Albert-Ludwigs-Universität Freiburg

Institut für Informatik
Arbeitsgruppe Grundlagen der künstlichen Intelligenz

Implementation of the UCT Algorithm for Doppelkopf

Silvan Sievers

Freiburg, April 10, 2012
(revised April 24, 2012)

Supervisors:

Prof. Dr. Malte Helmert
Universität Basel
Departement Mathematik
und Informatik
Artificial Intelligence

Prof. Dr. Bernhard Nebel
Universität Freiburg
Institut für Informatik
Arbeitsgruppe Grundlagen
der künstlichen Intelligenz

Abstract

Doppelkopf is a German trick-taking card game with imperfect information. Its most interesting aspect certainly is the fact that teams are not fixed over a session of games and that teams are even not known at the beginning of most of the games, but only determined or revealed to the players during the course of playing. It further has a huge state space and a large strategic depth, making it infeasible to be solved by exhaustive search. UCT is an algorithm based on Monte Carlo tree search with an improved action selection policy. It has shown to be successful when applied to other (card) games.

This work's contribution consists of the implementation of a doppelkopf player using the UCT algorithm. Furthermore, an algorithm that generates consistent card assignments is presented. It can be used by the UCT algorithm to generate samples that complete the missing information about the other players. Experimental results show that the resulting UCT player attains a decent playing strength compared to a baseline approach.

Zusammenfassung

Doppelkopf ist ein Stichspiel mit unvollständiger Information. Der interessanteste Aspekt des Spiels ist mit Sicherheit die Tatsache, dass Teams im Laufe einer Doppelkopfsitzung immer wieder wechseln und den Spielern meistens sogar nicht einmal zu Spielbeginn bekannt sind, sondern erst während des Spiels festgelegt bzw. offenbart werden. Weiterhin hat das Spiel einen sehr großen Zustandsraum und eine große strategische Tiefe, weshalb es ungmölich ist, das Spiel durch vollständige Suche zu lösen. UCT ist ein auf Monte Carlo Suche basierender Algorithmus, welcher eine verbesserte Srategie zur Aktionswahl verwendet. Der Algorithmus wurde bereits mit Erfolg auf andere (Karten-) Spiele angewendet.

Diese Arbeit stellt die Implementierung eines Doppelkopfspielers vor, welcher den UCT Algorithmus verwendet. Außerdem wird ein Algorithmus zur Erzeugung von konsistenten Kartenzuteilungen präsentiert. Dieser kann vom UCT Algorithmus verwendet werden, um Samples zu erzeugen, welche die fehlenden Informationen über die anderen Spieler vervollständigen. Experimentelle Ergebnisse zeigen, dass der resultierende UCT-Spieler eine ordentliche Spielstärke im Vergleich zu einem Basisansatz erreicht.

Acknowledgements

In the first place, I want to thank my mentor Prof. Dr. Malte Helmert for enabling me to write this thesis about one of my favorite card games. He always found time to help me a lot by giving feedback both for the implementation and the written part. His pool of ideas and hints on how to improve things seems to be inexhaustible. I'd further like to thank Prof. Dr. Bernhard Nebel for accepting to be the second auditor of my thesis. This work was proofread by Lam Tran and Marius Greitschus and I want to thank both of them a lot for taking the time to search for spelling and grammar mistakes. Last but not least I want to thank my friends for their understanding especially during the two weeks before the submission deadline.

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, date	Signature

Contents

1 Introduction **1**

2 Doppelkopf **3**
2.1 History . . . 3
2.2 Rules . . . 3
2.2.1 Preliminaries . . . 4
2.2.2 Game types . . . 5
2.2.3 Doppelkopf session and compulsory solos . . . 6
2.2.4 Game type determination . . . 8
2.2.5 Card play . . . 9
2.2.6 Announcements . . . 9
2.2.7 Game evaluation . . . 11
2.3 Doppelkopf and AI games research . . . 14

3 UCT **16**
3.1 Monte Carlo tree search methods . . . 16
3.2 Multi-armed bandit problems and UCB1 . . . 17
3.3 The UCT algorithm . . . 19

4 Card Assignment Problem **22**
4.1 Consistency of card assignments . . . 22
4.2 Related problems . . . 23
4.3 Card assignment algorithm . . . 24
4.4 Analysis of the card assignment algorithm . . . 28

5 Implementation **30**
5.1 Doppelkopf framework . . . 30
5.1.1 General design and structure . . . 31
5.1.2 Game module and game process . . . 31
5.2 UCT algorithm . . . 34
5.3 Summary of program options . . . 37

6 Experiments **41**
6.1 Tuning parameters to obtain a good baseline UCT player . . . 42
6.1.1 First announcement style version . . . 43
6.1.2 Second announcement style version . . . 50
6.1.3 Chosen baseline configuration . . . 57

6.2 Comparing the two UCT versions 59
6.3 Testing single player options 61
6.3.1 Using team points as a bias for the UCT rewards 61
6.3.2 Changing the announcement rule 62
6.3.3 Using the correct UCT formula 64
6.3.4 Not using an MC simulation 66
6.3.5 Changing the action selection version 67
6.4 Combining several player options 70
6.4.1 Comparing action selection versions with no MC simulation . . . 71
6.4.2 Combining good options for the first UCT version 73
6.4.3 Combining good options for the second UCT version 75
6.4.4 Chosen best configurations for both UCT versions 78
6.5 Increasing the number of rollouts and the number of games 81
6.6 Human player against best UCT player 84

7 Conclusion **88**
7.1 Summary 88
7.2 Outlook 89

Bibliography **90**

1 Introduction

Playing all kind of different games has always been an amusement for mankind and thus with the arising knowledge in computer science and artificial intelligence, games have also become the focus of research. A primary goal often consists in analyzing games with respect to the question if they can be "solved" or not, where solving means that a program is able to perfectly play the game in the sense that it has a best response to all possible moves an opponent makes. Solving is not possible in many cases because the game is too complex and the state space is too large. Thus, many approaches of game research strive to find good approximations for playing a game. Such algorithms are commonly based on heuristic search methods or other tools that allow to restrict the problem's size or to abstract the original problem to a smaller version which can then be solved and the solution can be used to generate an approximate solution for the original problem.

This thesis investigates the German card game doppelkopf. Doppelkopf is a trick-taking game of incomplete information for four players, played with 48 cards. The main "feature" of doppelkopf is that there are no fixed teams, every player tries to collect as many points as possible over a session of games. During most of the games, there are two teams consisting of two players each, but the teams are usually not known in advance! This enables the strategic depth of the game, being a challenge even for human players. Additionally, the game play can be separated into a game type determination phase and an actual card play phase, during which also announcements (or bids) can be made. The game thus shows some similarities to the well studied games of bridge and skat, but has not been matter of games research so far, to my best knowledge.

As doppelkopf also has a large state space, it is unfeasible to solve it by complete exploration of the game tree. This is the reason and the motivation for this thesis to investigate the application of *Upper Confidence Bounds applied to Trees* (UCT) [12] to doppelkopf. The UCT algorithm was developed in 2006 and since then had a growing success in the application for other games like go [10, 9], skat [17] and Klondike solitaire [3]. It uses classic Monte Carlo tree search combined with sampling of worlds for the estimation of values of game moves. The main goal is to show that UCT can be used for a game with incomplete information like doppelkopf and further to show that this approach yields reasonable results when comparing it to a baseline approach, which is the only satisfying comparison method due to the lack of previous research. Furthermore, this thesis also describes several possible extensions that slightly modify the original UCT algorithm in order to test if they can improve the resulting player further.

In order to be able to apply the UCT algorithm to a game of incomplete information, either this missing information must be provided to the UCT algorithm or the UCT algorithm can directly be operated on the belief state space rather than on the abstract

state space. The choice was made in favor of the first approach and thus a way of sampling complete information worlds for the UCT algorithm needs to be found. The solution proposed by this thesis is an algorithm that generates consistent and mostly uniformly random card assignments that can be used as samples for the UCT algorithm.

The outline of this work is as follows: this introduction being the first chapter, the second chapter is dedicated to the introduction of the game doppelkopf with all the rules and background information necessary for the further understanding. The follow-up chapter describes Monte Carlo tree search methods and based on these, the original version of the UCT algorithm is explained. The fourth chapter presents the card assignment algorithm and analyzes the degree of randomness of the generated card assignments. In chapter five, the program implemented for this thesis is presented. This includes the description of the implementation of doppelkopf itself, the explanations of the two implemented different versions of the UCT algorithm and the presentation of the different options that can be used to modify certain characteristics of the UCT algorithm. The sixth chapter finally presents a lot of experiments done with the program written for this thesis. First, different parameter configurations are tested with the goal of finding a suitable baseline player which is then used for comparison experiments including other configurations that use one or more of the different options possibly enhancing the UCT algorithm. The final best UCT version found is tested in playing against a human player. The final chapter draws a conclusion of the thesis and gives an outlook for future work.

2 Doppelkopf

This chapter introduces the game of *Doppelkopf*, a card game well known in Germany but not so much in other regions of the world. There will be a very brief overview of the history and the origins of doppelkopf, followed by a explanation of the game rules and concluding with a motivation of why doppelkopf is of interest for games research and how the game has to be classified.

2.1 History

Doppelkopf, often abbreviated "Doko", literally means "double-head" when translated into English. There is not much known about the history of doppelkopf, but it is generally assumed that it has originated from the game *Schafkopf* (literally *sheeps-head*), which is a very popular game in Bavaria and which has "official" rules since 1895 [19]. At least, doppelkopf is a trick-taking card game like schafkopf and it is played with a double deck of schafkopf cards, hence explaining the name doppelkopf: double (schaf)kopf. Furthermore, it has many similarities with the card game *Skat*, which also probably originated from schafkopf.

Doppelkopf is mostly common in the northern regions of Germany, where it is nearly as popular as skat, but less known outside of Germany. In Bavaria, schafkopf is still the prevailing variant. One "problem" with doppelkopf is that there are regional differences with many variants of the game rules; players that do not know each other often need to discuss a lot in order to agree on a set of rules before they can actually start playing. One of the main goals of the Deutscher Doppelkopf Verband[1] (DDV, German Doppelkopf Association), which was founded on March 27, 1982, is to settle some "official" rules. It also organizes national championships, maintains a league ("Bundesliga") and publishes a ranking of players.

2.2 Rules

As stated above, doppelkopf exists in numerous variations, differing from region to region. For the rest of this thesis, I will stick to the official rules designed by the DDV – the game is already sufficiently complicated without adding and allowing all special rules that may exist. The rules can be found on the site of the DDV[2] (only in German) or, in a less formal way, but still described correctly, on the English Wikipedia article

[1] http://www.doko-verband.de
[2] http://www.doko-verband.de/download/turnierspielregeln.pdf

about doppelkopf.[3] Some of the examples below are taken (and sometimes modified) from the Wikipedia article. When searching the Internet for more English information about doppelkopf, one may find several websites, but many of them do not explain the official rules correctly (for gaining a general understanding of the game, this may be sufficient, though).

2.2.1 Preliminaries

Doppelkopf is a *trick-taking card game* for four players. One game of doppelkopf always consists of two teams, the *Re*-team (short: re, "a player is re") and the *Kontra*-team. One of the main features of the game is the fact that the actual teams are most of the time not known in advance but only determined during the game, which makes it very interesting from a strategic point of view. Normally, the goal for re is to achieve at least 121 points; kontra wins when re fails to do so.

Doppelkopf is played with a double (shortened French) deck of cards, each consisting of 24 cards divided into the four suits clubs (♣), spades (♠), hearts (♡) and diamonds (♢). Each suit contains the following cards: ace (A), ten (10), king (K), queen (Q), jack (J) and nine (9). In the future, a specific card will be described by a symbol for its suit followed by the abbreviation for its name, e.g. ♡K stands for the king of hearts. Card values are depicted in Table 2.1, thus totaling 240 points for the whole deck of 48 cards.

Card	Value
Ace	11
Ten	10
King	4
Queen	3
Jack	2
Nine	0

Table 2.1: Card values

One of the players starts being the dealer and deals each player twelve cards (starting with the player positioned on the left to him, dealing each player three cards in a clockwise order until all cards are dealt) face down. Players are only allowed to view their own cards (the so-called *hand*). They are not allowed to show their cards or to communicate any information about their hands to the other players except by the means intended by the game rules, such as announcing (see later). The player left of the dealer starts the game (with some exceptions explained later), normally by playing a card, face up, so that all players can see it. Card play will be described more in detail later (Subsection 2.2.5). For the next game, the player who started will be the dealer and so on. There is no limit in how many *rounds* (a round consists of four games, i.e. each player is dealer once) are played, but in an official tournament, 24 games will be

[3] http://en.wikipedia.org/wiki/doppelkopf

played and each player is dealer six times (and thus also starts playing six times, which is important because it generally signifies an advantage for that player).

2.2.2 Game types

There are different game types. For each game, a type needs to be determined. How game type determination works is described in the next subsection. The different game types mainly vary trump and non-trump suits and sometimes also the playing teams may be pre-determined or determined in another than the usual way.

The default game type is the *normal* or *regular game*, in which the players holding a ♣Q, called "Die Alten" (*the elders*), constitute the re-team and the other two players are the kontra-party (assuming for now that no player holds both ♣Q). As players do not know the other players' cards, they also do not know which player is part of which team as long as not both ♣Q have been played (or announcements are made, see later). In a regular game, the ♡10, all queens, jacks and the remaining cards from the ♢-suit form the trump suit; all other cards form the three non-trump suits. An overview including the ranks of the cards is depicted in Table 2.2.

Trumps in descending order		
♡10, ♣Q, ♠Q, ♡Q, ♢Q, ♣J, ♠J, ♡J, ♢J, ♢A, ♢10, ♢K, ♢9		
Non-trumps in descending order per suit		
Clubs	Spades	Hearts
♣A, ♣10, ♣K, ♣9	♠A, ♠10, ♠K, ♠9	♡A, ♡K, ♡9

Table 2.2: Suits in a regular game

A very similar game type is the so-called "Hochzeit" (*marriage*), where one player has both ♣Q. In this case he[4] may play alone without announcing anything (resulting in a "stilles Solo", translated into *silent solo*, which is the same as a diamonds solo, see later) or he announces a marriage (unless another player wants to play a solo which then would have higher priority, see later) and plays with the first other player to win a trick. If the marriage player wins the first three tricks himself, he is forced to play a diamonds solo. The marriage player himself is always part of the re-team, and depending on if he marries another player, this other player joins him, otherwise the other three players form the kontra-team. This means that the teams are determined and known by all players at the latest after the three first tricks have been completed. Most important for now is that the trump suits in a marriage game are exactly the same as in a regular game.

All remaining types of game are *solo games*. In any solo game, the re-team is only constituted by the solo player, the other three players are kontra. The first category

[4] A player naturally may be female or male; to avoid the trouble of writing he/she, his/her/their always, I'll state right now that whenever I write about a player and use a male pronoun, I also want to include female players.

of solo games is a *color* or *trump solo*, where the only difference to the trumps of the regular game is that the diamonds suits (possibly) gets substituted by another suit of the solo player's choice. Note that the hearts tens, the queens and the jacks will always remain the top trumps, just the lowest cards (i.e. diamonds ace to nine) get replaced by their equivalent of the other suit. Consequently in a diamonds solo, the trump and non-trump suits are exactly the same as in a regular game. Due to the hearts tens being always the highest trumps, there are two trump cards less in a hearts solo (because only aces, kings and nines of hearts join the trump suit). Table 2.3 depicts trump and non-trump suits in a hearts solo.

Trumps in descending order		
♡10, ♣Q, ♠Q, ♡Q, ♢Q, ♣J, ♠J, ♡J, ♢J, ♡A, ♡K, ♡9		
Non-trumps in descending order per suit		
Clubs	Spades	Diamonds
♣A, ♣10, ♣K, ♣9	♠A, ♠10, ♠K, ♠9	♢A, ♢10, ♢K, ♢9

Table 2.3: Suits in a hearts solo

The next category of solo games are the *queens* and *jacks solos*. In these games, only the queens (or the jacks respectively) form the trump suit (in the same order from clubs to diamonds as always) and each "normal" suit forms its own suit, where the jacks (or the queens) are ordered below the kings and above the nines. Table 2.4 exemplarily shows the suits for a queens solo.

Trumps in descending order			
♣Q, ♠Q, ♡Q, ♢Q			
Non-trumps in descending order per suit			
Clubs	Spades	Hearts	Diamonds
♣A, ♣10, ♣K, ♣J, ♣9	♠A, ♠10, ♠K, ♠J, ♠9	♡A, ♡10, ♡K, ♡J, ♡9	♢A, ♢10, ♢K, ♢J, ♢9

Table 2.4: Suits in a queens solo

The last game type is a "Fleischlos" or "Asse Solo" (*fleshless* or *aces solo*) with no trump suit at all; every suit is ordered as usual with the queens under the kings but above the jacks which are higher than the nines. The suits are shown in Table 2.5.

2.2.3 Doppelkopf session and compulsory solos

According to the official rules, a doppelkopf *session* for four players consists of six rounds with four games each, totaling 24 games. (There are some variants for five players where one player always needs to sit out.) Within these games, each player must play at least

Suits in descending order per suit			
Clubs	Spades	Hearts	Diamonds
♣A, ♣10, ♣K, ♣Q, ♣J, ♣9	♠A, ♠10, ♠K, ♠Q, ♠J, ♠9	♡A, ♡10, ♡K, ♡Q, ♡J, ♡9	♢A, ♢10, ♢K, ♢Q, ♢J, ♢9

Table 2.5: Suits in an aces solo

one solo, a so-called "Pflichtsolo" (*compulsory solo*). When playing a compulsory solo, the solo player always is in the *first position*, i.e. he starts the card play. In order to avoid that the player that would normally have started this game loses his right to be the starting player, a compulsory solo must be "repeated" in the sense that the same player deals the cards again in the next game. That way, in the end of the session, all players have been first-positioned exactly six times; five times for every game other than a compulsory solo and once with their own compulsory solo. Note that it can happen that one player deals cards more often than the others if he is the dealer for several compulsory solos.

Once the number of remaining games equals the number of players that still need to play a compulsory solo (this may happen because getting good enough cards for a solo does not happen that often), these players are forced to play their compulsory solos, starting with the player positioned next to the dealer (here, "next" means the next player positioned after the dealer who still needs to play a solo). This procedure is called "Vorführung" (maybe best translated as *exhibition*, because the player is given no choice but to play a solo, no matter how "bad" his cards may be). Note that even if another player than the one positioned next to the dealer still needs to play his compulsory solo and would like to do so, he is not allowed to. He will be forced to play his solo when it is his turn during the remaining games. Players thus should always try to avoid getting into this situation by risking to play a solo even with a not so good hand. Compulsory solo games of a "Vorführung" must not be repeated in the sense that the same dealer must deal again, but the player positioned next to the dealer will become the dealer for the next round (even if he has to deal his own "Vorführung", i.e. even he is the next one to play a forced compulsory solo).

After having played their compulsory solo, players are allowed to play more solo games during the session; these are called "Lustsolo" (literally *lust solo*, which means a solo that a player is free to play if he likes to) and they do not give the right of starting card play to the solo player as a compulsory solo does. Note that the first solo a player plays is always the compulsory one and all later solos will be lust solos. This means players cannot choose to first play a lust solo and then later do a compulsory solo[5]. Note that only "announced solos" count as compulsory solo, i.e. a silent solo as explained above and a "failed marriage" (i.e. the case where the marriage player wins the first three tricks

[5]This scenario may sound strange, but for some solo games a player really needs to start playing, otherwise he could not play the solo; so if a player has cards for a solo where he does not necessarily need to start the game, he would like not to use his right to start playing, but he is forced to!

and thus plays alone) do *not* count as a compulsory solo. The reasoning behind this rule is that both of these "non-announced solos" are considered to be "easy solos": when playing a silent solo, the kontra-team does not know it plays against a soloist and thus cannot profit from playing three against one because every player will play by himself as long as he does not know who he plays with. In the case of the failed marriage, the marriage player already made three tricks, and usually the first tricks are very valuable ones. Still these solo games count as lust solos, which is important for the scoring.

2.2.4 Game type determination

As already stated, before a game can actually start, the players need to find out which game type they will play – a regular game, a marriage or any kind of solo. To do so, starting with the player left to the dealer, each player says whether he is "gesund" (*healthy*) or he has a "Vorbehalt" (*reservation*), meaning either that he is fine with playing a regular game or that he wants to either announce a marriage or play a solo. If several players have a reservation, then the following ordering applies, starting with the highest priority:

1. compulsory solo
2. lust solo
3. marriage

To find out which player has what kind of reservation without revealing too much information about the players' cards, players are first asked if they want to play a compulsory solo or not, again starting with the player on first position, i.e. the player left to the dealer. Players who played their compulsory solo already are not allowed to answer "yes". If all players answer "no", then they are asked if they want to play a lust solo. As the first-positioned player among those with a reservation of the same priority is admitted to play a solo, the asking stops as soon as a player answers a questions with "yes". Note that if there are several players having a reservation, then a marriage can never be admitted. This procedure guarantees that always the player with the highest priority is admitted to play a solo, but it is a bit involved. In practice, players often do not ask separately for compulsory solos and then for lust solos, but generally for a solo. A compulsory solo is still always admitted first before a lust solo.

With the same reasoning, a player who is sure that his reservation will be of highest priority may even skip the procedure of saying "healthy" or "reservation" and immediately announce a solo he wants to play. This is also valid later during the reservation procedure, i.e. as soon as a player knows that his solo has highest priority (e.g. because a player positioned in front of him is healthy) he is allowed to announce it and the rest of the reservation procedure will not be finished. This is explicitly stated in the official rules to allow players to prevent other players from revealing information about their cards. For example, if player A is sure he wants to play a solo and has the highest priority but does not immediately announce his solo, some other player B may say "reservation" as

well. If player B now sits before player A and does not want to play a solo, then this is because he wants to play a marriage. Thus he must have both ♣Q – an information, which would then be available to the solo player as well as to all three opponents, which possibly could help the opponents to play against the soloist.

2.2.5 Card play

Every game is divided into twelve *tricks*, where a trick consists of four cards, one card played by each player. The player in the first position plays a card, the remaining three players play a card (according to certain rules which will be explained soon) in the positioned order. After completion of the trick, the winner of the trick takes the cards of the trick and puts them face down in front of him and does not take them onto his hand. These cards are of no more use until the current game is finished, they are only important when counting how many points a player made during the game. Note that players are not allowed to view the played tricks except the most recently completed one.

The rules for playing cards do not differ from the rules for schafkopf or skat. Cards are divided into a trump suit (except for the aces solo) and several non-trump suits, depending on the game type being played. The player who starts the trick is free to choose any of his cards, setting the suit for the current trick. All players have to *follow suit*, i.e. they have to play a card of the same suit. If a player cannot follow suit, he is free to play any of his cards. If the first-played card was of a non-trump suit and a player cannot follow suit and he decides to play a trump card, then he "trumps" the other cards. Trump cards are always higher than any non-trump cards. As a consequence, the player who played the highest trump card wins the trick (in both cases that the trick suit is trump or non-trump). If nobody played a trump card, then the player playing the highest card in the suit of the trick wins the trick. Since each card exists twice, it can happen that players are tied for the highest card because they played the same. In this case, the first-played card always wins.

