and one trying to approach a perfect information state using determinization (DT).

5.2.1 Tuning the amount of play-outs

As with the rule-based agent, we will first conduct a test to determine the parameters to use for further experiments. In the case of the Monte Carlo agents, this means finding a number of play-outs that is representative enough for the performance of the agent and fast enough to realistically perform enough tests to achieve statistically significant results. To this end, we have compared multiple sessions in which we put CV agents with more play-outs against CV agents with less play-outs: this gives an insight in the relative performance of the play-outs, and therefore can be used to determine the difference in performance. As the performance increases with the amount of play-outs by default, it is enough to simply test increasing versions rather than all possible combinations. To ensure the previously encountered performance difference between positions does not skew results, the agents were tested in pairs playing on opposite sides. This way, the distance to or from a player with a different skill level is the same across the playing field, as if playing one on one. The results of these experiments can be found in Table 5.3.

	Pair 1			Pair 2		
Test #	Play-outs	Win %	Avg. pts.	Play-outs	Win %	Avg. pts.
1	5	79.47	59.77	1	21.55	84.52
2	10	61.19	70.01	5	40.58	78.86
3	20	58.41	71.59	10	43.41	77.92
4	30	54.10	73.25	20	47.72	76.09
5	40	53.06	74.00	30	48.86	75.55
6	50	52.52	74.12	40	49.23	75.57
7	60	52.55	74.17	50	49.54	75.30

Table 5.3: Performance of CV agent pairs ($n = 10000$).

These results lead to a fairly smooth closing of the gap in win percentage, with the last tests only differing a few percentage points in performance compared to the "lesser" version. Even though the performance seems to scale infinitely with the amount of play-outs, we will maintain a value of 50 play-outs based on these results. By using this value, we retain the speed of the program without much performance loss compared to versions with (many) more play-outs.

5.2.2 Tuning the look-ahead

Another parameter that needs to be tuned is the look-ahead of the Monte Carlo agents. As stated in Section 4.3, the Monte Carlo agents look a maximum of m tricks ahead to determine the best card for that situation. This also needs to be tuned: we need to discover whether any value from one to thirteen (the maximum amount of tricks) is the best number of tricks to look ahead. As with the tuning of the threshold in Section 5.1.1, we test the look-ahead against agents that do not look ahead at all: they play out the full round for every available move. Also, as with the previous tuning, this experiment was conducted using just CV agents as they are faster and are structured the same way as the DT agents. The results of this experiment can be seen in Figure 5.2.