2.2.6 Announcements

If no marriage is being played, each player knows which team he belongs to. During the first tricks, a player can make an "Ansage/Absage" (*announcement* or *bid*[6]) claiming that his team will reach a specific goal. If a team does not fulfill these self-given goals, it automatically loses even if it had won under normal winning conditions. The interest in taking the risk for such announcements is the increased game value and sometimes also to give information about the strength of a hand to the other players, especially to the teammate.

[6]In German, the official rules distinguish between "Ansage" (*announcement*) for re or kontra, and "Absage" (best translated into *denial* or *rejection*) for the rest, because the intention of the latter is to prevent the other team of reaching a certain amount of points/making a trick at all. For simplicity, I will stick to announcement for all kind of "Ansage" and "Absage".

The first possible type of announcements is "re" or "kontra", depending on a player's team. With the announcement of re or kontra, a team claims to *win* the game, which usually means to gain more than 120 points, with the exception of replies (see later). As a consequence of the announcing player revealing his team, all other players know if they play with or against the announcing player (if they do not know it already anyway, e.g. in case of a solo being played).

The second type of announcements is a whole group of announcements, stating that the opponent team will get less than a certain amount of points. These announcements can only be made *after* an announcement of re or kontra. For example, if re was announced by a player and his partner wants to make an additional announcement, he should identify himself as a re-player before doing so (unless it is clear that he also belongs to re, because kontra did not announce kontra). The possible announcements are:

- "Keine 90" (*No* 90), often abbreviated to "Keine 9" (*No* 9), meaning that the opponents will get *less than* 90 points. This also means they win by reaching 90 points.
- "Keine 60", "Keine 30" (*No* 60, *No* 30): the same as no 90 but with a higher goal of reaching a specific amount of playing points for the announcing team
- "Schwarz" (*black*), meaning the other team will not make a single trick, not even a trick worth 0 points.

Players can only make announcements while they still hold at least a certain amount of cards in their hand (they can always do announcements earlier than the latest moment allowed):

- For announcing re or kontra, the announcing player needs to have at least eleven cards, i.e. the announcement must be made before playing the second card.
- An announcement of no 90 (no 60, no 30, black) requires the player to hold at least ten (nine, eight, seven) cards and the announcement of re or kontra of the team before (does not have to be the same player).

Players are allowed to leave out a lower announcement and to immediately make a high announcement, e.g. to directly announce no 90 (but they have to say for which team), which then also implies the announcement of re or kontra. This is only possible if the left out announcement is still legal at the time of making the announcement. For example, a team which announced re cannot announce no 60 while the announcing player still holds nine cards, because the implied no 90 would be illegal at this moment. As both teams may do as many announcements as they like to, it can happen that no team wins in the end, e.g. when both teams announce (at least) no 90 and both teams reach (at least) 90 points.

An exception for a late announcement of re or kontra is given as an "Erwiderung" (*reply*): whenever a team does an announcement, the other team is allowed to reply re or kontra against this announcement as long as they hold at least one card less than

the announcing team needed for their last announcement. For example, if the re-team announced no 60 (it does not matter when they do so – at the beginning, at the last possible moment, i.e. with nine cards, or sometime in between), then the kontra-team is allowed to say kontra as long as the team is holding at least eight cards. A reply does *not* entitle to further announcements as no 90 etc., unless it was said while holding at least eleven cards (i.e. it was a legal announcement and not only a reply).

In the case of a marriage, the teams are only determined after the "clarification trick". Before this trick has been completed, no announcements are allowed. If a marriage partner is found (or not found) after the second (third) trick, the number of cards a player needs to hold for making an announcement (this includes replies) is reduced by one (two). If the first trick already determines the teams, nothing is changed.

2.2.7 Game evaluation

First, the winning team needs to be determined if there is a team that won.

If no announcements except re or kontra were made, the following holds: the re-team normally wins the game if it reaches at least 121 points and it also wins by reaching 120 points if it did not announce re and if kontra at the same time announced kontra but not more. Similarly, kontra wins with 120 points if both teams did not announce anything or if re announced only re and kontra announced only kontra or nothing. If only kontra announces kontra and re does not announce anything then kontra needs 121 points to win.

If a team announced more than just re or kontra, then the following winning criteria hold for both teams: a team must reach at least 151, 181 or 211 points to win if it announced no 90, no 60 or no 30. If a team announced black, then the opposing team is not allowed to take a single trick. If a team did not announce anything above re or kontra, then it suffices to reach 90, 60 or 30 points to win if the other team announced no 90, no 60 or no 30. A team also wins by making a trick if the other team announced black (and if the team did not commit to any goal by announcing no 90 or more). Note that the latter winning criteria are independent of the announcing time of re or kontra, e.g. if a team announces re in a regular manner (while having at least eleven cards) and thus not as a reply, but then later kontra announces no 90 or more, then re still wins by reaching 90 points and it does not need to reach 121 points, even if at the moment of announcement of re, that would have been the case (because kontra did not announce anything at that moment). Both re and kontra announcements only state that the team is going to win the game (and not to reach more than 120 points) – if the winning conditions are changed later towards some stronger conditions, then re and kontra are adapted to those new conditions.

As already noted above, in some cases where both teams announced at least no 90, it may happen that no team won. This is important for the following scoring rules.

A game is valued by the *score points* it is worth (contrasting the points made by winning tricks during the game)[7]. Score points are granted in a zero-sum manner, that

[7]Again note that in German, the distinction is more clear by using "Augen" (literally *eyes*) for card

is the losing team gets as many negative points as the winning team gets positive points. For all solo games, the so calculated score points value will be multiplied by three for the soloist (in order to keep the valuation zero-sum).

The following score points are always granted to either the winning team if there is one or to the team(s) to which they apply if there is no winning team:

- (1*a*) One score point for the team reaching at least 120 (90, 60, 30) points against an announcement of no 90 (no 60, no 30, black) of the other team. Thus a team could theoretically get up to four score points by reaching at least 120 points against an announcement of black.

- (1*b*) One score point for the team reaching at least 151 (181, 211) points/winning all tricks (i.e. the other team has less than 90 (60, 30) points/is black). Thus a team can get up to four score points by winning all tricks.

The following score points are only granted to the winning team if existent:

- (2*a*) One score point for winning the game.

- (2*b*) Two score points each for an announcement of re and kontra.

- (2*c*) One score point for each other announcement (i.e. no 90 etc.), no matter of which team. Thus a team could theoretically get up to eight score points (including the score points for re and kontra) if both teams announced black.

There are several special score points which are only granted in non-solo games, i.e. in regular and successful marriage games. These points are always granted to both teams, no matter if they won or not (it may happen that the winning team gets negative points because the losing team gained more special points):

- (3*a*) One score point for the kontra-team winning "Gegen die Alten" (*against the elders*), i.e. winning against the re-team which has both elders (the ♣Q). This is intended to reward the kontra-team which has lower winning chances because the re team always has at least two high trumps.

- (3*b*) One score point for the team catching a "Fuchs" (*fox*) – a ♢A – of the other team. A team "catches a fox" if a player of the team wins a trick where a player from the opposing team played a ♢A. No extra points are granted for "bringing home" a fox of a teammate, though. As there exist two foxes, a team can get up to two score points by catching both of them.

- (3*c*) One score point for the team winning the last trick with a "Karlchen" (*charlie*), a ♣J. A team wins the last trick "with a ♣J" if a player of the team plays a ♣J and it is the highest card in the trick, i.e. no other higher trump card is in the trick

points and "Punkte" (*points*) for score points. To avoid speaking about "eyes", I'll use points for card value points and score points for the points granted to the players after the game finished, although I may sometimes drop the "score" in score points when it is clear from context.

(no matter if it was played by the teammate or the other players). There are no points granted for preventing a player from winning the last trick with a charlie, i.e. by playing a higher trump card.

- ($3d$) One score point for each trick containing at least 40 points (a so-called doppelkopf) made by the team.

All score points are added up for each team and the players of the team with more points gets the difference of points between both teams as a positive value, the players of the other team get the difference of points as a negative value. Score points are summed up for players over all games of the session. The winner is the player with the highest amount of score points at the end.

The following examples illustrate the calculation of score points. Score points are calculated for each team first, then the difference between the teams is the final result, as described above. Note that in practice, people tend to directly add up all score points (either positive or negative) for the winning team only rather than doing it for both teams separately and calculating the difference afterwards. The result obviously is the same.

- Example no. 1:
  regular game; re announced no 90, re gets 182 points, kontra wins the last trick with a charlie
  - re won the game: +1 for re ($2a$)
  - re was announced: +2 for re ($2b$)
  - no 90 was announced: +1 for re ($2c$)
  - kontra has less than 90 points: +1 for re ($1b$)
  - kontra has less than 60 points: +1 for re ($1b$)
  - kontra won the last trick with a charlie: +1 for kontra ($3c$)

  Total: the difference of score points is $6 - 1 = 5$, thus the re-players are granted +5, the kontra-players −5 score points.

- Example no. 2:
  regular game; re announced no 60, kontra announced kontra, re gets 145 points, re catches a fox of a kontra-player
  - kontra won the game: +1 for kontra ($2a$)
  - kontra won against the elders: +1 for kontra ($3a$)
  - re was announced: +2 for kontra ($2b$)
  - no 90 was announced: +1 for kontra ($2c$)
  - no 60 was announced: +1 for kontra ($2c$)
  - kontra was announced: +2 for kontra ($2b$)

  - kontra reached at least 90 points against an announcement of no 60 of re: +1 for kontra (1*a*)
  - re caught a fox: +1 for re (3*b*)

  Total: the difference of score points is $9-1=8$, thus the kontra-players are granted +8, the re-players −8 score points.

- Example no. 3:
  Same as example no. 2, but kontra additionally announces no 90. In this case nobody wins (both teams fail to reach their self-given goals), thus none of the score points only being granted for the winning team (i.e. rule 2) are granted, but rule 1 is now applied to both teams (and not only to the winning team)!
  - re reached at least 120 points against an announcement of no 90 of kontra: +1 for re (1*a*)
  - kontra reached at least 90 points against an announcement of no 60 of re: +1 for kontra (1*a*)
  - re caught a fox: +1 for re (3*b*)

  Total: the difference of score points is $2-1=1$, thus the re-players are granted +1, the kontra-players −1 score points.

- Example no. 4:
  Solo game; re announced no 90, re gets 85 points
  - kontra won the game: +1 for kontra (2*a*)
  - re was announced: +2 for kontra (2*b*)
  - no 90 was announced: +1 for kontra (2*c*)
  - kontra reached at least 120 points against an announcement of no 90 of re: +1 for kontra (1*a*)
  - re has less than 90 points: +1 for kontra (1*b*)

  Total: the difference of score points is $6-0=6$, thus the kontra-players are granted +6 score points and the single re-player gets $-(6\times 3)=-18$ score points.

2.3 Doppelkopf and AI games research

Obviously doppelkopf could be seen just as "another trick-taking card game with imperfect information" as skat, bridge or others that have previously been examined in AI games research and its related fields. Still there are some significant differences to the games mentioned before, such as complexity of the state space, the fact that teams are not always known before playing and change throughout a session, the fact that points made during the game are also important and not only the number of tricks as in bridge for example, and the different conventions good players use to share information about their hands and to optimize card play and game values.

Doppelkopf is played with 48 cards and four players, thus there are

$$\binom{48}{12} \cdot \binom{36}{12} \cdot \binom{24}{12} \cdot \binom{12}{12} \approx 2.358 \cdot 10^{26}$$

different card deals. For one fixed deal, the number of possible states can be calculated by computing the number of possibilities of how i remaining cards can be distributed among the 4 players and summing up over all possible numbers for i. If a player needs to have k cards, there are $\binom{12}{k}$ possibilities for a fixed deal. Combining the above, the number of states per deal is the following:

$$\sum_{i=0}^{48} 4 \cdot \binom{12}{\lfloor (i+3)/4 \rfloor} \cdot \binom{12}{\lfloor (i+2)/4 \rfloor} \cdot \binom{12}{\lfloor (i+1)/4 \rfloor} \cdot \binom{12}{\lfloor i/4 \rfloor} \approx 2.384 \cdot 10^{13}$$

As there are states that are consistent with many card deals, multiplying the number of deals with the number of states per deal only yields an upper bound for the size of the complete state space: $2.358 \cdot 10^{26} \cdot 2.384 \cdot 10^{13} \approx 5.622 \cdot 10^{39}$. Additionally there are a lot of possible game moves to make before the actual card playing starts in order to determine the game type, each resulting in a game subtree which has nothing in common with other subtrees (because another player is the soloist, another game type is chosen etc.), again increasing the number of possible states. Also the fact that announcing is possible for quite a long time during the first tricks (depending on what announcements players make) multiplies the size of the state space by a good amount, although the actual game state with respect to card play is not being changed by an announcement (except for the knowledge of the announcing player's team). Compared to skat which has been well examined over the last years (e.g. a skat player based on Monte Carlo (MC) simulation [14], an improved player with features such as state inference [4] or a player based on UCT [17]), doppelkopf is a way more complex game and thus it is of interest to see how far similar approaches used for skat can get at doppelkopf.

As for classification of the game doppelkopf, it is a four person zero-sum game with imperfect information. Although for some game types it may be broken down to a two person game because teams are fixed, this is practically not valid because the players still do not know their team's cards, thus they still have to play "alone" in a certain way. Also players can infer information about the other players' cards, but nevertheless players usually only have near-to-perfect information during the last two tricks.

To my best knowledge, doppelkopf has not been examined by the AI games research community yet, thus it seems to be an interesting possibility to start doing so by trying to adapt the UCT algorithm which has been successfully applied to similar games before. In their work "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search" Long et al. [15] state several reasons why Monte Carlo search methods have a lot of success when being applied to games with imperfect information, thus giving hope that also for doppelkopf a UCT algorithm with an MC search may be implemented successfully. The question though is whether UCT can handle all the subtleties of doppelkopf, especially the enormous strategic depth of the game which requires a lot of good handling of hidden and indirect information when played by human players.

3 UCT

This section introduces the original UCT algorithm as developed by Kocsis and Szepesvári [12].

3.1 Monte Carlo tree search methods

Consider a large state space Markovian Decision Problem (MDP) or a large game tree with the problem of finding the optimal (or at least near-optimal) action for a given state (provided that a generative model for the MDP/the game is available). Often, computing a solution with a "classical" deterministic approach is infeasible, explaining the use of so-called *Monte Carlo (MC) methods*. For the purpose of using the UCT algorithm, I will focus on *Monte Carlo tree search* (MCTS) methods and always imply a tree search algorithm even if only writing a "MC algorithm" or similar. An MCTS method is an algorithm relying on repeated random sampling to compute an (approximate) result. For this purpose, it builds a search tree with the root node corresponding to the given state for which an near-optimal action has to be found. An estimated value for actions at the root is computed by averaging over the accumulated reward values obtained during sampling. The estimated values obtained converge to their expectation value when choosing actions uniformly at random.

As even a simple MCTS algorithm quickly comes to its limits for larger problems, it needs to be improved further. This is the motivation for Kocsis and Szepesvári [12] who propose to choose actions during MC sampling not randomly, but selectively. An exemplary calculation for a problem with a large number of available actions at each node and with a fixed depth d for the MC search tree shows that by restricting sampling to only use half of the available actions, the overall work load can be reduced to $(1/2)^d$. Consequently, if a subset of "suboptimal" actions may be identified early on in the search procedure (and then ignored for sampling), then the time gained can be used to search more promising parts of the search tree. This "deepening" of the search in interesting parts of the tree allows for refining estimate values faster and thus should yield a huge performance improvement. Furthermore, using a guidance for selecting "good" actions is important to ensure that the estimate values converge towards the optimal value.

For the purpose of using the UCT algorithm, I will consider *rollout-based MCTS algorithms* which build the search tree in an incremental manner. A sample can be considered a rollout (also called episode), which is a sequence of state-action pairs that represents a path from the initial state to a terminal node. During a rollout, the search tree is traversed starting at the root and then choosing an action according to a selection strategy. When a terminal node (or a depth limit) is reached, the reward obtained at

that node is applied to the nodes visited during the rollout to update the estimate values for actions.

A general advantage of using a rollout-based algorithm is its *anytime aspect*. As the algorithm iteratively increases the depth and/or width of the search tree, the value of the information computed by the algorithm increases step by step. Thus in settings where time or memory is limited, one can profit from a rollout-based MCTS algorithm by stopping it after an arbitrary number of rollouts, after a certain time elapsed or when a memory limit is reached and still the algorithm will have computed some meaningful results. These results might have been improved if more time or memory was given, but the result is not incorrect nor not available as it would be the case with many other algorithms if prematurely stopped.

Another more specific advantage of a rollout-based MCTS algorithm is to keep track of estimates of visited state-action pairs, thus enabling the use of this information for future episodes. The main idea of the UCT algorithm is to replace the uniform random sampling by a selective sampling strategy which relies on information gained during previous rollouts. If action selection is done in a good way (thus identifying a set of "suboptimal" actions mentioned above), this should potentially speed up the convergence of estimate values for the action and thus the convergence toward an optimal action for the given state. The higher the fraction of nodes being re-encountered during different rollouts, the higher the profit of a rollout-based MCTS algorithm compared to the vanilla MCTS algorithm.

An action is suboptimal at a given state if its value is less than the maximum value of all actions available at this state. As action values depend on the subtree rooted at the successor generated by applying the action at the state, the problem consists of minimizing the estimation error of action estimates fast. An MCTS algorithm should balance between testing alternatives that it did not visit so far (in earlier episodes) and using actions with the highest estimate value so far (over all previous episodes) in order to make those estimate values more precise (as precision of estimate values for state-action pairs increases with the number of rollouts visiting the according states). This problem is known as the *exploration-exploitation dilemma*. A most basic form of such a dilemma shows up in the multi-armed bandit problem.

3.2 Multi-armed bandit problems and UCB1

A *multi-armed bandit* gets its name from a classical slot machine (also called one-armed bandit) with multiple levers (or arms). When pulled, each lever yields a reward drawn from a distribution of that particular arm. A gambler playing the slot machine does not know the distribution of rewards in advance, and his aim is to maximize his profit. Thus he faces the exploration-exploitation dilemma each time he needs to decide which arm to play: either an arm of which he assumes or hopes that it has a "good" payoff, or an arm which he did not pull yet (or only few times) and which could even yield a higher reward in the long term.

A *K-armed bandit problem* is an example for a stationary stochastic process and is

defined by the *sequence of random variables* X_{it} for each arm $i = 1, \ldots, K$ and time step $t \geq 1$. Successive play of arm i yields rewards $X_{i1}, X_{i2}, \ldots$ with $X_i = \{X_{i1}, X_{i2}, \ldots\}$ being independent and identically distributed according to an unknown distribution function. An *allocation strategy* or a *policy* A maps each time step t to an arm i to play, where the arm i to be played at time t can depend on earlier results of A, i.e. on results of playing some arm j at some time $u < t$. The expected reward of allocation policy A after n time steps is defined as $\mathbb{E}_{A,n} = \mathbb{E}\left[\sum_{i=1}^{K} \sum_{t=1}^{T_i(n)} X_{it}\right]$, where $T_i(n) = \sum_{t=1}^{n} \mathbb{1}(I_t = i)$ is the number of times arm i has been chosen by A up to time n and where $I_t \in \{1, \ldots, K\}$ is the index of the arm being played by policy A at time t. The optimal allocation policy A^* is the one playing the optimal arm at all time steps, thus maximizing the total expected reward $\mathbb{E}_{A^*,n} = \max_i \mathbb{E}\left[\sum_{t=1}^{n} X_{it}\right]$. The *regret* $R_{A,n}$ of a policy A after n time steps is defined as the loss caused by not always having played the best arm, thus: $R_{A,n} = \mathbb{E}_{A^*,n} - \mathbb{E}_{A,n}$. The goal for a policy which tries to solve the exploration-exploitation dilemma is to minimize its regret.

The *Upper Confidence Bounds* (UCB1 – there exist two variants) policy for the multi-armed bandit problem was developed by Auer et al. [1] and was shown to guarantee a worst-case regret logarithmic in the number of total plays under the assumption that the payoffs lie in the interval $[0, 1]$. As this is the optimum that a policy can reach for a large class of distributions [12], it thus resolves the exploration-exploitation dilemma. The algorithm keeps track of the average reward $\overline{X}_i$ of each arm i and the number of times $T_i(n)$ it was played up to and including the current time step n. For initialization of $\overline{X}_i$ and $T_i(n)$, each arm is played exactly once before starting the actual algorithm. The algorithm then chooses to play the arm with the best upper confidence bound, calculated according to the UCB1 formula:

$$I_t = \arg\max_i (\overline{X}_i + c_{i,n})$$

where $c_{i,n}$ is a bias term which depends on the time step, i.e. the total number of times an arm was played, and on the number of times the specific arm was played:

$$c_{i,n} = \sqrt{\frac{2 \log n}{T_i(n)}}$$

This way, arms which have not been played many times get a growing weight over time and thus eventually will get selected for exploration even if their estimated reward may be low. This weight however decreases with increasing number of total plays and thus for large n, an arm with a low estimated reward only gets selected if it was played significantly less often than other nodes with higher rewards. Note that also the initialization (playing each arm once) is consistent with the UCB1 formula because $c_{i,n}$ would grow to infinity when $T_i(n)$ was approaching zero, and thus an arm with $T_i(n) = 0$ would need to be played anyway. So the initialization just does what applying the algorithm would do if division by zero was possible in the sense that the result was a infinitely high number.

Assuming that rewards X_{it} lie in the interval $[0, 1]$, Auer et al. [1] prove with the help of Chernoff bounds and Hoeffding's inequality that the following bounds for the UCB1 policy hold at time step n:

$$\mathcal{P}(\overline{X_i} \geq \mathbb{E}(X_i) + c_{i,n}) \leq n^{-4} \tag{3.1}$$

$$\mathcal{P}(\overline{X_i} \leq \mathbb{E}(X_i) - c_{i,n}) \leq n^{-4} \tag{3.2}$$

These bounds state that the average reward achieved by UCB1 lies in the interval of the the real expectation value plus/minus the bias term with high probability (increasing with the number of played arms).

3.3 The UCT algorithm

UCT stands for *UCB1 applied to Trees* and was first described by Kocsis and Szepesvári [12]. It has been applied to the game of Go (a game with perfect information) shortly after its development and since the resulting program MoGo was a huge success [10, 9], the algorithm has experienced a growing popularityf also in many other domains such as General Game Playing (CadiaPlayer, [7, 8]), Klondike Solitaire [3], the Canadian Traveller Problem [6] and Probabilistic Planning (PROST, [11]).

The main idea behind UCT is to improve a (rollout-based) MCTS by using the UCB1 policy at inner nodes of the tree when sampling actions, thus treating each inner node as a separate multi-armed bandit problem. The pseudocode for one iteration of the UCT algorithm for a game tree is given in Algorithm 1, taken (and slightly adapted) from a work by Shafiei et al. [18]. The first three lines initialize the root of the search tree without expanding it yet. The loop starting at line 5 searches through the nodes of the existing tree by determining values for all successors and then following the subtree rooted at the node with the maximum value (line 15). Value determination is done as follows: If there exists an unvisited successor at the current node (line 8), then its value is set to an infinitely large number. Otherwise, if all successors have been expanded already, the UCT formula (line 11) is applied to determine the successor's value. Note that by doing this extra check for unvisited successors, division by zero in the UCT formula is avoided, as successors that have been expanded have also been visited at least once. Also preferring unvisited successors over already visited ones is consistent with the UCT formula, as a node with a visit counter of zero would get an infinite high exploration bonus and thus would get selected also when applying the formula. This also corresponds to the afore mentioned "playing each arm once" initialization commonly used for multi-armed bandit problems.

If the leaf node where the while-loop (line 5) stopped at needs to be expanded (line 17), then this is done and a random successor of the newly added node is chosen (which is again a leaf node). Here, expanding means that the successors are added to the tree, their visit counter and the accumulated rewards are set to zero, but they are not expanded themselves yet. If the current node is a non-terminal state (line 21; this can only be the case after the above check for exploration succeeded and thus a new node

Algorithm 1 One iteration of the UCT algorithm

```
1: if root = NULL then
2:     root ← MakeNode                          ▷ Initializes the root of the tree
3: end if

4: traverser ← root
5: while not IsLeaf(traverser) do               ▷ IsLeaf(node) tests if
                                                  the given node is a leaf
6:     expandLeaf ← TRUE
7:     for i = 1 to number[traverser.children] do
8:         if traverser.children[i].counter = 0 then   ▷ Will be incremented in
                                                         UpdateValues(...)
9:             values[i] ← ∞                ▷ a vector of reward values for all players
10:            expandLeaf ← FALSE
11:        else
12:            values[i] ← traverser.children[i].values + C sqrt( log(traverser.counter) / traverser.children[i].counter )
13:        end if
14:    end for
15:    traverser ← traverser.children[arg max values]
16: end while

17: if expandLeaf and not IsTerminal(traverser) then
                                    ▷ IsTerminal(node) tests if the given node
                                      corresponds to a terminal state
18:    ExpandNode(traverser) ▷ Adds all children of the node and sets their
                               counters to zero, but does not expand them
19:    traverser ← RandomChild(traverser)
20: end if

21: if not IsTerminal(traverser) then
22:    outcome ← DoMonteCarloSimulation(traverser)
                                    ▷ Does a Monte Carlo simulation (with random
                                      action choice) until a terminal node is reached
23: else
24:    traverser.values ← GameValue
                                  ▷ GameValue is a vector of rewards for all players,
                                    computed from the value of the completed game
25:    outcome ← traverser.values
26: end if

27: UpdateValues(traverser, outcome)
                  ▷ Updates the accumulated rewards values and increments the
                    number of visits counters of the nodes along the path to the root
```

has been added to the tree, but still there was not terminal state reached yet), then an MC simulation (with random action selection) is carried out starting at the current node until a terminal state is reached where the state's UCT reward must be computed. Otherwise, if the while-loop (line 5) ended up at a leaf (and thus a terminal state) or the expansion of the leaf reached a terminal state (line 23), then no simulation needs to be performed. The UCT rewards corresponding to the terminal state where the iteration of the UCT algorithm ended up are then propagated back along the path of all visited nodes to the root (line 27). This means all nodes visited during the iteration get their visit counter incremented and the obtained UCT rewards are added.

When doing a theoretical analysis of the UCT algorithm, a non-stationary bandit problem – in contrast to the stationary multi-armed bandit problem the UCB1 algorithm was originally designed for – needs to be examined. This is the case because the sampling probabilities for a specific successor of a node in the tree are changing with each rollout and hence the expected reward $\overline{X}_i$ of a node i will drift over time. To take these drifting rewards into account, the bias term $c_{i,n}$ needs to be replaced or modified appropriately so that the inequalities bounding the average rewards in the case of a stationary multi-armed bandit problem (see equations (3.1) and (3.2)) are still fulfilled even in the case of non-stationary problems. Kocsis and Szepesvári [12] first show that UCB1 can also be applied to non-stationary bandit problems so that its properties still hold if the rewards satisfy some drift conditions. Their main result is to modify $c_{i,n}$ by multiplying a constant C and by removing the factor 2 in front of the logarithm in the nominator of the formula. They further prove that the UCT rewards obtained at internal nodes of the game tree using this new bias term with an appropriate constant C satisfies the drift conditions needed for non-stationary bandit problems. The proof of the consistency of the whole procedure is then done with the means of induction over tree depth. For more detailed analyses and a proof sketch see [12] and the references mentioned there.

4 Card Assignment Problem

UCT is designed for the use under perfect information which is not available when playing doppelkopf. There are two possible solutions for this problem: either one directly applies the UCT algorithm to the belief state space rather then the concrete state space, thus dealing with information sets rather than single states of complete information, or one completes the missing information before applying the UCT algorithm. This could be done by assigning the remaining cards (i.e. the cards that the player does not hold and that have not been played before) to all other three players which would then result in a game with complete information. The second approach seemed to be the more appropriate one in order to use the "original" version of the UCT algorithm and thus is the one chosen solution used in this work. Therefore, an algorithm for generating consistent card assignments needs to be implemented.

The first design goal of an algorithm for a UCT player playing doppelkopf solving the card assignment problem is that it preferably assigns the card uniformly at random to avoid generating any bias towards sampled worlds which could then influence the MC search of the UCT algorithm. Still the algorithm should not assign cards purely at random but first consider available information about other players and especially it should distribute the cards in a way which is consistent with the prior game process (of course the prior game could be completely ignored, but this would probably have a very bad impact on the performance of the UCT algorithm).

4.1 Consistency of card assignments

The starting point for the card assignment problem is the following: a game of doppelkopf is at a certain time step where a game type has been chosen, maybe some cards have already been played and/or some announcements have been done. A player only knows his own cards, but he has some additional information about the cards of the other players:

1. In a marriage game, the marriage player must hold both ♣Q and thus none of the other players can hold a ♣Q.
2. If a player, asked if he wants to play a solo, denied to do so, then he wanted to play marriage, thus the same as in 1 holds.
3. In a regular game, a player who announced re must hold (at least) one ♣Q in his hands if he did not play one yet.
4. In a regular game, a player who announced kontra cannot hold a ♣Q.

5. In any game, a player who did not follow suit at a given moment cannot hold any card belonging to this suit.

To summarize, a player has the following information: he knows which cards have been played and which cards he holds in his hands and thus can compute the set of remaining cards and can possibly exclude for some other player to hold some specific cards or the other way around, he may know that some player must hold a specific card. Note that all cards exist twice in the card deck and that a computer program in contrast to a human player has to differentiate syntactically between two semantically identical cards. This means that if one player must have a ♣Q, the player does not know which of the two it is unless he holds the other one or the other one was played already. This results in an algorithm which may seem to be a bit more tedious than one expects at first glance.

Before describing the algorithm implemented to solve the card assignment problem, the following section will compare the problem at hand with two similar problems.

4.2 Related problems

Obviously the card assignment problem can be seen as a *matching problem* [5] from graph theory: given a graph $G = (V, E)$, a matching M is a set of pairwise non-adjacent edges, i.e. no two edges share a common vertex. In the case of the card assignment problem, the set of remaining cards needs to be assigned to the players, or to be more precise, to the player's card slots (i.e. each player has a number of card slots equal to the number of cards he needs to get assigned). The resulting graph is a *bipartite graph* $G = (V = (X, Y), E)$ [5], where X is the set of cards to be assigned, Y is the set of the card slots of all players and E is the set of edges between X and Y. There is an edge between a card $x \in X$ and a player's card slot $y \in Y$ if and only if this player can have this card, i.e. if assigning x to y is consistent with the prior game process. Obviously X and Y have the same cardinality and the card assignment problem translates to the problem of finding a *perfect (bipartite) matching* in the corresponding graph.

Alternatively the card assignment problem can be formulated as a *constraint satisfaction problem* (CSP) [16]: a CSP is a 3-tuple $\langle X, D, C \rangle$, consisting of a set X of variables, a domain D for the variables in X and a set of constraints C over the variables X. A solution for a CSP is an evaluation $v : X \to D$ which satisfies all constraints C. In the case of the card assignment problem, X is the set of cards that need to be assigned to players, D is the set of card slots of all players and C is the set of constraints. There are two types of constraints: the first type is unary and restricts the domain of a variable $x \in X$ so that the card corresponding to x can only be assigned to slots of the player(s) that can have the card in a consistent card assignment. The second type of constraints is binary and states that all variables $x \in X$ must have a different value $d \in D$, i.e. no two cards can be assigned to the same card slot of a player. Then every solution to the CSP constructed in the way described corresponds to a consistent card assignment.

For the purpose of designing an algorithm which assigns the cards uniformly at random, the probability for a card to be assigned to a specific player needs to be known. This is where the card assignment problem could be transformed into a matching problem,

a CSP or a third representation, with the goal of computing the number of consistent card assignments which corresponds to the number of solutions to the problem. If this number is known, then an algorithm for the card assignment problem can be designed in the following way: let N be the number of consistent card assignments and let x be a card of the remaining cards X that still needs to be assigned. Assign the card hypothetically to player i if this is allowed in the sense that it is consistent with the prior game process. Then the number N_i of consistent card assignments, given that x would be assigned to i, is computed, otherwise N_i is set to zero. This has to be done for all four players, resulting in N_1, N_2, N_3 and N_4. Then card x gets ultimately assigned to player i with probability N_i/N. Doing this for all remaining cards $x \in X$ yields a uniformly random card assignment.

There exist several polynomial algorithms for solving matching problems (e.g. the Hungarian algorithm [13]) and for the special case of bipartite matchings, there is even a fully polynomial time randomized approximation scheme for counting the number of bipartite matchings [2]. Also for solving CSPs, there are several known algorithms, most of them based on backtracking, constraint propagation (e.g. the AC3-algorithm [16]) or local search. However determining the number of perfect matchings in a general graph is #P-complete and also using the polynomial approximation scheme for the bipartite case seems to be too much work load for just computing a number which will then be used in another algorithm which does the actual card assignment. Similarly, solving CSPs with a finite domain is an NP-complete problem in general and thus computing the number of solutions for such a CSP is at least #P-complete. Computing the number of solutions for complex problems before the actual card assignments can be generated seems to be a too large overhead to be justified just by having a perfect uniformly random card assignment. Instead I propose my own rather intuitive card assignment algorithm which will be described in the next section.

4.3 Card assignment algorithm

The general idea of the algorithm is the following: first assign all cards that need to be assigned in a specific way, i.e. there is only one possibility of assigning them in the sense that there exists only one player who can have the card. This is repeated until there are no more such uniquely assignable cards. Then the algorithm needs to check if a player still needs to get a ♣Q, which can be the case because he announced re, because three other players announced kontra, because two players announced kontra and the second re-player is doing the card assignment and thus knows the last player needs to have the other ♣Q or because a player wanted to play a marriage but could not because someone else plays a solo – all other cases such as marriage are covered by the case above where also a ♣Q can uniquely be assigned to only one player. If there is such a player, then this is first taken care of by assigning one or two ♣Q to the player(s) in question, breaking ties in an arbitrary way as it does not make a semantic difference for the UCT algorithm. As soon as no more cards are to be assigned in a unique way, cards get assigned to a random player. Each time after a card was assigned to a player,

two checks are performed which possibly lead to a repetition of the algorithm because a player may have all cards he needs or because a player needs to get all cards he can possibly get.

Algorithm 2 The card assignment algorithm: Part 1

```
1:  data
2:      remainingCards: set<Cards>
3:      cardsToAllowedPlayers: Card → set<Player>
4:      playersThatNeedCQ: set<Player>
5:      playersCardsCount: Player → int
6:      cardsToAssignedPlayer: Cards → Player
7:  end data

8:  procedure AssignCard(card, player)
9:                              ▷ requires: remainingCards.contains(card)
10:                                         cardsToAllowedPlayers[card].contains(player)
11:                                         playersCardsCount[player] > 0

12:     cardsToAssignedPlayer[card] ← player
13:     remainingCards.remove(card)
14:     ReduceByOne(playerCardsCount[player])

15:     if playerCardsCount[player] = 0 then
16:         for card in remainingCards do
17:             cardsToAllowedPlayers[card].remove(player)
18:         end for
19:     end if

20:     if IsQueenOfClubs(card) and playersThatNeedCQ.contains(player) then
21:         playersThatNeedCQ.remove(player)
22:     end if
23: end procedure
```

This paragraph describes the first central method of the algorithm in more detail: AssignCards(). The pseudocode of the algorithm is split into two parts and shown in Algorithms 2 and 3. There are four data structures that contain the information the UCT player has about the other players cards. They should be precomputed once and must be available to both involved methods of the algorithm. As AssignCards() modifies these data structures and in order to generate a lot of card assignments for the same given state, they should also be stored separately as a copy in order to avoid recomputation. These data structures are shown in the first block starting at line 1: a set of cards *remainingCards*, storing all cards that need to be distributed among players, a map *cardsToAllowedPlayers* of the same size, mapping each remaining card to a set of players being able to have this card, a set of players *playersThatNeedQueenOfClubs* that contains all players that need to have a ♣Q and a map *playersCardsCount* which maps

each player to the number of cards he must get assigned. There is an additional data structure *cardsToAssignedPlayer*, initially empty, which will store the result of the card assignment by mapping each card from *remainingCards* to the player it got assigned to.

Starting in line 8, the method AssignCard(*card, player*) is shown. Given a player *player* and a card *card*, it assigns *card* to *player*. To do so, the three statements shown in lines 9 to 11 are required to hold: *card* must be contained in the set of remaining cards, *player* must be allowed to get *card* and the player still needs to get at least one card, i.e. he did not get as many cards as he needs yet. The method then does the actual assignment, removes the card from the set of remaining cards and decrements the number of cards the player still needs by 1. In line 15, a check to determine if the player now has enough cards is performed. If this is the case, the player gets completely removed from all entries in *cardsToAllowedPlayers* so that he will not be considered further. The next check is performed in line 20 and tests if the card just assigned is a ♣Q and if the player the card was assigned to still needed to get a ♣Q. In the case the check succeeds and the player gets removed from *playersThatNeedCQ*. This concludes the first method AssignCard.

The core method of the card assignment algorithm, AssignCards(), starts at line 24 and will be described next. The first check at line 25 is a termination criterion, as the method calls itself recursively, each time assigning exactly one more card to a player. The **for**-loop beginning at line 28 searches for the first card in the set of remaining cards which can only be assigned to one player. If such a card is found, the method AssignCard is called for this card and the player that can have it. Afterwards, AssignCards is called again in order to possibly find more cards that are to be uniquely assigned. This needs to be done because AssignCard may always change *cardsToAllowedPlayers* by removing a player who got all the cards he needs.

If there are no (more) cards that only can be assigned to a single player, then the next "unique assignment possibility" check is performed: starting at line 36, the algorithm iterates over all three other players, i.e. the players different from the UCT player who uses the card assignment algorithm for the UCT algorithm. The set of cards that the player (still) can have has to be computed. If the number of cards the player can have equals the number of cards he still needs to get, then all these cards get assigned to that player. As this finally removes the player from all entries in *cardsToAllowedPlayers*, the method AssignCards must be called again, starting again with the first loop where it is now again possible to find a card which can only be assigned to one player. Note that the recursive call stops the **for**-loop in line 36, but this is still a consistent procedure as it is not possible that assigning a card which can only be assigned to one player could be "wrong". It is especially impossible that if another player who needs to have exactly the cards he can have (which would have been checked if the **for**-loop was continued) cannot have all these cards assigned anymore, because that would be a contradiction to the existence of a consistent card assignment as such. Also, if two players need exactly all the cards they can have, also the third player's cards are exactly determined. More formally, let a be the set of cards which can be assigned to players 1 and 2, b the set of cards for players 1 and 3, c for 2 and 3 and d for all three players. Let the number of cards player 1 still needs be denoted by N_1, similar N_2 and N_3 the number of cards

Algorithm 3 The card assignment algorithm: Part 2

```
24: procedure AssignCards
25:     if remainingCards = ∅ then
26:         return
27:     end if

28:     for card in remainingCards do
29:         players ← cardsToAllowedPlayers[card]
30:         if |players| = 1 then
31:             player ← the element in players
32:             AssignCard(card, player)
33:             return AssignCards()
34:         end if
35:     end for

36:     for player in { p ∈ {1, 2, 3, 4} | p ≠ UCT player } do
37:         possibleCards ← { c | remainingCards.contains(c)
                                ∧ cardsToAllowedPlayers[c].contains(player) }
38:         if |possibleCards| = playersCardCount[player] then
39:             for card in possibleCards do
40:                 AssignCard(card, player)
41:             end for
42:             return AssignCards()
43:         end if
44:     end for

45:     if playersThatNeedCQ ≠ ∅ then
46:         card ← some element of { c | remainingCards.contains(c)
                                       ∧ IsQueenOfClubs(c) }
47:         player ← some element of playersThatNeedCQ
48:         assignCard(card, player)
49:         return AssignCards()
50:     end if

51:     card ← first element in remainingCards
52:     player ← ChooseUniform(cardsToAllowedPlayers[card])
53:     AssignCard(card, player)
54:     return AssignCards()
55: end procedure
```

players 2 and 3 still need. Furthermore it must always hold that the number of cards a player can have is at least as big as the number of cards he still needs to have. Formally, $|a| + |b| + |c| \geq N_1$, $|a| + |c| + |d| \geq N_2$ and $|b| + |c| + |d| \geq N_3$. Obviously, entirely removing a player from the lists because he got all cards does not break the inequalities that guarantee that there are enough cards for the other two players.

When the algorithm for the first time reaches the next if-statement at line 45, then there are no more cards that can only be assigned in a unique way. Before assigning cards randomly though, there is another check to be performed: if there is still a player who needs to get a ♣Q assigned, then this is done next. In the case that there are even two players who need a ♣Q, then an arbitrary ♣Q of the two cards is assigned to an arbitrary player of those who still need one. The algorithm will then recursively call AssignCards again and will also assign the other remaining ♣Q to the correct player in one of the future calls to the method. When none of the above checks trigger, then line 51 will be reached and the card will be assigned randomly to one of the players who can have it. Note that this implementation of the card assignment algorithm does not use card slots of players, but the players themselves. This is discussed further in the next section.

4.4 Analysis of the card assignment algorithm

As stated at the beginning of this chapter, an important goal when generating card assignments is to avoid any bias which could influence the UCT algorithm. Thus it is important to have a look at the card assignments produced by the algorithm described in the previous section. However, one should first note that this card assignment algorithm always terminates because in each call to AssignCards, it assigns exactly one card to a player and thus the termination check in line 25 will eventually succeed (and recursively succeed until all recursive calls have been terminated). As the algorithm always first assigns cards that can only be uniquely assigned to a player and for the remaining cards, it assigns them strictly at random, it obvious that whenever the algorithm terminates, a correct and consistent card assignment was generated. Furthermore, for the same reason (the way of constructing the card assignments), each legal card assignment (ignoring double cards and card positions) can be generated with a strictly positive probability. Still, the produced card assignments are not generated uniformly at random. Generally speaking, this is due to the fact that the probabilities used in the algorithm are not calculated according to the relation of the number of times a card is assigned to a specific player divided by the number of consistent card assignments. More specific, already the fact that the algorithm at hand does not consider card slots but only players to assign cards to does not take into account the different probabilities of a card to be assigned to a player depending on how many cards this player still needs. This can be demonstrated with the help of the following small example: there are four cards left to be assigned to the three players other than the UCT player himself. One of them must be uniquely assigned to a player, thus there are three cards left for two players, one of whom (say player 1) needs to get two cards and the other one (say player 2) just

needs one. All three cards can be assigned to both players. The first card the algorithm assigns should be assigned to player 2 with probability of one third (in two out of six possible assignments) and to player 1 with probability of two thirds (in four out of six possible assignments). As the algorithm just considers the players and not the actual card slots, it will assign the card to player 1 or 2 with an equal probability of one half.

Of course, the algorithm could be adapted to circumvent this problem by also considering the number of cards a player needs rather then only the set of possible players a card can be assigned to. Still the card assignment produced would not be uniformly random because assigning one or two ♣Q to a player who still needs a ♣Q ignores the probabilities of the cards being assigned to players. The above example only needs to be slightly adapted to demonstrate this fact: again there are three cards which need to be assigned to two players, but this time two of the cards are a ♣Q and player 1, who needs to get two cards, must get a ♣Q assigned. As there are no more cards to be uniquely assigned to a specific player, the algorithm will now reach line 45 and assign a ♣Q to player 1. Afterwards, there are two cards remaining which will be assigned to player 1 and 2 with an equal probability of one half. This means that player 1 ends up having both ♣Q or just one ♣Q and the other card with an overall probability of one half each. Similarly, player 2 gets a ♣Q or the third card with probability of one half each. Clearly, this is again "wrong" in the sense that player 2 should have the third card with a probability of one third only and a ♣Q with a probability of two thirds. The problem in this case is the fact that player 1 will anyway end up with a ♣Q, and thus the algorithm would not need to take care of assigning it first to that player, but as it does so, the correct probabilities are not respected.

5 Implementation

This chapter is intended to describe how the doppelkopf framework and the UCT algorithm for doppelkopf is implemented. Both the framework and the UCT algorithm incorporate some subtleties; the framework implements doppelkopf in a way that slightly differs from the official rules at some points (still without changing the semantics of the game but only some of the procedures before the card play begins). The UCT algorithm could generally be implemented in many different ways and was implemented with several options which will be described. The first section illustrates the implementation of the doppelkopf framework and the second section explains the different UCT versions implemented for the framework. An additional third section summarizes all available options for the complete program.

5.1 Doppelkopf framework

There exist a few commercial doppelkopf programs, but I only found one open source implementation for doppelkopf, called *FreeDoko*[1]. It is written in C++ and implements a GUI for playing with human players, an AI player who plays based on rules and heuristics (with a huge amount of options that can be configured). As I was only interested in the official rules and as the open source project has a huge amount of sources, I decided to start a new implementation of doppelkopf from scratch. The disadvantage with this approach is that comparing my program with an existing implementation was not possible. In exchange I was free to choose my own implementation details and programming style which made it easier to adopt the UCT algorithm in a convenient way. I also resigned to implement a GUI but preferred to write a console-based program, as the main purpose of writing the program was to test AI players based on the UCT algorithm and not to be played by human players.

The whole program is written in C++ and uses the module "program options" from the boost library[2]. It compiles with the gnu g++ compiler[3] under linux[4].

[1] http://free-doko.sourceforge.net/en/

[2] http://www.boost.org

[3] http://gcc.gnu.org/

[4] There might be some problems when using either a boost version which is too old (lower than 1.40.0) or a g++ version which is too new (problems occurred when using version 4.6); versions 4.3 and 4.4 should work.