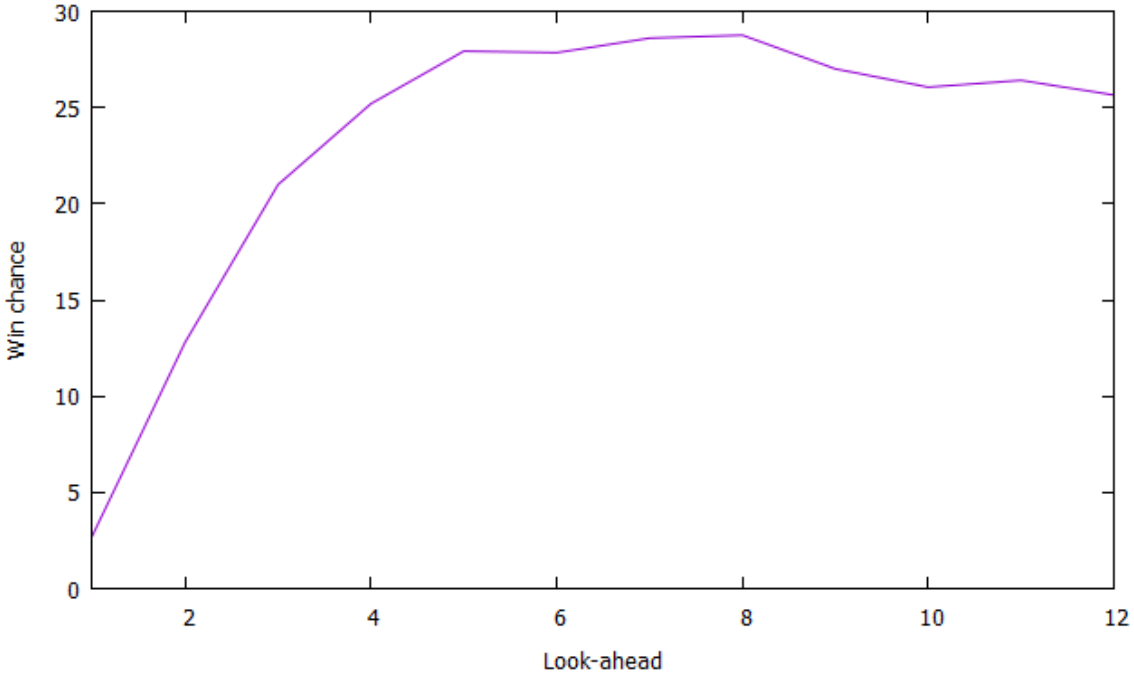

Figure 5.2: Win chance of a look-ahead CV agent against those without ($n = 10000$).

As the sample size was quite small due to the time needed for each experiment, there seemed to be some inconsistencies with the graph regarding finding the best value. We expected a parabola with a clear peak, but the win chance for both a look-ahead of six and ten did not match this expectation. Therefore, we decided to run a test with the average amount of points gained to obtain more closure: see Figure 5.3.

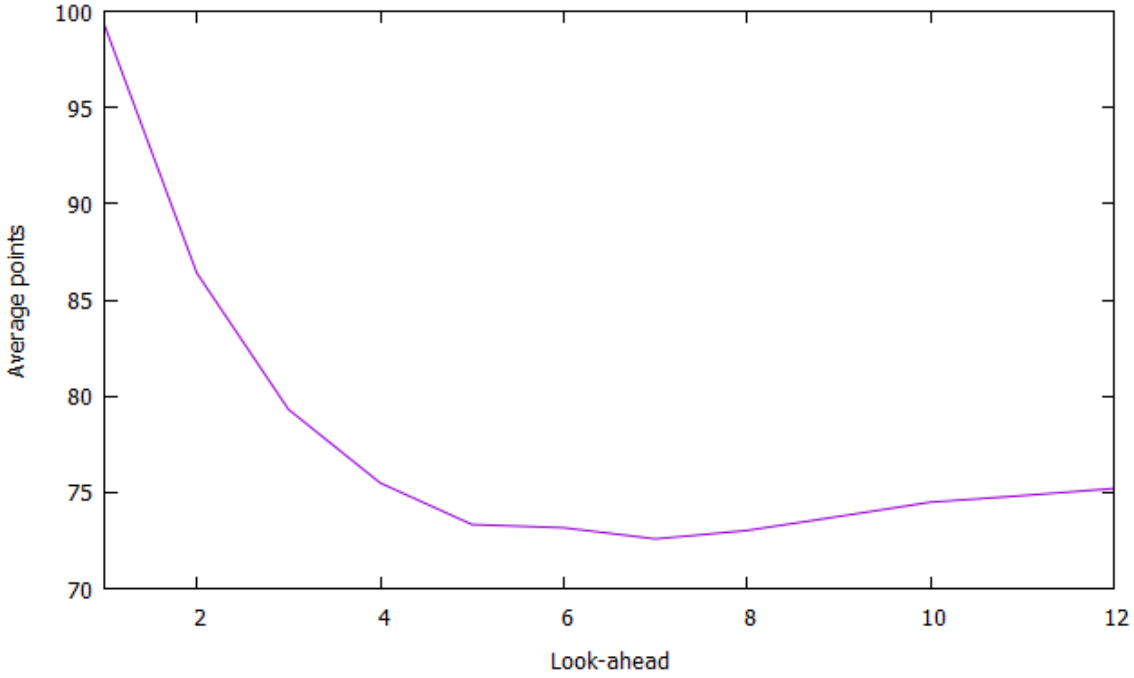

Figure 5.3: Average points of a look-ahead CV agent against those without ($n = 10000$).

Looking at both figures side by side, we determined that a look-ahead value of seven was the best value to use for further experiments: it represents the lowest average amount of penalty points and is very close to the best in terms of win chance. As win chance is more sensitive to variance than the average amount of points, it is likely that this value results in similar to better performance than the peak of the win chance graph.