5.1.1 General design and structure

In the following, I will describe the design of the architecture and the cooperation of different parts and modules of the program while always keeping a certain level of abstraction. Particularly, I will not describe the architecture itself, i.e. I will not write about specific class names, class diagrams or similar details. Furthermore, as stated in the introduction of this chapter, I explain the process of the game as implemented in the framework, because it is not clear how the official rules should be implemented (e.g. human players can announce at any moment as long they are allowed to; a computer program cannot but has to be asked at discrete time steps).

The general design of the framework is oriented towards a *client-server structure*. Thus, the intent is that the program can be extended easily to have network support and be run on a server. It also makes the structure of the program modular, as the core of the program which implements the game of doppelkopf and its rules (from now on called the *game module*) is strictly separated from the players who play against each other using the program. The interface via which the game and the players communicate works as follows: when starting a new session, all players are first informed about the chosen game options (e.g. number of games, with compulsory solo or not etc.). Then each time before a new game of the session is started, the game module "deals the cards" in the sense that each player gets informed about which cards he got assigned to. The game itself is then played as follows: the game module asks the player at turn to make a move by sending him a vector containing all legal moves and asking him to return an index to the position of the vector which corresponds to the chosen move by the player. A move in this case covers all cases of game moves of doppelkopf, i.e. it can be a move to determine game types at the beginning as well as a card or an announcement move. The advantage of sending the player all legal moves he is allowed to make over asking him to return any move is that the program avoids to check if the move returned by the player is legal or not. This would need to be done if the player was free to submit any move he could think of. The only thing the game module needs to do is to check that the index returned is indeed valid, i.e. it lies in the range of the size of the vector. Another advantage is that the player can be totally ignorant towards the game of doppelkopf, i.e. a random player can just return a random index. After the game module received a valid index of the player at turn, it sends the chosen move to all players. When the game end is reached, the game module determines the winner(s) and calculates the score points. It then informs the players about the score points each player got.

5.1.2 Game module and game process

The game module constitutes the "game master" so to say, i.e. it operates the whole session and communicates with the players who on their part cannot and should not communicate between themselves. This subsection describes the whole game process and the functionality of the game module. As stated above, the game module manages the session with all chosen program options and starts a game by dealing cards and assigning them to the players. The game type determination is the most complicated part of the

game process: the game module starts by checking if the number of remaining games equals the number of open compulsory solos, in which case the player on first position is forced to play his compulsory solo (the so called "Vorführung" or exhibition, see Subsection 2.2.3). Otherwise, the player positioned first with an open compulsory solo is determined and asked for an "immediate solo", i.e. he is asked if he wants to immediately announce a solo (because his solo would have highest priority), thus shortening the reservation procedure. If there is no such player (because all players already played their compulsory solo or because the chosen options disable compulsory solos), then just the player on first position is asked if he wants to shorten the reservation procedure. If he answers with "yes", then in the next move he will be asked for a game type and then the card play starts. Otherwise, the program asks all players if they have a reservation, starting again with the player on first position. Every time a player answers "no" (i.e. he is healthy) and there is still no player who has a reservation, another player may be allowed to shorten the reservation procedure because he now knows that his solo would have the highest priority. Again, if this player wants to do so, he will be asked to set a game type and the card play will start. Otherwise, the next player (positioned after the last player who was asked for a reservation) will be asked for a regular reservation. This is done until all players have been asked for a reservation or a game type (a solo) has been prematurely determined.

The following example illustrates this non-trivial procedure: player 1 is on the first position, player 2 and 4 did not play their compulsory solo yet but player 1 and 3 already did. Player 2 will thus be the first player to be asked if he wants to shorten the reservation procedure. If he does not want to, then starting with player 1, players will be asked for regular reservations. If player 1 and 2 deny a regular reservation, then the game module must ask player 4 next if he wants to shorten the reservation procedure, because now that he knows that the only other player with a higher priority than himself does not have a reservation, he knows that his solo has highest priority. Of course, if also player 4 is healthy, then player 3 gets asked for a regular reservation and finally player 4 will be asked.

After the players have been asked for a reservation and there is none, a regular game will be played, i.e. the game module advances to card play which is described later. Otherwise, the players need to be asked if they want to play a solo or not (in the latter case they can only have a reservation because they want to play a marriage). This is where the implemented procedure slightly differs from the one stated in the official rules: in order to not to have to permanently check after each response of a player if another player now could immediately announce his solo, players are allowed to shorten the reservation procedure at most once, i.e. during the procedure described above, while players say if they have a reservation or not. Note that it is reasonable to assume that if a player did not want to immediately announce his solo the first time he could do, he also will not do so the second or third time he is asked.

The procedure to ask players if they want to play a solo or not works as follows: starting with the first-positioned player, each player is asked. If a player answers "yes" and he has the highest priority, then the asking procedure is stopped and the player is admitted. The player has the highest priority if he either did not play a compulsory

solo yet (and thus is the first-positioned player with an open compulsory solo), if all players with an open compulsory solo either did not have a reservation or said "no" to the question if they want to play a solo or not or if all players with a reservation positioned somewhere behind the player already played their compulsory solo and thus he is also allowed for a lust solo. Otherwise, if a player answers "no", he can only be admitted if he is the only one with a reservation. In this case, the player can announce his marriage. Note that this procedure is slightly different from the one described in the official rules, as the game module does not first explicitly ask if players want to play a compulsory solo and then asks for lust solos, but it just asks them if they want to play a solo or not. This means that in some cases a player must reveal if he intended to play a lust solo or not where the same procedure implemented strictly according to the official rules would not have forced the same player to reveal it, because he would never be asked if a player with a compulsory solo positioned somewhere behind him decided to play a solo before. I think that the "information gain" for the compulsory solo player in this case is sufficiently small to be neglected.

After the game type has been determined, the game module starts the card playing part of the game, which means that in all cases except for a marriage where announcing is not possible before the clarification trick, the players are first asked for announcements and then for playing a card. In a doppelkopf game played with humans, players can "always" make an announcement as long as they are allowed to, which is obviously not a reasonable choice for a doppelkopf program. The implementation I chose therefore allows players to make an announcement in the order of positions after a card has been played. Also before the very first card is played, all four players are asked for an announcement. The player who just played a card is not asked for an announcement afterwards, as he was asked before he played a card and thus his information did not change. Thus, after the first card has been played, there are always only three players being asked for an announcement. If there is no announcement done during such a "round" of players being asked, then the next player who needs to play a card is asked to do so. Otherwise, as soon as an announcement is done, all other three players need to be asked for an announcement (again).

There are two interesting implementation details for the card play and announcement procedures: first, there exist two versions for the way the announcing process is modeled, configurable via program options. The first version asks a player to choose between all legal announcements, i.e. the player can immediately announce black if he likes to. The second version only asks if the player wants to make an announcement or not, and if the player answers "yes", the announcement done corresponds to the "next higher" announcement for the player's team. If for example a player of the re-team announced re and the second re-player answers "yes" to the question, this will be interpreted as an announcement of no 90. In order to still allow players to "skip" announcements, a player who answered "yes" will be asked again immediately, even before all other three players are asked. The reason the second version was added to the first one was the hope of reducing the branching factor for nodes in the game tree in order to enhance the performance of the UCT algorithm. This is discussed more in the chapter about experiments. The second implementation detail worth to notice is the following: when

the game module determines if a player is allowed to make an announcement or not, i.e. when the game module determines if a player has to be asked for an announcement or not at all, then it can happen that players are asked for an announcement even if their only option is to answer "no". The reason behind this is to prevent to leak information about a player's team. This becomes clear in the following example: a regular game is played, a re-player announced no 60, no ♣Q has been played so far and no other announcements have been done. The latest moment for all players for an announcement or a reply in case of the two kontra-players is while still holding at least nine cards. Now one of the kontra-players announces kontra as a reply, e.g. while holding ten cards. Now both kontra-players cannot do any further announcement, but as the re-player who announced no 60 cannot know who of the two players is re and who is kontra, the game module should ask both players for announcements, even if the kontra-player's only option is to answer "no". Otherwise, the program would reveal the second kontra-player. The player who announced kontra as a reply does not need to be asked again, because all players know he is kontra and thus he cannot do any further announcements. The same holds for a player who announces black. To summarize, as long as the players' teams in a regular game are not known to *all* players, the game module keeps track of the latest possible moment for announcements for each player separately, avoiding to leak any information about a player's membership of a team. As soon as the teams are known, the latest moment for an announcement possibly needs to be updated for some players and from then on, the latest moment for an announcement of the player's team is considered when checking if he is still allowed to make an announcement or not.

As noted above, when the game is finished, the game module computes the score points and broadcasts them to all players who are thus able of keeping track of the overall standings. It also needs to do a few other things such as updating the player who is positioned first in the next game, checking if the last game was a compulsory solo and possibly updating the according data structures and checking if the next game needs to be a "Vorführung" or not.

5.2 UCT algorithm

Players using the doppelkopf program described above must of course implement the mentioned interface, i.e. they must be able to accept a set of cards, a vector of moves from which a chosen index has to be returned, the chosen move by the current player and the score points after a game is finished. A player who wants to use the UCT algorithm must keep track of game process and needs to collect information from it. In other words, a UCT player needs to have a "belief game state", i.e. a game state with imperfect information from the player's specific point of view. The task for a UCT player is to compute a move that is "as good as possible" when asked by the game module to choose a move. As the application of the UCT algorithm presumes the availability of a world model to generate perfect information worlds, the UCT player also needs to fill in the missing information about other players' hands. As stated in Chapter 4, the chosen solution is to use an algorithm for computing card distributions given the current belief

game state of that player. The UCT player implemented for the doppelkopf program uses the card assignment algorithm presented in Section 4.3. With the help of the generated worlds, a UCT player is then able to transform his belief game state into a "concrete game state", which then can be used during the MCTS of the UCT algorithm, where nodes (which correspond to states) must be expanded. In the implementation, the UCT player re-uses parts of the game state implementation of the game module.

The work flow of a UCT player then is the following: when the session is started, the player receives the chosen options. Each time a new game is started, the UCT player needs to start keeping track of the game process by maintaining a belief game state. Every time he is asked to make a move, he uses the UCT algorithm to compute the best move. Depending on the selected version of the UCT algorithm and other options, several simulations and/or rollouts are performed, each using many different card assignments created with the help of the card assignment algorithm.

I implemented two versions of the UCT algorithm; they do not differ greatly, but still the results sometimes show rather big differences, see Chapter 6, Section 6.2. The first version of the UCT algorithm uses "simulations" in addition to the common rollouts; where a simulation consists of several rollouts. When started, a card assignment is computed and fixed for each simulation. All rollouts performed during one simulation then use the same card assignment to build a search tree. As soon as a leaf node with an unvisited successor is encountered, one of the successors is chosen and added to the tree. An option for the UCT player allows to always choose the first index, a random index or, if the action is a card move, to choose a "safe card" which guarantees that he or his teammate wins the trick if he has such a card (see Section 5.3). The newly created leaf node is then added to the tree. From this point on, the algorithm updates the current state with one of the legal moves until a terminal state is reached. An option allows to choose how this is done: the first version performs an MC simulation where only the current state gets updated, but no new nodes are created and added to the tree. A second version continues the tree search and adds a new node for each state encountered until the terminal state is reached. The latter version needs more memory but may sometimes allow to go further into the depth of the tree, as more nodes are added to the tree during each rollout and thus the chance of re-visiting a node during a future rollout is increased. On the other hand, it is not very probable to reach the same newly created leaf node again so that the algorithm could profit from the nodes that have been added on the path from the leaf node to a terminal state. Thus, both variants seem to be worth being examined. In both cases, when a terminal state (or a terminal node corresponding to a terminal state) is reached, the game value is computed and transferred into UCT rewards for players. How this is done exactly depends on several options and is described below. During the back-propagation of the UCT rewards in the UCT algorithm, rewards are added up at all nodes along the path from the last node added to the tree (this is the node corresponding to the terminal state if all nodes are added to the tree or it is the single newly created leaf node added to the tree before the MC simulation was started) up to the root node and the visit counters are increased. When all rollouts for the current simulation are finished, the average rewards of all successors of the root node are computed and summed up over all simulations.

Note that the successors of the root node correspond to the application of the possible moves the UCT player has and that they are always the same, independent of the card assignments. Parallel, a counter for each possible move at the root node is increased by one for the move which would be chosen if only the one simulation was taken into account. Each simulation constructs a new search tree; no information from previous simulations is re-used. When all simulations have been computed, the sum of the average rewards is normalized by dividing the values through the number of simulations[5]. The move with the maximal normalized summed up average reward sometimes differs from the move which would have been chosen in majority of simulations, but the summed up average rewards yield better performance in some informal experiments and is thus the criterion used to determine the best move.

The second version of the UCT algorithm implemented for the UCT players is solely based on rollouts; no simulations are performed and only one search tree is constructed over time. For each rollout, a new card assignment is used. As a consequence, nodes in the game tree from previous rollouts may become inconsistent with the current card assignment, because obviously at a player's node where he has to play a card, the possible successors will differ from rollout to rollout with different card assignments for this player. Thus this version of the UCT algorithm has to deal with information sets rather than single states. However this can be easily transferred back to the more simple case of nodes corresponding to game states as in the first version of the UCT algorithm: during a rollout, all successors of a node which are not consistent with the current assignment, i.e. for which there exists no action that leads to that node, are ignored. All other reachable consistent nodes always contain the identical information during different rollouts with one small exception to this rule: in a rare case, it can happen that a node reached again in a different rollout corresponds to a game state which only differs in the next player to move. The only reason for this to happen is the following: if player A announces black at a moment before that node is encountered in the game tree, then the teammate of player A is not allowed to make any more announcements, assuming that teams are known for all players because both ♣Q have been played or because other announcements have been done. Now, suppose a player plays a card, which generates a successor node where the player positioned behind him, let it be player B, has to be asked for an announcement next. Player B in that case it not the teammate of player A and thus allowed to make an announcement. In a future rollout with a different card assignment (because in the actual game, teams are *not* known yet and thus different players may have a ♣Q in different card assignments), player A's teammate may be player B, and thus the "same" node is reached by exactly the same game process, but a different player has to move, because player B cannot be asked for an announcement (again assuming the teams are known to the players in which case players who obviously cannot do any further announcements are not asked anymore). This version of the UCT algorithm thus also needs to check if the same player

[5]The normalization would not be necessary, but the normalized numbers better reflect the average rewards of a single simulation (and are better comparable to the second version of the UCT algorithm)

has to move next when testing successors of a node for consistency with the current card assignment. When the algorithm reaches a node where some of the legal moves have not been explored, one needs to be chosen and the corresponding node will be added to the tree. This and everything else which follows afterwards (i.e. MC simulation or continued tree search, computation of UCT rewards and back propagation) is done in the same way as in the first version of the UCT algorithm and thus depends on the chosen settings. The final determination of the best move is different, as there are not several simulations that need to be aggregated. Instead, the algorithm just returns the successor which has the highest average accumulated reward for the UCT player.

UCT rewards for each player are computed from the game value as follows: the player's score points get multiplied by a constant configurable via program options. In order to also consider the playing points and not only the score points (assuming that a player should always prefer reaching 149 over 121 points, although the result in the means of score points is the same), the playing points, divided by another constant which can be set via the program options, are added to the previously calculated value. Another option allows to toggle to use either the player's points or the team points of a player. When traversing the search tree, the UCT formula (as described in Section 3.3) needs to be applied if all successors of a node have already been visited. The exploration term of the UCT formula can also be configured via the program options by choosing a value for the exploration constant. Another option concerning the exploration term results from a bug in an early implementation, where instead of the number of visits of the current node, the number of the current rollout (i.e. the number of total visits at the root node so far) was used for the nominator n in the exploration term

$$c_{i,n} = \sqrt{\frac{\log n}{T_i(n)}}$$

Thus the user can choose to use this "wrong" UCT formula or the normal one; the reason for this is that some informal local experiments have shown that the wrong formula sometimes even produces better results.

5.3 Summary of program options

So far, many program options have been mentioned throughout this chapter. This section provides a summary of all relevant options for the game model and the UCT algorithm. Note that the default values may seem to be chosen arbitrarily, but they were selected according to many experiments and reflect the values for the best configuration found, see also Chapter 6. The program supports a few more options which are related to debugging or implementation details and which are not listed here.

- Number of games: specify the number of games to be played. The default value is 1000 which is not the number of games played in an official tournament (24), because results differed a lot when only playing 24 games.

- Compulsory solo: specify if the players have to play a compulsory solo or not. The default is FALSE, as any decent player will play at least one solo during a set of 1000 games and thus there is no need to force players to do so.
- Announcement style version: specify if the first or second version for announcing should be used. The first version allows a player to choose an announcement from all legal announcements whereas the second version only asks a player if he wants to make an announcement and if necessary infers the announcement automatically and then asks the same player again for a further announcement. The default is to use the second version.
- Player types: each player type can be specified to be a UCT, random or human player. The default is UCT for the first player and random for the other three players.
- Player options for UCT players:
  - Version: specify if the player should use the first or the second version of the UCT algorithm. The default is to use the second version.
  - Score points factor: specify the constant which is multiplied to the score points when computing the UCT rewards. The default value is 500.
  - Playing points: specify if the bias term added to the score points should be calculated based on the player's points or the team points of the player. The default is to use the team points.
  - Playing points divisor: specify the constant which the playing points are divided by before being added to the score points. The default is 1, i.e. the playing points are not modified. It is only reasonable to use this option if the value for the score points constant is smaller than its default value or values similar to it, and thus the playing points would have a too big impact on the UCT rewards compared to the score points.
  - Exploration constant: specify a value for the exploration constant of the UCT formula. The default value is 20000.
  - Number of rollouts: specify the number of rollouts to be used in the UCT algorithm, no matter which version. This means this value is the number of rollouts per simulation for the first UCT version and the the number of total rollouts for the second UCT version. The default value is 1000 as the default version is the second UCT version. When using the first UCT version, this should be adapted to be 100 (because the default value for the number of simulations is 10) so that both versions use 1000 rollouts in total.
  - Number of simulations: specify the number of simulations that should be used in the UCT algorithm. The default value is 10. This option only has an effect if the first version of the UCT algorithm is used.

- Announcing rule: there are three supported options: the first one is to allow the player to do regular announcements. The second one only allows the player to make an announcement if the average reward for the corresponding move in the game tree is not negative. The reason for this option is that the analysis of the game process of some local experiments have shown that especially the second version of the UCT algorithm tends to over-announce, because apparently this version of the UCT algorithm has difficulties to differentiate between several "bad" choices. The third and last option is to completely forbid any announcing for the player. The default is to allow only allow announcing if the corresponding UCT rewards are positive.
- UCT formula: specify if the UCT formula should be calculated in the normal correct way or in the "wrong" way described above, where the number of visits of the node in question is replaced by the number of the current rollout. The default is to use the correct UCT formula.
- MC simulation: specify if no MC simulation should be carried out but all states encountered during a rollout should be added as nodes to the tree, i.e. more than one per rollout or if an MC simulation should be carried out as soon as a leaf node was added to the tree, i.e. only one node is added to the tree per rollout. The default is to use no MC simulation.
- Action selection: there are two moments where a successor needs to be chosen out of a set of unvisited ones: the first one is when adding the first node of the current rollout (which always needs to be done no matter if an MC simulation is carried out or not) and the second one is during the remaining tree search or MC simulation, depending on the chosen option, thus the differentiation between those two otherwise similar decisions.
  There are five versions combining different action selection policies for the two decisions: the first one chooses the first successor when adding the first node and chooses a random successor during the continued tree search or the MC simulation. The second version uses random action selection in both cases. The third version again chooses the first successor when adding the first node of the rollout and from then on uses a heuristic guided action selection (see below) for card moves and chooses a random successor if the move is of any other type. The fourth version uses random move selection for the first node and the heuristic choice for the tree search or the MC simulation and the last version uses the heuristic approach in all cases. The default is to use the first version.

The *heuristic* for action selection is based on playing "safe cards": If there are already cards in the trick, then the heuristic checks if a teammate of the player wins the trick so far. If so, it needs to further check if any of the opponents can play a higher card. If not, then the heuristic chooses a valuable card of the legal cards of the player, preferring to a valuable non-trump card over playing a valuable low ranked trump over playing a valuable high ranked trump card (i.e. ♡10). If the trick is not safely owned by a

teammate or owned by an opponent, then the following procedure applies: the heuristic checks if the player can play a non-trump card so that nobody can play a higher card (or trump it). If this is not possible for non-trump suits, then the heuristic checks if the player can play a non-trump card of a suit where his teammate can win the trick (also considering the positioning of players, i.e. it is not enough to check if the teammate has the highest card of that suit or if he can trump it, but he also needs to be able to play it before a player of the other team can play the same highest card). If also this is not possible, then the heuristic makes the same checks for the trump suit: if the player can play a high trump card which wins the trick for sure, then this card is chosen, otherwise, if a teammate holds a high trump card then can secure the trick, then a low valuable trump should be played. If none of these options are possible, then the heuristic chooses a random card.

6 Experiments

This chapter presents experiments made with the doppelkopf program described in the previous chapter. The first section shows experiments that served to find reasonable parameter values for a baseline UCT player for both UCT versions. This baseline UCT player uses the most "simple" settings possible, especially concerning the player options. The values to be tested include the parameters for the UCT formula and the computation of UCT rewards out of the game result. The second section is dedicated to comparing the two different implementations of the UCT algorithm. The third section is dedicated to testing all important player options of the doppelkopf program. For this purpose, the baseline UCT player is compared to a UCT player that modifies the parameter that needs to be tested and leaves everything else as in the baseline version. Depending on the domain of the parameter, this may result in several combinations being tested. The fourth section presents results of experiments making changes to several parameters at a time, thus combining some of the experiments from the third section. The goal is to test if combining several configurations that have been proven to be good can still yield an overall improvement. The fifth section is intended to analyze the profits of increasing the number of samples for the quality of results and the effect of increasing the number of games played on the reliability of the results. Also time and memory usage will be discussed. The final section shows the result of a tournament of the best UCT player configuration found against a human player.

The general settings for the experiments are the following, unless mentioned differently for specific cases: as 24 games as played in a standard tournament is not a lot when players are influenced by random events such as card dealing, random players or random choices during the execution of the UCT algorithm, a standard experiment consists of 1000 games. Players do not have to play compulsory solos either when playing 1000 games; experiments have shown that the UCT players play "well enough" to voluntarily play a solo from time to time, so there is no need to force them. Especially, forcing a random player to play a solo nearly always results in a big loss for that player. Whenever comparing two different values for a parameter, usually two UCT players adapting the different values play against each other, together with two random players. Additionally, the two UCT players also play alone against three random players to see the direct performance without the influence of another UCT player. The number of rollouts was chosen to be 1000 for both UCT versions, i.e. 10 simulations at 100 rollouts each for the first version and 1000 rollouts for the second version. This value is neither too low to produce poor results nor high enough to make the computation last several days for one set of 1000 games. The standard values for the following experiments are summarized in Tables 6.1 and 6.2.

Both the influence of raising the number of samples used for the algorithm, i.e. the

General Options	
Number of games:	1000
Use compulsory solo?:	no
Use random cards?:	yes
Random seed for card dealing:	2012

Table 6.1: General settings for all experiments of this chapter unless stated differently

Player Options: UCT version 1	
Number of rollouts:	100
Number of simulations:	10
Player Options: UCT version 2	
Number of rollouts:	1000

Table 6.2: Players' settings for all experiments of this chapter unless stated differently

(total) number of rollouts, and the influence of changing the number of games played in an experiment will be investigated in a extra section. For now I assume that using more samples up to a certain amount will increase the quality of results and increasing the number of games played will increase statistical reliability of results. Thus these parameters will be constant for all tests done in Section 6.3, as this section is supposed to test the influence of UCT player parameters that directly concern the character of the UCT algorithm.

6.1 Tuning parameters to obtain a good baseline UCT player

This section presents experiments that were made to find some reasonable values for baseline players for both UCT versions. The baseline players use the simplest settings for the UCT algorithm, i.e. all player specific options are set to their default values: players' playing points rather than their team points are used to bias the score points when computing the UCT rewards, a player is allowed to make announcements (and not only if the average rewards are positive or even none at all), players use the "wrong" UCT formula (as this was the first version implemented), an MC simulation is carried out rather than expanding the tree by more nodes than one per rollout and when choosing a successor out of a set of unvisited successors, the first one is chosen before adding the node to the tree and a random one is chosen during the MC simulation (i.e. the first version of the action selection option is used). The only relevant global option that concerns all players at the same time is the announcement style version used. As some experiments showed some problems concerning the version where a player can freely choose from all legal announcements rather than only saying "yes" or "no", both versions will be tested already in this section.

The remaining parameters which should thus be tested concern the computation of

the UCT rewards out of the game result and the computation of the UCT formula. More precisely, they include:

- Score points factor
- Playing points divisor
- Exploration constant

Note that the score points factor was chosen to be at least 500 in all experiments, i.e. big enough that the playing points added to the scaled score points do not need to be made smaller by dividing by a constant, thus the playing points divisor was set to be 1 for all experiments. Also note that choosing a score points factor of 1000, an exploration constant of 1000 and a playing points divisor of 1 is exactly the same as choosing 1, 1 and 1000 for the three parameters respectively.

Table 6.3 summarizes the player options which are fixed for experiments of this section.

Player Options	
Playing points divisor:	1
Use team points instead of player's points?:	no
Allow announcements?:	yes
Use wrong UCT formula?:	yes
Do an MC simulation?:	yes
Action selection:	version 1

Table 6.3: Players' settings for all experiments of this section

6.1.1 First announcement style version

This first subsection is dedicated to experiments where the first announcement style version is used, i.e. players are allowed to choose from all legal announcements and can not only say "yes" or "no" when asked if they want to make an announcement. As the domains of the tested parameters are integer valued, there are too many combinations of values to test many of them when playing against each other. Therefore, the first series of tests consist of choosing some parameter settings that seem to be appropriate and then letting a UCT player with those settings play against three random players. The obtained results are of course of less importance compared to results where a UCT player with a certain setting of parameters directly plays together with another UCT player. Still, this first series serves to determine a set of reasonable parameter settings which can then be used in a second series where UCT players play against each other.

First UCT version

Table 6.4 shows the results of a UCT player using the first UCT version playing together with three random players. The first column shows the values used for the exploration

	500	1000	2000
500	9.07 ± 0.94	9.43 ± 0.93	-
1000	9.23 ± 0.92	8.98 ± 0.95	9.18 ± 0.95
1500	9.45 ± 1.03	9.77 ± 0.90	8.83 ± 0.95
2000	-	-	8.74 ± 1.00
2500	-	-	8.78 ± 0.94
3000	-	9.42 ± 0.88	-
4000	10.44 ± 1.05	-	9.98 ± 0.93
4500	11.06 ± 1.13	-	-
5000	10.32 ± 1.08	9.80 ± 1.00	-
7000	-	10.67 ± 1.07	9.52 ± 1.00
8000	11.10 ± 1.29	-	-
9000	-	10.68 ± 1.07	-
10000	10.66 ± 1.36	11.10 ± 1.17	9.32 ± 1.00
15000	-	-	10.27 ± 1.07

Table 6.4: Results of a UCT player playing against three random players, using the first UCT version and different values for the score points constant (columns) and the exploration constant (rows). The values shown are the 95% confidence intervals for the average score points for a round, i.e. multiplying the value with 250 yields the concrete score points the UCT player achieved in the experiment.

constant and the first row shows the values used for the score points constant. The values shown are the 95% confidence intervals for the average score points for a round (i.e. four games) of the UCT player. The scores of the random players (always negative) have been omitted, as have been memory and time usage of each single experiment, as the values are very similar for each setup. The best three configurations (with very similar results) are the combinations 500/4500, 500/8000 and 1000/10000 for the score points constant/exploration constant. Apparently, increasing the exploration constant while keeping the score points constant at the same value increases the player's performance up to a certain level. This can be observed for all three tested values of the score points constant (500, 1000 and 2000). Also if the score points constant is doubled (e.g. from 500 to 1000), it seems to be appropriate to also approximately double the exploration constant at the same time in order to achieve a similar good result. Thus the relation between the two values has a fairly high influence on the results. As a consequence, in order to avoid to overload this chapter with tables, I will only show results of experiments where a value of 500 for the score points constant has been chosen. Similar results can of course also be obtained by testing other (higher) values, but then also the exploration constant must be increased accordingly. This has been confirmed in other experiments which are omitted in order to avoid putting even more tables into this chapter.

Table 6.5 shows some experiments where the configuration 500/4500 found above is tested against some very similar players. The results are very astonishing in the sense that the second UCT player outperforms the first UCT player no matter which

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	4500	4500	4500	4500	4500
	Player 2 Options				
SP factor:	400	500	500	500	600
Expl const.:	4500	3500	4500	5500	4500
	95% confidence intervals for average round scores				
P1 (UCT):	-1.77 ± 2.13	-0.90 ± 2.14	-2.30 ± 2.03	-1.62 ± 2.26	-1.05 ± 2.07
P2 (UCT):	12.22 ± 2.08	10.77 ± 1.84	12.43 ± 1.98	11.49 ± 2.15	11.42 ± 1.96
P3 (Rnd):	-4.34 ± 2.07	-4.07 ± 1.95	-4.55 ± 1.85	-5.25 ± 2.02	-4.79 ± 1.90
P4 (Rnd):	-6.12 ± 2.04	-5.80 ± 2.01	-5.58 ± 1.94	-4.62 ± 1.94	-5.58 ± 2.00

Table 6.5: Two UCT version 1 players positioned next to each other

configuration is used. These huge differences cannot solely be explained by the only slightly differing configuration. Suspecting the reason for the results to be the positioning of the players, the same experiments have been conducted again, but with switched positions for the UCT players. The results are depicted in Table 6.6.

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	4500	3500	4500	5500	4500
	Player 2 Options				
SP factor:	500	500	500	500	500
Expl const.:	4500	4500	4500	4500	4500
	95% confidence intervals for average round scores				
P1 (UCT):	-4.50 ± 2.21	0.32 ± 1.90	-2.30 ± 2.03	-2.99 ± 2.21	0.74 ± 1.98
P2 (UCT):	13.90 ± 1.95	11.14 ± 2.01	12.43 ± 1.98	12.37 ± 2.03	10.96 ± 2.02
P3 (Rnd):	-4.41 ± 2.04	-5.96 ± 1.83	-4.55 ± 1.85	-3.52 ± 1.88	-6.07 ± 1.79
P4 (Rnd):	-4.99 ± 2.04	-5.50 ± 1.91	-5.58 ± 1.94	-5.87 ± 2.15	-5.62 ± 1.87

Table 6.6: Two UCT version 1 players positioned next to each other with inversed positions

Clearly, the positions of the UCT players have a huge influence, as switching positions did not change any of the observed behavior. Apparently, one can generally say that a UCT player placed in front of another UCT player gets exploited. Some informal local experiments have confirmed that no matter which concrete position players are placed on, a UCT player positioned in front of another UCT player always will get exploited; also when three UCT players and only one random player are playing, the UCT player positioned after the random player will always end up being the worst. The crucial advantage that the UCT player placed behind the first UCT player has is that he can always play cards *after* the first UCT player played a card, with the exception of the cases where the second UCT player starts a trick. Thus he can quite easily exploit any "mistakes" the first player does. This effect is even reinforced when the first UCT

player tends to over-announce, which is the case when using the first announcement style version, as seen when analyzing in more details the game process of some local experiments. Consequently, experiments with a setup where two UCT players are placed next to each other are of a very small significance and therefore the experiments have been repeated with a positioning where the two UCT players are placed vis-à-vis, with a random player positioned before and after each of them.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	4500	4500	4500	4500	4500
	Player 3 Options				
SP factor:	400	500	500	500	600
Expl const.:	4500	3500	4500	5500	4500
	95% confidence intervals for average round scores				
P1 (UCT):	9.50 ± 2.21	6.84 ± 2.23	8.19 ± 2.00	10.91 ± 2.17	8.34 ± 2.02
P2 (Rnd):	-9.57 ± 1.84	-8.17 ± 1.90	-9.30 ± 1.81	-10.15 ± 1.90	-9.32 ± 1.89
P3 (UCT):	7.63 ± 2.39	9.67 ± 2.10	8.12 ± 2.10	8.04 ± 2.41	8.71 ± 2.02
P4 (Rnd):	-7.57 ± 1.89	-8.34 ± 1.80	-7.01 ± 1.85	-8.80 ± 1.94	-7.73 ± 1.89

Table 6.7: Two UCT version 1 players positioned vis-à-vis

The results shown in Table 6.7 demonstrate very well that the effect of exploiting due to the positioning of UCT players is not existent in this setup, as two identical UCT players that play together with two random players nearly yield the same result, as can be seen in the middle column. Also deviating from the chosen configuration (500/4500) seems not to be good except for the configuration 500/3500 and also 600/4500 seems to be slightly favored. Both cases indicate that the exploration constant in relation to the score points constant was chosen slightly too big, possibly 500/4000 would be a better choice.

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	4500	3500	4500	5500	4500
	Player 3 Options				
SP factor:	500	500	500	500	500
Expl const.:	4500	4500	4500	4500	4500
	95% confidence intervals for average round scores				
P1 (UCT):	7.55 ± 2.20	9.09 ± 2.02	8.19 ± 2.00	8.06 ± 2.13	8.69 ± 1.87
P2 (Rnd):	-10.60 ± 1.88	-9.65 ± 1.96	-9.30 ± 1.81	-9.85 ± 1.93	-10.08 ± 1.86
P3 (UCT):	9.18 ± 2.21	8.17 ± 2.23	8.12 ± 2.10	8.84 ± 2.17	8.84 ± 2.14
P4 (Rnd):	-6.13 ± 1.84	-7.61 ± 1.91	-7.01 ± 1.85	-7.05 ± 1.90	-7.45 ± 1.76

Table 6.8: Two UCT version 1 players positioned vis-à-vis with inversed positions

To be sure that the positions really have no or only a little influence when placing UCT players vis-à-vis, the experiment is repeated with inversed positions for the UCT

players. The results are shown in Table 6.8. They seem to be consistent in the way that if the UCT player placed on position one in the first experiment wins, then he also wins when placed on the third position. There is one exception: the configuration 600/4500 (last column) wins in the first setup but is a little worse than the other UCT player in the second setup. However, the average round score points are very close to each other, so that the difference can be caused by the different hands the players have during one experiment (this observation is also supported by the quite large and overlapping confidence intervals). When summing up the average round score points of the two experiments, the configuration 600/4500 achieves a slightly better result than the configuration 500/4500. By increasing the number of games played, this behavior (that one player wins on one position, but loses on the other one) could probably be ruled out or at least the effect could be reduced (the confidence intervals would also become smaller). There is some further evidence that the absolute positions of players indeed matter at least a little bit, because taking a look at the second column, the configuration 500/3500 wins with a difference of 2.83 average round score points over the other UCT player with the configuration 500/4500 in the first experiment, but when switching positions, the UCT player with the configuration 500/3500 only wins by a difference of 0.94 average round score points. Both of the two last observations indicate that the relation between the score points constant and the exploration constant was chosen a little bit too hight, maybe 500/4000 would be a better choice for a baseline UCT player when using the first announcement style version. Further, note that by using a vis-à-vis positioning of the two UCT players, an exploitation of the other UCT player can be avoided (due to the symmetry of the positioning), but 1000 games is obviously not enough to rule out any differences in the results which are due to the different card deals the players have to play. A solution is to increase the number of games played in an experiment, but to significantly reduce the confidence intervals, the number of games must be increased a lot and experiments would take a long time to run. Therefore, experiments where two UCT players are tested will always be repeated with inversed player positions.

Second UCT version

The same setup of experiments for UCT players using the second UCT version will be presented in the following. Thus the first series of experiments consists of letting one UCT player play together with three random players, testing different combinations for the score points constant and the exploration constant. As explained above, the results shown are restricted to those where a value of 500 is used for the score points constant and the exploration constant is varied.

Taking a look at Tables 6.9 and 6.10 and according to the results of the first UCT version seen above, one can observe that increasing the exploration constant in relation to the score points constant seems to be reasonable up to a certain amount. More generally, the differences between the results of different configurations are way more significant in these experiments compared to those of the first UCT version and also the confidence intervals are a bit larger. Thus all results obtained from the second

	Player 1 Options			
SP factor:	500	500	500	500
Expl const.:	500	1000	1500	4000
	95% confidence intervals for average round scores			
P1 (UCT):	3.96 ± 1.32	5.48 ± 1.20	6.64 ± 1.21	8.03 ± 1.51
P2 (Rnd):	-2.50 ± 1.39	-3.40 ± 1.42	-2.13 ± 1.42	-2.89 ± 1.36
P3 (Rnd):	-1.70 ± 1.40	-2.24 ± 1.31	-3.03 ± 1.27	-3.57 ± 1.34
P4 (Rnd):	0.24 ± 1.44	0.16 ± 1.32	-1.48 ± 1.39	-1.56 ± 1.30

Table 6.9: One UCT version 2 player with three random players (1)

	Player 1 Options			
SP factor:	500	500	500	500
Expl const.:	4500	5000	8000	10000
	95% confidence intervals for average round scores			
P1 (UCT):	5.96 ± 1.62	6.30 ± 1.76	0.42 ± 2.17	-4.09 ± 2.55
P2 (Rnd):	-2.80 ± 1.32	-2.96 ± 1.34	-0.45 ± 1.36	1.70 ± 1.39
P3 (Rnd):	-2.59 ± 1.22	-1.83 ± 1.37	0.06 ± 1.31	1.52 ± 1.44
P4 (Rnd):	-0.58 ± 1.33	-1.51 ± 1.44	-0.03 ± 1.26	0.86 ± 1.38

Table 6.10: One UCT version 2 player with three random players (2)

UCT version are a bit less reliable. More insights and comments on the differences between both UCT versions are given in Section 6.2. The best result is obtained by the combination 500/4000 which is thus further tested in the next series of experiments, where a UCT player using this configuration is tested in direct play together with a UCT player using a slightly different configuration. Both UCT players are positioned next to each other.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	4000	4000	4000	4000	4000
	Player 2 Options				
SP factor:	400	500	500	500	600
Expl const.:	4000	3000	4000	5000	4000
	95% confidence intervals for average round scores				
P1 (UCT):	-0.50 ± 3.01	-3.04 ± 3.09	-3.59 ± 3.30	-1.98 ± 3.20	-0.74 ± 3.37
P2 (UCT):	7.49 ± 3.28	7.19 ± 2.55	9.17 ± 3.06	9.63 ± 3.22	6.84 ± 3.09
P3 (Rnd):	-3.03 ± 2.44	-2.17 ± 2.39	-2.62 ± 2.33	-4.57 ± 2.26	-3.54 ± 2.39
P4 (Rnd):	-3.95 ± 2.34	-1.97 ± 2.31	-2.96 ± 2.22	-3.08 ± 2.28	-2.57 ± 2.30

Table 6.11: Two UCT version 2 players positioned next to each other

Tables 6.11 and 6.12 show the results of those experiments, the second one using the same setup but with inversed positions. As observed for the corresponding setup for

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	4000	3000	4000	5000	4000
	Player 2 Options				
SP factor:	500	500	500	500	500
Expl const.:	4000	4000	4000	4000	4000
	95% confidence intervals for average round scores				
P1 (UCT):	-5.74 ± 3.36	-3.90 ± 2.87	-3.59 ± 3.30	-3.47 ± 3.73	-2.30 ± 2.98
P2 (UCT):	10.57 ± 3.00	11.98 ± 3.01	9.17 ± 3.06	8.68 ± 2.95	7.90 ± 2.94
P3 (Rnd):	-3.43 ± 2.32	-3.25 ± 2.34	-2.62 ± 2.33	-2.89 ± 2.52	-2.02 ± 2.23
P4 (Rnd):	-1.39 ± 2.42	-4.83 ± 2.37	-2.96 ± 2.22	-2.32 ± 2.51	-3.58 ± 2.30

Table 6.12: Two UCT version 2 players positioned next to each other with inversed positions

the first UCT version, the second UCT player positioned directly behind the first one outperforms the first one by a large amount. This is also confirmed when switching the positions. For this reason, the experiments have been repeated with a vis-à-vis positioning.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	4000	4000	4000	4000	4000
	Player 3 Options				
SP factor:	400	500	500	500	600
Expl const.:	4000	3000	4000	5000	4000
	95% confidence intervals for average round scores				
P1 (UCT):	7.35 ± 3.36	10.87 ± 3.18	10.04 ± 3.21	9.36 ± 3.10	11.15 ± 3.23
P2 (Rnd):	-11.19 ± 2.31	-10.45 ± 2.33	-9.71 ± 2.22	-9.24 ± 2.53	-9.50 ± 2.53
P3 (UCT):	11.39 ± 3.27	6.11 ± 2.88	6.04 ± 3.30	6.48 ± 3.50	7.03 ± 3.18
P4 (Rnd):	-7.55 ± 2.38	-6.54 ± 2.32	-6.36 ± 2.54	-6.59 ± 2.23	-8.68 ± 2.51

Table 6.13: Two UCT version 2 players positioned vis-à-vis

Tables 6.13 and 6.14 show the results of those experiments, where the second table contains the results of experiments with the same setup as the first table, but with inversed positions. Even although not positioned next to each other, the results of two identical UCT players playing together with two random players yield quite different results: the first UCT player achieves 10.04 compared to 6.04 average round score points of the second UCT player, as seen in the middle column of the first (and second) table. This difference makes it hard to analyze the different configurations that only slightly differ. Still, having a look at the first and the fourth column of the first table, it seems that the relation between the exploration constant and the score points constant is not big enough, as increasing the exploration constant or lowering the score points constant of the second UCT player moves the results more in favor of this player (always compared

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	4000	3000	4000	5000	4000
	Player 3 Options				
SP factor:	500	500	500	500	500
Expl const.:	4000	4000	4000	4000	4000
	95% confidence intervals for average round scores				
P1 (UCT):	9.87 ± 3.28	7.81 ± 2.75	10.04 ± 3.21	12.93 ± 3.51	10.23 ± 2.94
P2 (Rnd):	-7.96 ± 2.41	-7.27 ± 2.34	-9.71 ± 2.22	-9.12 ± 2.32	-7.71 ± 2.42
P3 (UCT):	6.05 ± 3.33	7.78 ± 3.05	6.04 ± 3.30	4.91 ± 3.24	5.25 ± 3.09
P4 (Rnd):	-7.96 ± 2.39	-8.32 ± 2.51	-6.36 ± 2.54	-8.72 ± 2.39	-7.77 ± 2.40

Table 6.14: Two UCT version 2 players positioned vis-à-vis with inversed positions

to the results displayed in the middle column, which serves as a reference). Similarly, column two and five of the first table show that decreasing the relation of the score points constant to the exploration constant even shifts the results more in favor of the first UCT player. Taking a look at the second table, switching the positions confirms the results of the first table: in the first column, the winning configuration of the first experiment also wins after switching positions. In the second column, this is less obvious but still holds: in the first table, the configuration 500/4000 achieves slightly better than 500/3000 (comparing the values to those of the middle column) and in the second table, it largely improves its results compared to the middle "reference" column. Alternatively, adding up the average round score points of both experiments also confirms that 500/4000 is indeed the better configuration. Analog observations can also be made for the last two columns. All together, 500/4000 seems indeed to be a reasonable configuration for a baseline UCT player when using the first announcement style version.

The main problem with the experiments of the second UCT version still remains the huge fluctuation of the results, making it hard to draw conclusions. Note that also the confidence interval has nearly doubled its size compared to the experiments conducted for the first UCT version. Possibly, increasing the number of games could help, but it is quite obvious already after those few experiments that the second UCT version cannot compete with the first one, no matter what "good" configuration can be found. More on this topic will be discussed in Section 6.2.

6.1.2 Second announcement style version

This subsection uses the exact same procedure of experiments as the preceding one, the only difference is that the second announcement style version is used, i.e. players can only say "yes" or "no" when being asked if they want to an announcement or not, automatically announcing the next level of announcement when answering "yes".

	Player 1 Options			
SP factor:	500	500	500	500
Expl const.:	500	1000	2000	4000
	95% confidence intervals for average round scores			
P1 (UCT):	9.05 ± 0.79	8.95 ± 0.81	9.33 ± 0.76	9.72 ± 0.83
P2 (Rnd):	-3.16 ± 1.11	-2.90 ± 1.12	-3.19 ± 1.07	-3.48 ± 1.13
P3 (Rnd):	-3.76 ± 1.03	-3.95 ± 0.98	-3.91 ± 0.99	-3.86 ± 1.05
P4 (Rnd):	-2.14 ± 1.04	-2.10 ± 1.11	-2.22 ± 1.10	-2.38 ± 1.10

Table 6.15: One UCT version 1 player with three random players (1)

	Player 1 Options			
SP factor:	500	500	500	500
Expl const.:	7000	10000	15000	20000
	95% confidence intervals for average round scores			
P1 (UCT):	9.87 ± 0.84	9.56 ± 0.82	8.33 ± 0.84	7.89 ± 0.81
P2 (Rnd):	-3.64 ± 1.07	-3.28 ± 1.02	-2.70 ± 0.98	-2.78 ± 0.99
P3 (Rnd):	-4.00 ± 0.97	-4.00 ± 0.95	-3.56 ± 0.88	-3.28 ± 0.91
P4 (Rnd):	-2.23 ± 1.05	-2.28 ± 1.04	-2.07 ± 1.00	-1.83 ± 1.00

Table 6.16: One UCT version 1 player with three random players (2)

First UCT version

The first experiments again test different combinations for the score points constant and the exploration constant, fixing a value of 500 for the first one, using a setup where one UCT player (using the first UCT version) plays together with three random players.

Tables 6.15 and 6.16 show the results. The best configuration found is the one with values of 500/7000 for score points constant/exploration constant. Analog to the previous subsection where the first announcement style version was used, one can observe that increasing the relation of the exploration constant at the score points constant also increases the quality of the UCT player up to a certain amount.