5.2.3 Determinization method

Determinization, as stated before, is a method to determine the best strategy for a game with imperfect information. This is accomplished by using a possible distribution of the cards that are left to play for the Monte Carlo play-outs, adhering to the information currently known about card possession of the other players. In our program, we use an algorithm that is explained in Section 4.3.2. As stated, our version of the algorithm is not ideal: it favours certain possibilities over others and is therefore skewed. Even so, we will show that the decrease in performance is negligible compared to the increase in speed in the following case study.

Case study: Determinization algorithm

In this case study, we look at a specific case used to test our determinization algorithm that determines what distribution is to be used. To this end, we fully worked out a single situation from a subset that the algorithm had trouble with, and compared the performance with the ideal algorithm that has no bias at all. The situation in question is abstractly depicted in Figure 5.4.

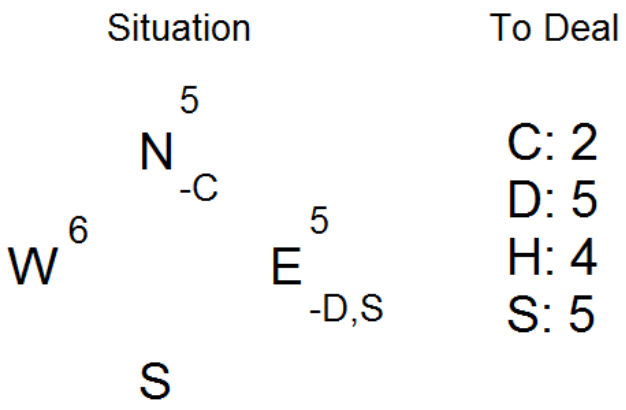

Figure 5.4: The situation in question.

This figure shows the four players, abbreviated as the four cardinal directions to clarify their position at the table. S is the agent for which determinization is to take place. The superscript numbers is the amount of cards left to deal to that player, while the subscript letters are the suits not owned by that player. On the right side are the amounts of the cards that still need to be dealt by suit: the ranks of the cards are irrelevant in this matter.

In this situation, there are $\frac{16!}{6!5!5!} = 2018016$ possible ways to deal the cards to the players. However, only 2352 distributions are actually possible given the restrictions. This value is an addition of three separate situations: those in which agent W has one of the Hearts cards, those in which agent W has one of the Clubs cards and those in which agent N has one of the Hearts cards. All other situations are impossible, due to the fact that agent E must have a combined total of five Clubs and Hearts cards. The situation in which agent W has one of the Hearts cards occurs $\binom{10}{5} * 4 = 1008$ times, denoting the amount of possible distributions of 1 Hearts card and 5 out of 10 combined Diamonds and Spades. Using the same type of calculations, the other situations occur $\binom{10}{5} * 2 = 504$ and $\binom{10}{4} * 4 = 840$ times, totalling the aforementioned 2352 distributions.

In an ideal scenario, all 2352 situations are equally likely to take place. This can be accomplished using an algorithm that randomly distributes all cards that are still left to be distributed, and thereafter evaluating the distribution according to known information (in this case, which agent owns what suits). If the distribution does not match the restrictions, it is discarded: otherwise, it is accepted. Thus, there is exactly a $\frac{1}{2532}$ chance of every situation occurring. Our algorithm, however, does not follow such a distribution. Instead, we use an algorithm that refuses to deal a card if the suit is not owned by the player that it is dealt to. The next card to be dealt is then tried for that player, again refusing to deal if the player does not own the suit. If a suitable card is found, it is dealt to the player. The initially refused card is then placed back in the deck at the former location of the suitable card. In the case a player cannot own every single card that is left to be dealt, the distribution is discarded: otherwise, the algorithm finishes and accepts.

As a result of this different type of distributing the unknown cards, there emerged a bias towards the situation in which agent W is given a Clubs card. This bias is equally likely for every such case, and amounts to an increase in occurrence by 13,2 percentage points. This increase naturally leads to a decrease in the other cases: these amount to 5,3 percentage points and 1,5 percentage points in respective order.

However, there is a benefit to using our algorithm: the speed. While the ideal algorithm was slightly better in terms of equal distributions, our algorithm improved the speed of solving this case 66.8 times. It is also debatable how much the performance loss actually is: using small-scale tests that compared the two agent types in pairs as in the experiments before, we found that there was little significant difference in performance. Since we decided that the speed increase greatly outweighed the inaccuracies of the algorithm, we decided it was a better choice.