Tables 6.17 and 6.18 show the result of the next series of experiments, where the configuration found above is tested against itself and against slightly differing configurations. The UCT players are placed behind each other, switching the positions in the experiments shown in the second table. Comparing the slightly differing configurations (first, second, fourth and fifth column) to the results obtained when the tested configuration 500/7000 plays against itself (middle column), one observes that all differing configurations achieve slightly less score points, with the exception of the configuration 600/7000, which slightly improves over the 500/7000 configuration. Note that the second placed UCT player apparently has a disadvantage due to the card deals, as already the middle column, where two identical UCT players are tested, shows an average round score points difference of 1.14. However, the second table, which shows the result for the same setup with inversed player positions, confirms the results of the first table, i.e. all configurations other than 500/7000 achieve less score points compared to the reference

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	7000	7000	7000	7000	7000
	Player 2 Options				
SP factor:	400	500	500	500	600
Expl const.:	7000	6000	7000	8000	7000
	95% confidence intervals for average round scores				
P1 (UCT):	7.11 ± 1.24	7.12 ± 1.29	7.29 ± 1.27	6.90 ± 1.29	6.98 ± 1.26
P2 (UCT):	5.70 ± 1.32	5.98 ± 1.32	6.15 ± 1.29	5.97 ± 1.29	6.23 ± 1.30
P3 (Rnd):	-7.10 ± 1.08	-6.86 ± 1.18	-7.62 ± 1.13	-7.23 ± 1.17	-7.03 ± 1.14
P4 (Rnd):	-5.72 ± 1.16	-6.24 ± 1.30	-5.82 ± 1.25	-5.64 ± 1.26	-6.18 ± 1.24

Table 6.17: Two UCT version 1 players positioned next to each other

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	7000	6000	7000	8000	7000
	Player 2 Options				
SP factor:	500	500	500	500	500
Expl const.:	7000	7000	7000	7000	7000
	95% confidence intervals for average round scores				
P1 (UCT):	6.84 ± 1.24	7.01 ± 1.22	7.29 ± 1.27	6.59 ± 1.24	7.53 ± 1.31
P2 (UCT):	6.32 ± 1.27	6.17 ± 1.32	6.15 ± 1.29	6.44 ± 1.26	6.14 ± 1.29
P3 (Rnd):	-7.92 ± 1.07	-7.34 ± 1.15	-7.62 ± 1.13	-7.22 ± 1.11	-7.93 ± 1.11
P4 (Rnd):	-5.24 ± 1.23	-5.83 ± 1.29	-5.82 ± 1.25	-5.81 ± 1.23	-5.74 ± 1.29

Table 6.18: Two UCT version 1 players positioned next to each other with inversed positions

middle column, while at the same time, the configuration 500/7000 improves. This holds again with the exception of the last column, where the 600/7000 configuration seems to have a little advantage over the 500/7000 version. It is worth to note that the differences between the results when reversing the player positions can nearly be neglected in comparison to the results of the previous subsection, where the first announcement style version was used. This makes the results more reliable and easier to interpret. To make sure that there is no exploitation involved when UCT players are placed next to each other, the experiments are repeated with a vis-à-vis positioning.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	7000	7000	7000	7000	7000
	Player 3 Options				
SP factor:	400	500	500	500	600
Expl const.:	7000	6000	7000	8000	7000
	95% confidence intervals for average round scores				
P1 (UCT):	6.66 ± 1.29	6.95 ± 1.38	6.55 ± 1.34	6.74 ± 1.28	6.45 ± 1.32
P2 (Rnd):	-6.73 ± 1.31	-6.56 ± 1.32	-7.31 ± 1.25	-6.91 ± 1.28	-6.30 ± 1.40
P3 (UCT):	5.61 ± 1.24	5.62 ± 1.23	6.16 ± 1.36	5.92 ± 1.24	5.88 ± 1.24
P4 (Rnd):	-5.54 ± 1.22	-6.00 ± 1.27	-5.40 ± 1.22	-5.75 ± 1.21	-6.02 ± 1.33

Table 6.19: Two UCT version 1 players positioned vis-à-vis

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	7000	6000	7000	8000	7000
	Player 3 Options				
SP factor:	500	500	500	500	500
Expl const.:	7000	7000	7000	7000	7000
	95% confidence intervals for average round scores				
P1 (UCT):	6.13 ± 1.27	6.29 ± 1.22	6.55 ± 1.34	6.85 ± 1.29	6.13 ± 1.30
P2 (Rnd):	-6.80 ± 1.30	-6.92 ± 1.32	-7.31 ± 1.25	-6.67 ± 1.35	-6.78 ± 1.30
P3 (UCT):	6.33 ± 1.25	6.56 ± 1.25	6.16 ± 1.36	6.30 ± 1.26	6.14 ± 1.42
P4 (Rnd):	-5.67 ± 1.33	-5.92 ± 1.32	-5.40 ± 1.22	-6.48 ± 1.31	-5.49 ± 1.35

Table 6.20: Two UCT version 1 players positioned vis-à-vis with inversed positions

Tables 6.19 and 6.20 show the results of these experiments. Taking a look at the first table, all configurations other than the one to test, namely 500/7000, achieve less average round score points, both compared directly against the configuration 500/7000 and to the reference result of the middle column, where two identical versions of the configuration 500/7000 played against each other. Also the second table, which shows the results of experiments with the same setup but inversed positions for the UCT players, confirms these results: the configuration 500/7000 is still the prevailing one,

with the exception of the fourth column, where the result of configuration 500/8000 is better than the reference value of the middle column, but still when adding up the results of both tables, 500/7000 has a slightly higher average round score. All in all, choosing 500 for the score points constant in combination with 7000 for the exploration constants is apparently a very reasonable choice for a baseline UCT player using the first UCT version.

Note that the last two experiments also show that using a positioning where UCT players are placed next to each other or vis-à-vis apparently does not influence the results when using the second announcement style version. This is certainly due to the fact that a UCT player cannot exploit another UCT player placed in front of him, which again is due to the fact that UCT players do not tend to over-announce when the second announcement style version is used. The reason for this is probably the reduced branching factor of nodes where a player has to decide if he wants to make an announcement or not. This observation still needs to be confirmed when using the second UCT version, but already now, it is worth to note that future experiments can use an arbitrary positioning of UCT players without risking to falsify the results.

Second UCT version

In the following, the same setup of experiments as before will be presented, but this time with UCT players using the second UCT version. Again, the first series of experiments consists of finding the best combination of values for the score points constant and the exploration constant by letting a UCT player player with three random players.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	500	1000	2000	4000	7000
	95% confidence intervals for average round scores				
P1 (UCT):	3.82 ± 1.17	6.64 ± 0.99	7.89 ± 0.82	9.62 ± 0.84	10.24 ± 1.00
P2 (Rnd):	-1.44 ± 1.30	-3.03 ± 1.24	-2.60 ± 1.02	-3.56 ± 0.97	-3.71 ± 0.96
P3 (Rnd):	-1.79 ± 1.35	-2.48 ± 1.16	-3.60 ± 0.96	-3.72 ± 0.90	-4.24 ± 0.87
P4 (Rnd):	-0.58 ± 1.22	-1.13 ± 1.24	-1.68 ± 1.03	-2.35 ± 1.00	-2.28 ± 0.94

Table 6.21: One UCT version 2 player with three random players (1)

Tables 6.21 and 6.22 show the results of these experiments. The combination 500/30000 is the best one, although only very slightly ahead of configuration 500/20000. Note that, as already observed in previous experiments, it still holds that increasing the exploration constant up to a certain amount while keeping the score points constant fixed benefits the obtained results.

Tables 6.23 and 6.24 show the results of experiments where the configuration 500/30000 is tested in direct play against itself and against slightly deviating configurations. UCT players are positioned next to each other and their positions are interchanged in the experiments depicted in the second table. The middle column, again the "reference" column because two identical UCT players with the configuration to be tested play against

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	10000	15000	20000	30000	40000
	95% confidence intervals for average round scores				
P1 (UCT):	11.51 ± 1.11	11.26 ± 1.14	11.98 ± 1.22	12.01 ± 1.23	10.57 ± 1.24
P2 (Rnd):	-3.92 ± 0.95	-3.64 ± 0.88	-4.19 ± 0.87	-3.90 ± 0.86	-3.65 ± 0.88
P3 (Rnd):	-4.55 ± 0.87	-4.55 ± 0.84	-4.60 ± 0.87	-4.72 ± 0.86	-4.10 ± 0.89
P4 (Rnd):	-3.04 ± 0.94	-3.07 ± 0.90	-3.19 ± 0.90	-3.39 ± 0.94	-2.82 ± 0.88

Table 6.22: One UCT version 2 player with three random players (2)

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	30000	30000	30000	30000	30000
	Player 2 Options				
SP factor:	400	500	500	500	600
Expl const.:	30000	24000	30000	36000	30000
	95% confidence intervals for average round scores				
P1 (UCT):	8.07 ± 1.76	8.31 ± 1.71	7.28 ± 1.77	8.38 ± 1.84	6.86 ± 1.80
P2 (UCT):	6.17 ± 1.59	6.04 ± 1.68	6.25 ± 1.74	6.11 ± 1.70	6.05 ± 1.77
P3 (Rnd):	-7.43 ± 0.96	-7.58 ± 0.92	-7.28 ± 0.92	-7.57 ± 0.96	-6.67 ± 0.97
P4 (Rnd):	-6.81 ± 1.00	-6.77 ± 1.06	-6.26 ± 1.02	-6.92 ± 1.00	-6.24 ± 1.01

Table 6.23: Two UCT version 2 players positioned next to each other

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	30000	24000	30000	36000	30000
	Player 2 Options				
SP factor:	500	500	500	500	500
Expl const.:	30000	30000	30000	30000	30000
	95% confidence intervals for average round scores				
P1 (UCT):	6.12 ± 1.84	8.12 ± 1.82	7.28 ± 1.77	7.62 ± 1.72	7.50 ± 1.90
P2 (UCT):	7.07 ± 1.71	5.11 ± 1.71	6.25 ± 1.74	6.15 ± 1.65	5.98 ± 1.73
P3 (Rnd):	-7.18 ± 0.94	-7.11 ± 0.91	-7.28 ± 0.92	-7.33 ± 0.93	-7.00 ± 0.88
P4 (Rnd):	-6.02 ± 0.98	-6.12 ± 0.99	-6.26 ± 1.02	-6.44 ± 0.96	-6.48 ± 1.01

Table 6.24: Two UCT version 2 players positioned next to each other with inversed positions

each other, shows a not so little difference between the results of the UCT players and also the confidence intervals are larger than the ones when using the first UCT version. This however corresponds to the previous observations of UCT players using the second UCT version. Still, the first table shows that 500/30000 achieves a better score than the differing configurations listed in columns one, two and four, but 600/3000 seems to be a bit favored. This observation is confirmed when taking a look at the second table, where player positions have been switched. Not only 600/30000, but also 500/24000 (which is a similar deviation from 500/30000 in terms of the relation between exploration constant and score points constant) achieve better results than 500/30000. The results of the fourth column are less obvious, as in the first table, 500/30000 clearly wins against 500/36000, but in the second table, it is the other way around. However, adding up the average score round points for both experiments, configuration 500/30000 still achieves a slightly better result than 500/36000.

	Player 1 Options				
SP factor:	500	500	500	500	500
Expl const.:	30000	30000	30000	30000	30000
	Player 3 Options				
SP factor:	400	500	500	500	600
Expl const.:	30000	24000	30000	36000	30000
	95% confidence intervals for average round scores				
P1 (UCT):	7.92 ± 1.68	6.37 ± 1.76	7.82 ± 1.76	7.98 ± 1.65	6.91 ± 1.70
P2 (Rnd):	-6.82 ± 0.99	-6.85 ± 1.00	-7.30 ± 1.01	-6.96 ± 0.99	-7.48 ± 0.95
P3 (UCT):	5.62 ± 1.72	6.76 ± 1.72	6.24 ± 1.79	5.94 ± 1.69	7.00 ± 1.71
P4 (Rnd):	-6.71 ± 0.99	-6.28 ± 1.05	-6.76 ± 1.03	-6.96 ± 0.98	-6.44 ± 0.98

Table 6.25: Two UCT version 2 players positioned vis-à-vis

	Player 1 Options				
SP factor:	400	500	500	500	600
Expl const.:	30000	24000	30000	36000	30000
	Player 3 Options				
SP factor:	500	500	500	500	500
Expl const.:	30000	30000	30000	30000	30000
	95% confidence intervals for average round scores				
P1 (UCT):	6.12 ± 1.73	7.12 ± 1.65	7.82 ± 1.76	5.90 ± 1.71	6.50 ± 1.69
P2 (Rnd):	-6.44 ± 0.98	-6.60 ± 0.97	-7.30 ± 1.01	-6.43 ± 1.02	-7.25 ± 1.04
P3 (UCT):	6.70 ± 1.84	6.33 ± 1.75	6.24 ± 1.79	6.34 ± 1.68	7.16 ± 1.81
P4 (Rnd):	-6.38 ± 1.07	-6.85 ± 1.00	-6.76 ± 1.03	-5.81 ± 1.02	-6.41 ± 0.99

Table 6.26: Two UCT version 2 players positioned vis-à-vis with inversed positions

To ensure that placing UCT players next to each other does not allow any kind of exploitation also when using the second UCT version, the same experiments as the ones

shown in the previous two tables are repeated with UCT players being placed vis-à-vis. Tables 6.25 and 6.26 show the results, where the experiments displayed in the second table use the same setup but with inversed positions for the UCT players. The first table perfectly supports the observations made in the previous two experiments, namely that 500/30000 is surpassed by 600/30000 and 500/24000. The second table does not confirm these results that well, as all configurations different from 500/30000 achieve worse results than 500/30000, but still 500/24000 and 600/30000 achieve the best out of these worse results. All together, I come to the conclusion that 30000 may be a bit too large and that lowering the exploration constant a bit would be appropriate. Taking a look at Table 6.22 again, one can see that 500/20000 achieved nearly as good results as 500/30000, at least in the test against three random players. Also 30000 is a large value compared to the value of 7000 chosen for the UCT player using the first UCT version. Therefore 500/20000 should be a very good configuration for a baseline UCT player using the second UCT version.

6.1.3 Chosen baseline configuration

Taking into account the results of the previous subsection, the first decision to make is a simple one: a baseline UCT player, no matter which UCT version it uses, should use the second announcement style version or more specific, as this is not a player's option but a global option, all further experiments will use the second announcement style version. There are several reasons for this choice: first, the obtained results are more reliable and show less deviations when comparing very similar configurations. Second, player positioning does not seem to play role when using the second announcement style version, as results obtained using a setup where two UCT players are placed behind each other are consistent to results obtained where they are placed vis-à-vis. Thus UCT players placed in front of another UCT player do not get exploited. This is certainly not the case when using the first announcement style version. Last but not least and related to the previous reason, UCT players do not tend to over-announce as it is the case when using the first announcement style version. This has been tested by analyzing the game process of some small tournaments comparing the different announcement style versions.

The second decision to make is whether to use the first or the second UCT version, but as it seems to be interesting to test both versions because they have different characteristics as seen so far, I decided to use two UCT baseline players, one for each UCT version.

The last choice concerns the values to choose for the score points constant and the exploration constant. Combinations for these parameters have been extensively tested and the results have been shown in the previous two subsections of this section. The combination of choice for the first UCT version baseline player is 500 for the score points constant and 7000 for the explorations constant and for the second CUT version baseline player, it is 500 and 20000.

All remaining parameters will stay unchanged and use the values stated in the introduction of this chapter and this section.

	General Options	
# games:	1000	1000
Comp. solos?:	no	no
Random cards:	yes	yes
Random seed:	2012	2012
Ann. version:	2	2
	Player 1 Options	
Version:	1	2
SP factor:	500	500
Team points?:	no	no
PP Divisor:	1	1
Expl const.:	7000	20000
# rollouts:	100	1000
# sim:	10	-
Allow ann.:	yes	yes
Wrong formula?:	yes	yes
MC simulation?:	yes	yes
Action selection:	1	1
	95% confidence intervals for average round scores	
P1 (UCT):	9.87 ± 0.84	11.98 ± 1.22
P2 (Rnd):	-3.64 ± 1.07	-4.19 ± 0.87
P3 (Rnd):	-4.00 ± 0.97	-4.60 ± 0.87
P4 (Rnd):	-2.23 ± 1.05	-3.19 ± 0.90

Table 6.27: A complete description of the configurations of the baseline UCT players for both UCT versions (used to play with the second announcement style version)

6.2 Comparing the two UCT versions

This section compares the two different implementations of the UCT algorithm, i.e. the first version using simulations and rollouts, fixing a card assignment per simulation, and the second version only using rollouts, each with a different card assignment. In addition to the baseline configuration, some additional "good" configurations according to the experiments shown in the previous section are tested. As usual, UCT players are tested playing against each other with two random players and alone against three random players.

	Player 1 Options			
Version:	1	2	1	2
	Player 3 Options			
Version:	-	-	2	1
	95% confidence intervals for average round scores			
P1 (UCT):	9.87 ± 0.84	11.98 ± 1.22	5.57 ± 1.18	9.07 ± 1.76
P2 (Rnd):	-3.64 ± 1.07	-4.19 ± 0.87	-6.78 ± 1.12	-7.43 ± 1.20
P3 (Rnd/UCT):	-4.00 ± 0.97	-4.60 ± 0.87	8.06 ± 1.83	5.22 ± 1.24
P4 (Rnd):	-2.23 ± 1.05	-3.19 ± 0.90	-6.85 ± 1.21	-6.86 ± 1.14
	Time			
Total time:	39m18s	1h59m49s	2h42m27s	2h43m37s

Table 6.28: Comparing the two UCT versions using the second announcement style version

Table 6.28 shows the two baseline configurations competing in separate experiments against three random players and in experiments where they directly play against each other together with two random players. Amazingly, the second UCT version wins against the first one in all experiments – I explicitly say amazingly because all first local tests and experiments always have shown way worse results for the second UCT version than for the first version. There are probably two reasons for this: first, many tests have been conducted using the fist announcement style version and not the second one and second, no "best" parameters for the score points constant and the exploration constant have been found at this moment, but some default configuration 1000/1000 was used and thus an exploration constant way too low for the second UCT version. Therefore, another series of experiments to compare the two UCT versions using the first announcement style will be conducted. For these, a different configuration of values for the score points constant and the exploration constant will be used, namely 500/4000 for both UCT versions, as these have shown the best results in the previous section.

Table 6.29 shows the results of these experiments. As expected, the first version of the UCT algorithm vastly outperforms the second version, both in direct comparison when playing together with two random players and when comparing results of matches against three random players. I originally expected the second UCT version to be at least as good as the first version, because using a different card assignment for each rollout over only using a few different card assignments and letting them be fixed for an entire

	Player 1 Options			
Version:	1	2	1	2
	Player 3 Options			
Version:	-	-	2	1
	95% confidence intervals for average round scores			
P1 (UCT):	10.44 ± 1.05	8.03 ± 1.51	22.97 ± 2.30	-11.04 ± 2.66
P2 (Rnd):	-3.75 ± 1.28	-2.89 ± 1.36	-7.92 ± 2.15	-6.90 ± 2.24
P3 (Rnd/UCT):	-4.28 ± 1.15	-3.57 ± 1.34	-9.89 ± 2.70	23.11 ± 2.40
P4 (Rnd):	-2.41 ± 1.30	-1.56 ± 1.30	-5.16 ± 2.26	-5.18 ± 2.20
	Time			
Total time:	46m54s	2h27m49s	3h34m4s	3h33m40s

Table 6.29: Comparing the two UCT versions using the first announcement style version

series of rollouts seemed to be a big refinement of the information used. Especially the amount of information contained in the final tree of the second version should be higher than the amount of information distributed over different trees constructed by the first version. This has been confirmed when using the second announcement style version, see above, but using the first announcement style version obviously causes huge problems for the second UCT version. The reason must thus lie in the larger branching factor of announcement move nodes of the game tree. This can result in the single tree constructed by the second UCT version to become very broad and thus to contain a lot of "diffuse" information, but none of the regions of the tree will really get explored intensively during the search or if so, then only a few of the interesting regions can be explored. In contrast to this, the first UCT version builds several search trees, and thus even if a single tree does not contain "good" information (because a card assignment is fixed for the whole tree and thus probably only one part of the tree is heavily explored, namely the part which seems to be promising under the fixed card assignment), aggregating over several trees apparently makes up for it.

Interestingly, Bjarnason et al. [3] come to a very similar conclusion when comparing results of UCT and "HOP-UCT", as they call the aggregation of several search trees constructed by UCT (which thus corresponds to the first UCT version in this case), applied to Klondike solitaire. They state that "Comparing the performance of HOP-UCT and UCT trials suggests that sampling multiple UCT trees boosts performance and decreases computing time compared to UCT trees with an equivalent number of total trajectories. [...] Not only does the HOP-UCT approach slightly outperform the UCT method, it requires less than one third the time to do it." These observations correspond astonishingly well to the results observed above, especially for the time aspect (the second UCT version needs approximately 3.2 times more time than the first version). The first UCT version even vastly and not only slightly outperforms the second UCT version.

Still, all the comments above can be neglected by preferring the second announcement style version over the first one, as it clearly allows the UCT players to achieve better results (note that the absolute average round score points are also higher when comparing the tests of a UCT players playing with three random players). Also note that using the

second announcement style version is a bit faster compared to using the first version. All remaining experiments of this chapter will thus use the second announcement style version.

6.3 Testing single player options

This section is dedicated to show experiments which compare the baseline UCT player with another UCT player that changes exactly one parameter, i.e. that changes one characteristic of the UCT algorithm in comparison to the baseline UCT players. Each of the following subsections describes which parameter is changed and shows how this is reflected in the results. The general setup of experiments is as follows: first, the UCT players with the different values for the tested parameter play together with three random players and second, they play directly against each other, together with two random players. In the latter setup, UCT players are positioned vis-à-vis, even if in an earlier section of this chapter, it was shown that when using the second announcement style version, the positioning of the UCT players does not matter. Also, all experiments are conducted both with UCT players using the first and the second version in order to see if maybe some of the player options tested yield an improvement for only one of the UCT version or a more accentuated change at one of the versions.

6.3.1 Using team points as a bias for the UCT rewards

This subsection is dedicated to investigate the influence of choosing between using a player's points or his team points as a bias in addition to the score points when calculating the UCT rewards.

	Player 1 Options			
Team points?:	no	yes	no	yes
	Player 3 Options			
Team points?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	9.87 ± 0.84	9.28 ± 0.83	6.36 ± 1.39	5.74 ± 1.27
P2 (Rnd):	-3.64 ± 1.07	-2.83 ± 1.07	-6.64 ± 1.32	-6.77 ± 1.29
P3 (Rnd/UCT):	-4.00 ± 0.97	-4.02 ± 1.02	5.92 ± 1.24	7.00 ± 1.29
P4 (Rnd):	-2.23 ± 1.05	-2.42 ± 1.04	-5.64 ± 1.27	-5.96 ± 1.27

Table 6.30: Comparing the use of team points against the use of player's points, using the first UCT version

Table 6.30 shows both the results of one UCT player playing with three random players (column one and two) and two UCT players playing together with two random players (columns three and four). Amazingly, the UCT player using his team points instead of his own points to bias the won score points achieves less score points in all of the four

experiments. I would have expected that using the team points as a bias should not be worse than using the player's points, but apparently this is not the case.

	Player 1 Options			
Team points?:	no	yes	no	yes
	Player 3 Options			
Team points?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	11.98 ± 1.22	12.00 ± 1.12	6.97 ± 1.74	7.91 ± 1.74
P2 (Rnd):	-4.19 ± 0.87	-4.30 ± 0.90	-7.29 ± 0.98	-7.80 ± 1.04
P3 (Rnd/UCT):	-4.60 ± 0.87	-4.45 ± 0.81	6.90 ± 1.65	6.68 ± 1.78
P4 (Rnd):	-3.19 ± 0.90	-3.25 ± 0.88	-6.58 ± 1.02	-6.79 ± 1.03

Table 6.31: Comparing the use of team points against the use of player's points, using the second UCT version

Table 6.31 shows the same setup of experiments, but UCT players use the second UCT version. Again somewhat amazing, now using the team points instead of the player's points achieves better results for the UCT players: in column three, the UCT player using his playing points and not his team points is slightly better than the UCT player doing the other way round, but in the fourth column, the UCT player using the team points wins by a large average round score points difference, and also when playing against three random players, the version using team points is a little bit better. To conclude, the first UCT version should use player's points to bias the score points when calculation the UCT rewards and the second UCT version should use the team points of the player instead.

6.3.2 Changing the announcement rule

This subsection investigates the effect of changing the announcement rules for UCT players, i.e. they can either be allowed to make all announcements, to only announce something if the calculated UCT reward for the corresponding successor is positive or to do no announcements at all. This option originally was intended to avoid over-announcing of the UCT players when using the first announcement style version, i.e. when they are allowed to choose from all legal announcements. As a consequence, announcement move nodes have a large branching factor and thus many times, the algorithm "wrongly" chooses one of the successors which corresponds to make an announcement although it is clear at the moment of the game that no announcement should be made at all. As over-announcing can be reduced a lot or even completely removed when using the second announcement style version, this option maybe does not yield any improvement.

Tables 6.32 and 6.33 show results of experiments where UCT players using the first UCT version play against three random players or against another UCT player with a different announcement rule, together with two random players. The first thing to notice in the first table, first three columns, is that there is no difference at all between allowing

	Player 1 Options				
Allow ann.:	no	yes	only +	yes	no
	Player 3 Options				
Allow ann.:	-	-	-	no	yes
	95% confidence intervals for average round scores				
P1 (UCT):	7.82 ± 0.67	9.87 ± 0.84	9.87 ± 0.84	6.53 ± 0.96	5.17 ± 0.92
P2 (Rnd):	-3.06 ± 0.93	-3.64 ± 1.07	-3.64 ± 1.07	-6.27 ± 0.96	-6.04 ± 1.01
P3 (Rnd/UCT):	-3.18 ± 0.85	-4.00 ± 0.97	-4.00 ± 0.97	4.65 ± 0.95	6.24 ± 0.95
P4 (Rnd):	-1.58 ± 0.93	-2.23 ± 1.05	-2.23 ± 1.05	-4.90 ± 1.00	-5.36 ± 0.99

Table 6.32: Comparing different announcement rules, using the first UCT version (1)

	Player 1 Options			
Allow ann.:	yes	only +	no	only +
	Player 3 Options			
Allow ann.:	only +	yes	only +	no
	95% confidence intervals for average round scores			
P1 (UCT):	6.07 ± 1.29	6.68 ± 1.31	5.14 ± 0.92	6.53 ± 0.96
P2 (Rnd):	-7.36 ± 1.23	-7.38 ± 1.22	-6.07 ± 1.01	-6.27 ± 0.96
P3 (UCT):	6.64 ± 1.30	6.01 ± 1.33	6.26 ± 0.95	4.65 ± 0.95
P4 (Rnd):	-5.35 ± 1.18	-5.30 ± 1.20	-5.34 ± 0.99	-4.90 ± 1.00

Table 6.33: Comparing different announcement rules, using the first UCT version (2)

all announcements or only announcements when the UCT reward is positive. This can be easily explained by the fact that the random players are not a "serious" opponent for the UCT player and thus he will always have positive UCT rewards for moves where he can choose to make an announcement or not, because when choosing not to make an announcement, the UCT player wins the game with a very high probability. The same does not hold anymore as soon as a second UCT player comes into play. Columns four and five of the first table and columns three and four of the second table clearly show that forbidding all announcements is a hard punishment for any UCT player. The comparison between allowing all announcements or only if they have a positive UCT reward can be found in the first two columns of the second table: the UCT player only making announcements if the UCT rewards for that successor are positive clearly achieves more points than the UCT player allowed to always make an announcement. This can be explained by the fact that UCT players free to announce even with negative UCT rewards still risk to make an over-announcement from time to time, which costs them a few score points. This cannot happen to the other UCT player who plays very defensively with respect to announcements.

Tables 6.34 and 6.35 show the results of the second series of experiments, which are basically the same than the ones above, but using the second UCT version. Also the results are quite similar: when playing against random players, it is clearly better to allow all announcements and even very slightly better to only allow making announcements if the

	Player 1 Options				
Allow ann.:	no	yes	only +	yes	no
	Player 3 Options				
Allow ann.:	-	-	-	no	yes
	95% confidence intervals for average round scores				
P1 (UCT):	9.20 ± 0.76	11.98 ± 1.22	12.00 ± 1.19	6.44 ± 1.19	3.91 ± 0.91
P2 (Rnd):	-3.25 ± 0.81	-4.19 ± 0.87	-4.18 ± 0.87	-5.61 ± 0.83	-5.49 ± 0.84
P3 (Rnd/UCT):	-3.63 ± 0.80	-4.60 ± 0.87	-4.59 ± 0.86	4.08 ± 0.87	6.37 ± 1.15
P4 (Rnd):	-2.32 ± 0.83	-3.19 ± 0.90	-3.24 ± 0.89	-4.92 ± 0.84	-4.79 ± 0.84

Table 6.34: Comparing different announcement rules, using the second UCT version (1)

	Player 1 Options			
Allow ann.:	yes	only +	no	only +
	Player 3 Options			
Allow ann.:	only +	yes	only +	no
	95% confidence intervals for average round scores			
P1 (UCT):	6.62 ± 1.74	6.80 ± 1.74	3.88 ± 0.91	6.47 ± 1.20
P2 (Rnd):	-7.50 ± 1.00	-7.55 ± 1.01	-5.52 ± 0.84	-5.62 ± 0.83
P3 (UCT):	7.73 ± 1.70	7.65 ± 1.73	6.48 ± 1.13	4.08 ± 0.88
P4 (Rnd):	-6.84 ± 1.03	-6.90 ± 1.04	-4.83 ± 0.84	-4.92 ± 0.84

Table 6.35: Comparing different announcement rules, using the second UCT version (2)

corresponding UCT rewards are positive. When taking a look at the experiments where two UCT players are involved, it is again obvious than forbidding all announcements is a very bad idea, as this lowers the average round score points by a large amount. More interestingly, the choice between allowing all announcements or only if rewards are positive is less clear than in the experiments where the first UCT version was used: when adding up the average score round points of the two experiments shown in the first and second column of the second table, the UCT player allowed to always make an announcement yields a score of 14.27 against a score of 14.53 of the UCT player who only can make an announcement if the expected reward is positive. Concluding this section, it is hard to choose between allowing free announcing for UCT players or restricting it to cases where the corresponding expected UCT reward is positive. This is also a question of playing style: either the UCT player plays more offensively (which can be good against week opponents) or more defensively (probably better against stronger opponents).

6.3.3 Using the correct UCT formula

This subsection examines the effect of using the correct UCT formula rather than the wrong one. The only reason using the wrong UCT formula is an option for a UCT player is that some local experiments have not shown a significant difference in the results after

fixing the original bug of the wrong UCT player. Also the difference of how the UCT formula is calculated is not very large, the effect of the nominator of the fraction under the square root being the number of total rollouts rather than the number of visits at the current node can only become large if the number of visits is low and the number of the current rollout is high. As a consequence, using the wrong UCT formula causes the exploration term to be a bit larger than it would normally be, but only when the number of the current rollout reaches a certain high value. This means that the UCT algorithm is likely to explore a bit more towards the end of iterations, when the number of rollouts grows large.

	Player 1 Options			
Wrong formula?:	no	yes	no	yes
	Player 3 Options			
Wrong formula?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	10.64 ± 0.90	9.87 ± 0.84	6.34 ± 1.27	6.36 ± 1.35
P2 (Rnd):	-3.22 ± 1.09	-3.64 ± 1.07	-6.42 ± 1.30	-7.46 ± 1.29
P3 (Rnd/UCT):	-4.72 ± 0.98	-4.00 ± 0.97	6.36 ± 1.26	6.53 ± 1.29
P4 (Rnd):	-2.70 ± 1.08	-2.23 ± 1.05	-6.29 ± 1.30	-5.44 ± 1.33

Table 6.36: Comparing the use of the correct UCT formula against the wrong one, using the first UCT version

Table 6.36 shows the results of experiments using the first UCT version. The UCT player using the correct UCT formula achieves a slightly better result when comparing the first two columns. The same holds for the direct comparison shown in the last two columns (when adding up the values of both experiments), although the improvement is less dominant.

	Player 1 Options			
Wrong formula?:	no	yes	no	yes
	Player 3 Options			
Wrong formula?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	12.82 ± 1.31	11.98 ± 1.22	8.66 ± 1.67	6.44 ± 1.78
P2 (Rnd):	-4.70 ± 0.93	-4.19 ± 0.87	-7.52 ± 0.99	-7.53 ± 1.03
P3 (UCT):	-4.93 ± 0.88	-4.60 ± 0.87	5.86 ± 1.75	7.85 ± 1.71
P4 (Rnd):	-3.19 ± 0.92	-3.19 ± 0.90	-7.00 ± 1.00	-6.76 ± 0.95

Table 6.37: Comparing the use of the correct UCT formula against the wrong one, using the second UCT version

Table 6.37 shows the results of the same experiments but using the second UCT version for the UCT players. It is obvious that the UCT player using the correct UCT formula achieves better results than the player using the wrong formula in all experiments, thus

strengthening the observations from the experiments with the fist UCT version. Using the correct UCT formula is thus a reasonable option to be tested further in Section 6.4.

6.3.4 Not using an MC simulation

In this subsection, the "use or not use an MC simulation" option is tested: if the UCT player uses an MC simulation as soon as a leaf node has been reached and a new node has been added to the search tree, then the game is only simulated from that moment on and no more nodes are added. If no MC simulation is used, then the algorithm continues to normally add one unvisited successor even after reaching a leaf node of the tree.

	Player 1 Options			
MC simulation?:	no	yes	no	yes
	Player 3 Options			
MC simulation?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	9.34 ± 0.82	9.87 ± 0.84	6.38 ± 1.25	6.38 ± 1.15
P2 (Rnd):	-3.43 ± 0.95	-3.64 ± 1.07	-7.23 ± 1.15	-7.08 ± 1.24
P3 (Rnd/UCT):	-3.47 ± 0.87	-4.00 ± 0.97	6.36 ± 1.22	6.12 ± 1.25
P4 (Rnd):	-2.44 ± 0.95	-2.23 ± 1.05	-5.51 ± 1.16	-5.42 ± 1.15

Table 6.38: Comparing the two different options concerning an MC simulation, using the first UCT version

Table 6.38 shows the results of the experiments where a UCT player plays with three random players (columns one and two) and the results of the experiments in which two UCT players are directly compared against each other, playing with two random players (columns three and four). Taking a look at the first two columns, it seems that using an MC simulation has a little advantage over adding all nodes encountered during search to the tree. The last two columns also show that the UCT player using an MC simulation achieves slightly better results (only when adding up the average round score points). Anyway, the results lie very close to each other. Thus there must be both advantages and disadvantages of adding many nodes to a tree or not. An advantage of adding all nodes encountered during search to the tree is the hope that in one of the future rollouts, at least a part of the path added to the tree might be re-visited again. If so, the information about the already visited nodes does not get lost but remains in the tree. On the other hand, the probability of reaching an exact same leaf node deep down in the tree again during a different rollout so that the search could profit from the additionally added nodes is very low. An interesting open question though is how the MC simulation option correlates with the action selection option, i.e. how successors are chosen as soon as a leaf node has been encountered. Combining both player options will be tested in Section 6.4.

Table 6.39 show the results of the same experiments but with UCT players using the second UCT version. In this case, not doing an MC simulation but adding all nodes

	Player 1 Options			
MC simulation?:	no	yes	no	yes
	Player 3 Options			
MC simulation?:	-	-	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	15.54 ± 1.50	11.98 ± 1.22	9.98 ± 2.18	6.21 ± 1.78
P2 (Rnd):	-5.40 ± 0.85	-4.19 ± 0.87	-7.84 ± 1.01	-6.71 ± 1.06
P3 (Rnd/UCT):	-5.45 ± 0.82	-4.60 ± 0.87	4.76 ± 1.59	6.64 ± 2.40
P4 (Rnd):	-4.69 ± 0.85	-3.19 ± 0.90	-6.90 ± 0.99	-6.15 ± 1.13

Table 6.39: Comparing the two different options concerning an MC simulation, using the second UCT version

encountered during search to the tree strongly outperforms a UCT player doing an MC simulation. A reason why when using the second UCT version, the advantages of adding more nodes to the tree prevail, could be the fact that only one big tree is constructed in contrast to the many smaller trees constructed by the first UCT version. Thus the probability of ending up in the same part of the search tree during some of the future rollouts is of course larger. Also for the second UCT version, it is of interest to test how the "use an MC simulation" option works together with the action selection option.

6.3.5 Changing the action selection version

The last subsection investigates the effect of using a heuristic to choose a successor of a node where the algorithm needs to select one of several unvisited successors. The algorithm differentiates two moments where a successor needs to be chosen: first, when reaching a leaf node of the tree which has at least two unvisited successors. One of them needs to be chosen and a new node will be added to the tree. From then on, either an MC simulation is carried out or the search continues normally, adding all nodes. In both cases, all game states reached after inserting a new node are of course nodes that cannot have been visited before already, and thus all successors are unvisited. To choose successor during the simulation/the continued search is the second place where a heuristic can come into play. As described in the previous chapter on the implementation, there are five versions for the action selection strategy that can be chosen via player options: the first one always chooses the first successor at the first decision moment and a random successor at all further decision moments. The second version always chooses a random successor. Starting with version three, at all decision moments during the MC simulation or the search after a first new node was added, a successor is chosen according to a heuristic (see Chapter 5 for details). The difference of versions three to five consists in the way the first unknown successor is chosen, just before the corresponding new node is added to the tree: version three selects the first one, version four chooses a random one and the fifth version also chooses this successor according to the heuristic.

The first series of experiments consist of UCT players playing against random players using different action selection versions. The results are shown in 6.40. The second

	Player 1 Options				
Action selection:	1	2	3	4	5
	95% confidence intervals for average round scores				
P1 (UCT):	9.87 ± 0.84	9.74 ± 0.87	10.47 ± 0.95	10.30 ± 0.91	10.34 ± 0.92
P2 (Rnd):	-3.64 ± 1.07	-3.10 ± 1.07	-3.63 ± 1.23	-3.66 ± 1.21	-3.94 ± 1.21
P3 (Rnd):	-4.00 ± 0.97	-4.17 ± 1.00	-4.63 ± 1.05	-4.50 ± 1.07	-4.38 ± 1.03
P4 (Rnd):	-2.23 ± 1.05	-2.46 ± 1.04	-2.21 ± 1.12	-2.13 ± 1.10	-2.02 ± 1.15

Table 6.40: One UCT player using the first UCT version playing with three random players

action selection version seems to be worse than the reference first one, but version three to five show some better results.

	1	2	3	4	5
1	-	6.21 ± 1.29, 6.17 ± 1.24	6.99 ± 1.31, 6.03 ± 1.34	6.89 ± 1.44, 6.60 ± 1.34	6.72 ± 1.33, 6.37 ± 1.34
2	6.44 ± 1.36, 5.69 ± 1.35	-	6.78 ± 1.43, 5.43 ± 1.35	6.53 ± 1.39, 5.97 ± 1.30	6.92 ± 1.37, 5.53 ± 1.35
3	7.09 ± 1.40, 6.57 ± 1.43	7.77 ± 1.40, 5.63 ± 1.31	-	7.06 ± 1.46, 6.56 ± 1.33	7.30 ± 1.43, 6.84 ± 1.34
4	6.36 ± 1.38, 6.94 ± 1.35	7.09 ± 1.40, 6.46 ± 1.34	7.48 ± 1.37, 6.39 ± 1.38	-	7.72 ± 1.40, 6.11 ± 1.36
5	6.96 ± 1.47, 6.46 ± 1.42	6.82 ± 1.42, 6.64 ± 1.32	7.81 ± 1.51, 5.82 ± 1.39	7.15 ± 1.45, 6.26 ± 1.45	-

Table 6.41: A matrix showing the results of experiments where two UCT players using the first UCT version play together with two random players, varying the action selection version (displayed in the first row and the first column). The first entry corresponds to the 95% confidence interval for the average round score points of the row player, the second entry for those of the column player.

Table 6.41 shows the results of two UCT players using the first UCT version playing against each other, together with three random players. The results are given in the form of a matrix where the values shown in the first row and the first column correspond to the action selection version used. The first entry in the matrix corresponds to the 95% confidence interval for the average round score points of the row player, the second entry for those of the column player. The row player is always set on position one and the column player on position three. Thus all comparison between two specific versions have been tested twice, once with inversed positions. As the values are all quite close to each other and no clear tendency can be observed, I decided to add up the average round score points for each comparison of the different versions. The result is shown in Table 6.42, where again the first entry corresponds to the result of the row player and the second entry to the result of the column player, but this time the values include both

experiments of each pairing.

	1	2	3	4	5
1	-	11.9, 12.61	13.56, 13.12	13.83, 12.96	13.18, 13.33
2	-	-	12.41, 13.2	12.99, 13.06	13.56, 12.35
3	-	-	-	13.45, 14.04	13.12, 14.65
4	-	-	-	-	13.98, 13.26
5	-	-	-	-	-

Table 6.42: The same experiments as those shown in the previous matrix, but this time the average round score points have been added up for each version in order to allow a better comparison. The first entry is still the 95% confidence interval for the average round score points for the row player and the second entry the result achieved by the column player.

Unfortunately, also after adding up the values, no clear results are obtained: version one wins in two out of four cases, the same holds for version two, version three loses three out of the four pairings, version four wins in three out of the four cases and version five wins two out of four pairings. I expected that guiding the action selection with the help of a heuristic rather than choosing a fixed or a random successors should at least not make the UCT player worse, but at least for the first UCT version, no real improvement can be observed.

	Player 1 Options				
Action selection:	1	2	3	4	5
	95% confidence intervals for average round scores				
P1 (UCT):	11.98 ± 1.22	4.55 ± 1.30	11.37 ± 1.14	3.34 ± 1.11	1.48 ± 1.07
P2 (Rnd):	-4.19 ± 0.87	-1.99 ± 0.88	-3.91 ± 0.96	-1.00 ± 0.85	-0.59 ± 0.86
P3 (Rnd):	-4.60 ± 0.87	-1.75 ± 0.84	-4.51 ± 0.91	-1.45 ± 0.89	-0.71 ± 0.83
P4 (Rnd):	-3.19 ± 0.90	-0.81 ± 0.87	-2.94 ± 0.97	-0.89 ± 0.85	-0.18 ± 0.88

Table 6.43: One UCT player using the second UCT version playing with three random players

In the first series of experiments, again one UCT player, but this time using the second UCT version, plays with three random players, testing different action selection versions. The results are quite devastating, as all action selection versions other than the reference one achieve a worse result. The only version that to some degree achieves reasonable results is the third version.

Table 6.44 shows the results of UCT players using the second UCT version playing against each other. As expected after seeing the above results, the original version which does not use the heuristic at, all dominates all other versions. Interestingly, even the second version, where the only change is that also the first time during a rollout where the algorithm needs to choose one successors, it chooses a random one and not the first one, achieves a real bad result compared to the first version. A reason for this

	1	2	3	4	5
1	-	13.42 ± 1.69, -4.46 ± 1.66	8.77 ± 1.71, 3.98 ± 1.47	13.01 ± 1.75, -2.65 ± 1.39	14.75 ± 1.67, -5.48 ± 1.33
2	-3.45 ± 1.67, 12.43 ± 1.60	-	-3.20 ± 1.81, 9.18 ± 1.48	2.71 ± 1.67, 1.06 ± 1.26	3.30 ± 1.70, 0.34 ± 1.22
3	6.11 ± 1.65, 8.28 ± 1.87	10.58 ± 1.50, -1.99 ± 1.71	-	11.14 ± 1.53, -2.75 ± 1.42	12.25 ± 1.46, -4.18 ± 1.33
4	-3.23 ± 1.49, 13.38 ± 1.66	1.00 ± 1.42, 1.56 ± 1.65	-2.45 ± 1.55, 9.70 ± 1.36	-	1.93 ± 1.45, 0.60 ± 1.33
5	-5.02 ± 1.41, 13.60 ± 1.61	-0.26 ± 1.41, 2.94 ± 1.67	-4.83 ± 1.51, 10.68 ± 1.36	-0.72 ± 1.54, 2.23 ± 1.35	-

Table 6.44: This matrix shows the results of experiments corresponding to the first matrix of this subsection, but the UCT players use the second UCT version.

might be that when a node corresponding to an announcement move is reached, the algorithm usually (i.e. in the first version) adds the first node to the tree first, which corresponds to choosing "no announcement" or answering "no" to the question if an announcement should be done or not. Playing defensively, first checking the option of making no announcement seems to be quite reasonable. Still, the differences in the resulting score points should not be that high. The only version that is not getting strongly beaten by the first version is version three (which also wins against version two, four and five), where the algorithm keeps the same decision making compared to the first version when reaching a leaf node for the first time in the rollout (always choosing the first successor) but uses the heuristic rather than choosing a random successor when doing an MC simulation or continuing the search. This also supports the observation that choosing the first successor over choosing a random one is "better", of course only due to the ordering of possible successors (cards are always sorted from the lowest rank to the highest rank within a suit, and non-trump suits are ordered before the trump suit, if existent, and announcements are always ordered starting with the choice "no"/"no announcement" followed by the legal announcements in increasing order or "yes" in case of the second announcement style version). Concluding this subsection, using a heuristic to bias action selection when choosing between several unvisited successors certainly does not fulfill the expectations, but deserves to be investigated further.

6.4 Combining several player options

This section investigates the effect of using several UCT player enhancements found in the last section. Concretely, for the first UCT versions, the options tested in combinations include the usage of a different announcement rule, namely to allow announcing only if the corresponding UCT reward is positive, to use the correct UCT formula and to continue adding all nodes to the search tree rather than doing a pure MC simulation. The latter option did not show a real improvement for the first UCT version, but it

also did not make it worse. For the second UCT versions, there is one additional "good" option which should be used: the UCT player should use his team points rather than his own playing points to bias the score points when calculating UCT rewards out of the game result. Note that the usage of the heuristic for action selection is not investigated further with the exception of testing the different versions of action selection combined with adding all nodes to the tree rather than using an MC simulation. This combination seems to be interesting because depending on the chosen action selection, very different nodes will be added to the tree after reaching a leaf node.