5.2.4 Performance determinization

To test the importance of determinization for the DT agent, we decided to test it against an identical agent that simply relied on a random distribution of the available cards. Once again, this was tested with 50 play-outs, in pairs with $n = 10000$. This turned out to be a 10.84 percentage points win increase for the determinization pair compared to the random pair. This value represents just the increase in performance by determinization alone: the other agent has access to all other functionalities such as determining the best card to play, and which cards are in the game. Though it was lower than we had expected, it shows that there is a significant difference in using determinization.

5.2.5 Performance in pair duels

After tuning all parameters and testing determinization, this final experiment section consists of the performance of DT agents against both the RB and CV agents. Once again, this was done in pairs to avoid the aforementioned bias regarding relative positions. First, we tested the win chance against RB agents: for the results, see Table 5.4.

	DT pair	RB pair
Win %	72.77	29.14
Average points	64.87	83.57

Table 5.4: Performance of a DT agent pair against a RB agent pair ($n = 10000$).

As we expected, the performance of Monte Carlo search with determinization outclassed the rule-based agents by quite a bit. Although the rule-based agent seems to use logical rules, it was limited greatly by not being able to use any emerging strategies.

We also tested the DT agents against the CV agent, using their function as a skill ceiling to determine the performance of the DT agents: see Table 5.5.

	DT pair	CV pair
Win %	29.73	72.18
Average points	82.87	65.05

Table 5.5: Performance of a DT agent pair against a CV agent pair ($n = 10000$).

While effective against the RB agents, this table states clearly that there is much to be improved upon: the added value of perfect information still stands firmly against the threat posed by the DT agents. However, it is debatable how much more the DT agents can accomplish: while quite some information can be gathered by using publicly available information, there is still a great amount of cards that is unknown even by using predictive strategies.

Chapter 6

Conclusion and Future Work

In this chapter, we make several conclusions based on the experiments in the previous evaluation chapter and discuss the results. Also, we look back on the project as a whole and discuss future improvements to the program and agents created. Finally, we expand on future work regarding this subject.

6.1 Conclusion

We have discussed three different types of agents for the game of Hearts in this thesis. To this end, we created a framework as a testing environment. While the random player functioned as expected, we have found interesting results regarding the importance of position when putting it up against the rule-based agent. Also, we have found the importance of the Queen of Spades regarding a threshold for shooting the moon when adhering to simple rules. Furthermore, we determined that the amount of play-outs for a basic Monte Carlo agent diminishes in value when increased for this game. Also, we presented a determinization algorithm faster than an ideal algorithm at a small performance cost. Lastly, we proved the effectiveness of determinization for Hearts, and tested its effectiveness against multiple types of agents.

6.2 Discussion

Regarding the program, there is much to be improved upon if the opportunity arises for future work on this subject. While the framework could be used as is, the rule-based player could be updated by adding new rules or changing them for the better. Also, the determinization algorithm could be further optimized to adhere to an ideal distribution while still improving program speed.

6.3 Future work

There is much future work to be done on this subject. First of all, there is much to be gained in formulating a strategy for passing the cards around: this was not a subject in this thesis. Also, one could use Monte Carlo Tree Search [BPW$^+$12] instead of a basic version to further improve performance of Monte Carlo players. Furthermore, agents using different techniques from the field of Artificial Intelligence could be added and evaluated for the game of Hearts. An interesting example would be an agent using neural networks: while this originally was to become the topic of this thesis, it can now provide a new challenge.

Bibliography

[BJS13] É. Bonnet, F. Jamain, and A. Saffidine. On the complexity of trick-taking card games. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 482–488, 2013.

[BPW+12] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

[LSBF10] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak. Understanding the success of perfect information Monte Carlo sampling in game tree search. In *Proceedings of the 24th Conference of the Association for Advancement of Artificial Intelligence*, pages 134–140, 2010.

[Stu08] N. R. Sturtevant. An analysis of UCT in multi-player games. In *Proceedings of the 6th International Conference on Computers and Games, LNCS 5131*, pages 37–49, 2008.

[SW06] N. R. Sturtevant and A. M. White. Feature construction for reinforcement learning in Hearts. In *Proceedings of the 5th International Conference on Computers and Games, LCNS 4630*, pages 122–134, 2006.