6.4.1 Comparing action selection versions with no MC simulation

This subsection is dedicated to testing the effect of combining the use of different action selection methods together with not using an MC simulation but adding all nodes encountered during search to the tree.

	Player 1 Options				
MC simulation?:	no	no	no	no	no
Action selection:	1	2	3	4	5
	95% confidence intervals for average round scores				
P1 (UCT):	9.07 ± 0.82	9.23 ± 0.86	9.91 ± 0.89	9.90 ± 0.88	9.92 ± 0.85
P2 (Rnd):	-3.08 ± 0.98	-3.60 ± 0.95	-3.50 ± 1.11	-3.64 ± 1.07	-3.66 ± 1.05
P3 (Rnd):	-3.53 ± 0.89	-3.52 ± 0.87	-3.75 ± 0.98	-4.23 ± 0.99	-3.84 ± 0.97
P4 (Rnd):	-2.46 ± 0.97	-2.11 ± 0.95	-2.66 ± 1.05	-2.03 ± 1.04	-2.42 ± 1.07

Table 6.45: One UCT player using the first UCT version playing together with three random players

Table 6.45 shows the results of UCT players using the first UCT version playing against three random players. They are a small improvement compared to the experiments done using an MC simulation in the sense that the usage of the heuristic for action selection becomes a bit more meaningful.

Table 6.46 again shows a matrix like in the previous section, showing the 95% confidence intervals for average round score points of the two UCT players playing against each other with two random players. Again, the first entry corresponds to the result achieved by the row player, the second entry to the results of the column player. Values shown in the first row and column mark the used version of the action selection strategy. In order to better see what happens, the results of both experiments of each paring are again added up and shown in Table 6.47

The results show that the first (original) version of the action selection strategy loses against all other configurations except for the second version, where still no heuristic is used but where the first node to be added to the tree is chosen randomly out of the set of unvisited successors rather than choosing the first one, as version one and three do. The second version loses against all other versions. The third version seems to be the best one, it wins against all others. The fourth version is better than the fifth version. These

	1	2	3	4	5
1	-	5.85 ± 1.12, 5.65 ± 1.07	5.63 ± 1.25, 6.54 ± 1.12	6.04 ± 1.26, 5.94 ± 1.17	5.79 ± 1.26, 5.98 ± 1.07
2	5.67 ± 1.15, 5.70 ± 1.12	-	5.73 ± 1.25, 6.12 ± 1.09	5.87 ± 1.26, 6.31 ± 1.12	6.47 ± 1.24, 5.79 ± 1.10
3	6.78 ± 1.20, 5.47 ± 1.22	7.13 ± 1.18, 5.54 ± 1.23	-	6.93 ± 1.27, 5.80 ± 1.15	6.96 ± 1.38, 5.85 ± 1.16
4	6.34 ± 1.12, 5.76 ± 1.20	7.33 ± 1.14, 5.30 ± 1.26	6.84 ± 1.29, 6.32 ± 1.27	-	6.96 ± 1.25, 5.51 ± 1.16
5	6.83 ± 1.13, 5.24 ± 1.17	6.72 ± 1.12, 5.81 ± 1.23	6.57 ± 1.24, 6.10 ± 1.18	6.42 ± 1.24, 6.28 ± 1.20	-

Table 6.46: A matrix showing the results of experiments where two UCT players using the first UCT version play together with two random players, varying the action selection version (displayed in the first row and the first column). The first entry corresponds to the 95% confidence interval for the average round score points of the row player, the second entry for those of the column player.

	1	2	3	4	5
1	-	11.55, 11.32	11.1, 13.32	11.8, 12.28	11.03, 12.81
2	-	-	11.27, 13.25	11.17, 13.64	12.28, 12.51
3	-	-	-	13.25, 12.64	13.06, 12.24
4	-	-	-	-	13.24, 11.93
5	-	-	-	-	-

Table 6.47: The same experiments as those shown in the previous matrix, but this time the average round score points have been added up for each pairing of versions in order to allow a better comparison.

results are a bit surprising, as they state that using the heuristic to choose successors is only reasonable *after* having added the first node to the tree, i.e. when continuing the search until a terminal state is reached. Further, when expanding a node for the first time during a rollout, choosing the first successor is the best choice. Again, this must be due to the ordering of legal moves. Also using an MC simulation rather than adding all nodes to the tree (see previous section) did not show a great improvement when using the heuristic. This can be explained rudimentarily by the fact that during an MC simulation, the nodes are not added to the tree, thus the information gain of using a heuristic during the MC simulation is only reflected in the results of the terminal state reached, but the nodes are not ultimately added to the tree. Still, the information is of course reflected in the UCT rewards which are added to the tree, thus it is not very clear why there exists this difference in the results.

	Player 1 Options				
MC simulation?:	no	no	no	no	no
Action selection:	1	2	3	4	5
	95% confidence intervals for average round scores				
P1 (UCT):	14.66 ± 1.47	3.18 ± 1.64	12.31 ± 1.29	3.59 ± 1.13	0.61 ± 1.22
P2 (Rnd):	-5.01 ± 0.85	-1.27 ± 0.82	-4.42 ± 0.91	-1.50 ± 0.81	-0.34 ± 0.88
P3 (Rnd):	-5.53 ± 0.79	-1.40 ± 0.84	-4.74 ± 0.91	-1.84 ± 0.83	-0.66 ± 0.87
P4 (Rnd):	-4.12 ± 0.86	-0.51 ± 0.90	-3.16 ± 0.93	-0.25 ± 0.85	0.40 ± 0.84

Table 6.48: One UCT player using the first UCT version playing together with three random players

Table 6.48 shows the corresponding results of the second UCT version when playing against three random players. As they are still as devastating as in the previous section, where the action selection strategy was tested using an MC simulation, this options is not further investigated for the second UCT version.

6.4.2 Combining good options for the first UCT version

The following experiments for the first UCT version combine using different announcement rules, using the wrong or the correct UCT formula and using an MC simulation or not.

Table 6.49 shows the results of the first series of experiments, where all combinations of using the mentioned three options (excluding cases where none or only one of the enhancements are used as this has been tested in the previous sections) are tested against three random players. Comparing the results with the average round score points value of 9.87 the baseline player achieves, only the third column shows a significant improvement for the UCT players that uses all enhancements except to not use an MC simulation, i.e. the player does an MC simulation in each rollout. Observations made in the previous sections include that it is not very clear if using an MC simulation or not is better when using the first UCT version, but when combining it with other "found to be good"

	Player 1 Options			
Allow ann.:	yes	only +	only +	only +
Wrong formula?:	no	yes	no	no
MC simulation?:	no	no	yes	no
	95% confidence intervals for average round scores			
P1 (UCT):	9.34 ± 0.82	9.07 ± 0.82	10.64 ± 0.90	9.34 ± 0.82
P2 (Rnd):	-3.43 ± 0.95	-3.08 ± 0.98	-3.22 ± 1.09	-3.43 ± 0.95
P3 (Rnd):	-3.47 ± 0.87	-3.53 ± 0.89	-4.72 ± 0.98	-3.47 ± 0.87
P4 (Rnd):	-2.44 ± 0.95	-2.46 ± 0.97	-2.70 ± 1.08	-2.44 ± 0.95

Table 6.49: One UCT player testing the combinations of using two or all of the three selected different options together

options, it seems that using an MC simulation should be preferred over adding all nodes to the tree.

	Player 1 Options			
Allow ann.:	yes	only +	only +	only +
Wrong formula?:	no	no	yes	no
MC simulation?:	no	no	no	no
	Player 3 Options			
Allow ann.:	only +	yes	only +	only +
Wrong formula?:	no	no	no	yes
MC simulation?:	no	no	no	no
	95% confidence intervals for average round scores			
P1 (UCT):	5.80 ± 1.07	6.17 ± 1.07	5.53 ± 1.07	6.03 ± 1.10
P2 (Rnd):	-6.21 ± 1.00	-6.37 ± 1.00	-6.27 ± 0.95	-6.33 ± 1.07
P3 (UCT):	6.00 ± 1.14	5.62 ± 1.15	5.93 ± 1.10	5.68 ± 1.09
P4 (Rnd):	-5.59 ± 1.00	-5.42 ± 1.00	-5.18 ± 0.96	-5.37 ± 0.99

Table 6.50: One UCT player testing the combinations of using two of the three selected different options against another UCT player using all three of them (1)

Tables 6.50 and 6.51 show the results of experiments where one UCT player using all possible combinations of using two of the three selected options plays against another UCT player using all three options at the same time. The first table shows very well that using all three options at the same time wins against the version which allows all announcements and the version which uses the wrong UCT formula. The second table however shows that the configuration which does not add all nodes to the tree but instead uses an MC simulation wins against the version using all three options (and which thus adds all nodes to the tree instead of only one per rollout). This observation corresponds very well to the one seen in Table 6.49. Of course one could now continue testing the configuration using that uses an MC simulation, the correct UCT formula and only allows announcements if the corresponding UCT rewards are positive, but I expect this configuration to be the strongest one when considering the experiments shown in this

	Player 1 Options	
Allow ann.:	only +	only +
Wrong formula?:	no	no
MC simulation?:	yes	no
	Player 3 Options	
Allow ann.:	only +	only +
Wrong formula?:	no	no
MC simulation?:	no	yes
	95% confidence intervals for average round scores	
P1 (UCT):	6.90 ± 1.13	6.60 ± 1.26
P2 (Rnd):	-6.78 ± 1.22	-7.25 ± 1.18
P3 (UCT):	5.44 ± 1.20	6.53 ± 1.19
P4 (Rnd):	-5.56 ± 1.15	-5.88 ± 1.17

Table 6.51: One UCT player testing the combinations of using two of the three selected different options against another UCT player using all three of them (2)

subsection.

6.4.3 Combining good options for the second UCT version

The following experiments for the second UCT version investigates the effect of using different combinations of using team points or player's points when calculating UCT rewards, allowing all announcements or only if the corresponding UCT rewards are positive, using the wrong or the correct UCT formula and using an MC simulation or not.

	Player 1 Options				
Team points?:	no	yes	yes	yes	yes
Allow ann.:	only +	yes	only +	only +	only +
Wrong formula?:	no	no	yes	no	no
MC simulation?:	no	no	no	yes	no
	95% confidence intervals for average round scores				
P1 (UCT):	15.32 ± 1.48	15.20 ± 1.32	14.96 ± 1.40	12.76 ± 1.24	15.01 ± 1.31
P2 (Rnd):	-5.33 ± 0.84	-5.22 ± 0.76	-5.24 ± 0.83	-4.37 ± 0.88	-5.15 ± 0.77
P3 (Rnd):	-5.38 ± 0.81	-5.49 ± 0.76	-5.49 ± 0.82	-5.05 ± 0.87	-5.42 ± 0.75
P4 (Rnd):	-4.62 ± 0.84	-4.50 ± 0.83	-4.23 ± 0.86	-3.34 ± 0.89	-4.43 ± 0.83

Table 6.52: One UCT player testing the combinations of using three of the four and all of the four selected different options together

The first series of experiments consists of letting one UCT player play together with three random players, using all possible combinations of using or not using the four selected player options, with the exception of combinations where only one or none of them is used because those combinations have been tested in the previous section. Table

	Player 1 Options		
Team points?:	no	no	no
Allow ann.:	yes	only +	only +
Wrong formula?:	no	yes	no
MC simulation?:	no	no	yes
	95% confidence intervals for average round scores		
P1 (UCT):	15.54 ± 1.50	14.55 ± 1.40	12.74 ± 1.28
P2 (Rnd):	-5.40 ± 0.85	-4.97 ± 0.84	-4.67 ± 0.93
P3 (Rnd):	-5.45 ± 0.82	-5.49 ± 0.79	-4.90 ± 0.88
P4 (Rnd):	-4.69 ± 0.85	-4.08 ± 0.84	-3.17 ± 0.92

Table 6.53: One UCT player testing the combinations of using two of the four selected different options together (1)

	Player 1 Options		
Team points?:	yes	yes	yes
Allow ann.:	yes	yes	only +
Wrong formula?:	yes	no	yes
MC simulation?:	no	yes	yes
	95% confidence intervals for average round scores		
P1 (UCT):	15.07 ± 1.44	12.78 ± 1.26	11.88 ± 1.12
P2 (Rnd):	-5.28 ± 0.84	-4.38 ± 0.88	-4.26 ± 0.90
P3 (Rnd):	-5.52 ± 0.84	-5.06 ± 0.87	-4.41 ± 0.81
P4 (Rnd):	-4.27 ± 0.86	-3.35 ± 0.89	-3.21 ± 0.88

Table 6.54: One UCT player testing the combinations of using two of the four selected different options together (2)

6.52 shows the results of experiments where three or all of the four combinations have been tested. The only relatively clear tendency that can be observed is that using an MC simulation yields less good results compared to the other combination. Tables 6.53 and 6.54 show the results of experiments where combinations of using two of the four options are tested. It is again quite difficult to read a lot from these results, but at least using an MC simulation still yields comparably poor results. Also, using all four enhancements together seems to be reasonable, as the results are only slightly worse than when using the player's points instead of the team points (first column) or when allowing all announcements instead of only if they rewards are positive (second column). The following experiments with two UCT players will always compare a combination of using some of the options against using all of them, because this still seems to be the most promising configuration, as every of the options when used alone improved the UCT player, as seen in the previous section.

	Player 1 Options			
Team points?:	no	yes	yes	yes
Allow ann.:	only +	only +	yes	only +
Wrong formula?:	no	no	no	no
MC simulation?:	no	no	no	no
	Player 3 Options			
Team points?:	yes	no	yes	yes
Allow ann.:	only +	only +	only +	yes
Wrong formula?:	no	no	no	no
MC simulation?:	no	no	no	no
	95% confidence intervals for average round scores			
P1 (UCT):	8.00 ± 2.18	8.40 ± 2.09	6.69 ± 2.15	6.83 ± 2.13
P2 (Rnd):	-7.25 ± 1.08	-7.08 ± 1.14	-6.85 ± 1.05	-6.92 ± 1.06
P3 (UCT):	5.78 ± 2.23	5.31 ± 2.13	6.64 ± 2.01	6.65 ± 2.08
P4 (Rnd):	-6.52 ± 1.05	-6.63 ± 1.09	-6.48 ± 1.03	-6.56 ± 1.03

Table 6.55: One UCT player testing the combinations of using three of the four selected different options against another UCT player using all four of them (1)

Tables 6.55 and 6.56 show the results of experiments where a UCT player using three of the four selected options plays against a UCT player using all four options. In the first table, the results of the pairings are quite close, one needs to add up the average round score points of both experiments (with inversed positions). Then the version using all four enhancements slightly wins against the version where player's points are used instead of team points and the version where announcements are always allowed rather than only if the corresponding UCT rewards are positive. These results correspond very well to the ones seen in Table 6.52, where the same combinations have been tested against three random players and where also the same two versions were even slightly better than the version using all four options at the same time. The results of the second table are more obvious, i.e. using the correct UCT formula and not using an MC simulation is better than using the wrong UCT formula or using an MC simulation.

	Player 1 Options			
Team points?:	yes	yes	yes	yes
Allow ann.:	only +	only +	only +	only +
Wrong formula?:	yes	no	no	no
MC simulation?:	no	no	yes	no
	Player 3 Options			
Team points?:	yes	yes	yes	yes
Allow ann.:	only +	only +	only +	only +
Wrong formula?:	no	yes	no	no
MC simulation?:	no	no	no	yes
	95% confidence intervals for average round scores			
P1 (UCT):	7.46 ± 2.08	9.97 ± 2.14	5.91 ± 1.64	10.83 ± 2.08
P2 (Rnd):	-7.10 ± 1.06	-7.87 ± 1.04	-7.08 ± 1.04	-7.71 ± 0.96
P3 (UCT):	5.90 ± 2.15	5.40 ± 2.06	7.99 ± 2.13	4.37 ± 1.66
P4 (Rnd):	-6.26 ± 1.04	-7.51 ± 1.09	-6.82 ± 1.02	-7.49 ± 1.01

Table 6.56: One UCT player testing the combinations of using three of the four selected different options against another UCT player using all four of them (2)

Again, these results correspond well to the ones shown in Table 6.52.

Tables 6.57, 6.58 and 6.59 show the result of experiments where a UCT player using two out of the four chosen options against a UCT player using all of them. In all six pairings, the UCT player using all four enhancements at the same time achieves a higher average round score points value than the other UCT player using a combination of two out of the four options. The first three pairings, shown in the first table and the first two columns of the second table, show no significant difference in the results and one needs to add up the average round score points of both experiments of each pairing. For the remaining three pairings, the version using two of the four options uses an MC simulation, which again yields to significantly worse results than when not using an MC simulation.

Summarizing the results of this subsection, it has been shown that combining several of the previously found to be "reasonable" player options works well and fortunately, using all four of them at the same time achieves the overall best results.

6.4.4 Chosen best configurations for both UCT versions

The configuration of the best UCT players found for both UCT versions is shown in Table 6.60. Note that the obtained results when playing with three random players indeed surpass the results of the baseline configurations, depicted in Table6.27

	Player 1 Options			
Team points?:	no	yes	no	yes
Allow ann.:	yes	only +	only +	only +
Wrong formula?:	no	no	yes	no
MC simulation?:	no	no	no	no
	Player 3 Options			
Team points?:	yes	no	yes	no
Allow ann.:	only +	yes	only +	only +
Wrong formula?:	no	no	no	yes
MC simulation?:	no	no	no	no
	95% confidence intervals for average round scores			
P1 (UCT):	7.79 ± 2.27	8.70 ± 2.11	8.06 ± 1.99	8.36 ± 2.18
P2 (Rnd):	-7.20 ± 1.11	-6.98 ± 1.15	-7.38 ± 0.96	-7.42 ± 1.04
P3 (UCT):	5.88 ± 2.25	4.80 ± 2.17	5.66 ± 2.14	5.95 ± 2.00
P4 (Rnd):	-6.47 ± 1.06	-6.53 ± 1.11	-6.34 ± 1.00	-6.90 ± 1.03

Table 6.57: One UCT player testing the combinations of using two of the four selected different options against another UCT player using all four of them (1)

	Player 1 Options			
Team points?:	no	yes	yes	yes
Allow ann.:	only +	only +	yes	only +
Wrong formula?:	no	no	yes	no
MC simulation?:	yes	no	no	no
	Player 3 Options			
Team points?:	yes	no	yes	yes
Allow ann.:	only +	only +	only +	yes
Wrong formula?:	no	no	no	yes
MC simulation?:	no	yes	no	no
	95% confidence intervals for average round scores			
P1 (UCT):	6.35 ± 1.76	9.47 ± 2.01	7.46 ± 2.14	10.03 ± 2.14
P2 (Rnd):	-7.64 ± 1.00	-7.81 ± 0.88	-7.12 ± 1.06	-7.86 ± 1.06
P3 (UCT):	7.96 ± 2.05	5.57 ± 1.67	5.94 ± 2.17	5.32 ± 2.11
P4 (Rnd):	-6.66 ± 1.00	-7.23 ± 0.90	-6.28 ± 1.06	-7.50 ± 1.10

Table 6.58: One UCT player testing the combinations of using two of the four selected different options against another UCT player using all four of them (2)

	Player 1 Options			
Team points?:	yes	yes	yes	yes
Allow ann.:	yes	only +	only +	only +
Wrong formula?:	no	no	yes	no
MC simulation?:	yes	no	yes	no
	Player 3 Options			
Team points?:	yes	yes	yes	yes
Allow ann.:	only +	yes	only +	only +
Wrong formula?:	no	no	no	yes
MC simulation?:	no	yes	no	yes
	95% confidence intervals for average round scores			
P1 (UCT):	5.85 ± 1.66	10.94 ± 2.09	5.97 ± 1.62	10.44 ± 2.12
P2 (Rnd):	-7.08 ± 1.05	-7.64 ± 0.96	-7.34 ± 1.05	-7.91 ± 0.96
P3 (UCT):	8.06 ± 2.14	4.12 ± 1.66	7.94 ± 2.28	4.96 ± 1.62
P4 (Rnd):	-6.83 ± 1.03	-7.42 ± 1.02	-6.57 ± 1.01	-7.48 ± 0.96

Table 6.59: One UCT player testing the combinations of using two of the four selected different options against another UCT player using all four of them (3)

	General Options	
# games:	1000	1000
Comp. solos?:	no	no
Random cards:	yes	yes
Random seed:	2012	2012
Ann. version:	2	2
	Player 1 Options	
Version:	1	2
SP factor:	500	500
Team points?:	no	yes
PP Divisor:	1	1
Expl const.:	7000	20000
# rollouts:	100	1000
# sim:	10	-
Allow ann.:	only +	only +
Wrong formula?:	no	no
MC simulation?:	yes	no
Action selection:	1	1
	95% confidence intervals for average round scores	
P1 (UCT):	10.64 ± 0.90	15.01 ± 1.31
P2 (Rnd):	-3.22 ± 1.09	-5.15 ± 0.77
P3 (Rnd):	-4.72 ± 0.98	-5.42 ± 0.75
P4 (Rnd):	-2.70 ± 1.08	-4.43 ± 0.83

Table 6.60: Configurations of the best UCT players for each version

6.5 Increasing the number of rollouts and the number of games

This section is intended to investigate the effect of raising the number of rollouts (and simulations for the first UCT version) on the quality of the results and the effect of increasing the number of games on the confidence intervals of the results. Furthermore, the amount of time and memory used when increasing those parameters will be observed. For this purpose, the best configurations found in the previous section will be used. Also the two UCT versions are again tested against each other, to determine the final version to be used for an experiment against a human player.

	General Options			
# games:	1000	1000	1000	4000
	Player 1 Options			
# rollouts:	100	100	1000	100
# sim:	10	100	10	10
	95% confidence intervals for average round scores			
P1 (UCT):	10.64 ± 0.90	11.06 ± 0.87	12.78 ± 1.05	9.98 ± 0.45
P2 (Rnd):	-3.22 ± 1.09	-3.53 ± 1.11	-4.19 ± 1.02	-3.14 ± 0.53
P3 (Rnd):	-4.72 ± 0.98	-4.88 ± 1.00	-4.80 ± 0.96	-3.72 ± 0.51
P4 (Rnd):	-2.70 ± 1.08	-2.64 ± 1.11	-3.78 ± 1.02	-3.12 ± 0.53
	Time and memory			
Peak mem:	2820kb	2820kb	3156kb	2820kb
Total time:	39m44s	6h32m7s	6h39m46s	2h37m26s

Table 6.61: One UCT player using the first UCT version comparing the use of a higher number of simulations and rollouts and a higher number of games

Table 6.61 shows the results of the first UCT version playing against three random players, varying the number of simulations, rollouts and games played. As expected, increasing the total number of rollouts used (be it by increasing the number of simulations or the number of rollouts per simulation) slightly improves the quality of the player. It is interesting to note that increasing the number of simulations results in a less significant improvement of the result compared to increasing the number of rollouts made in each simulation (columns two and three). Furthermore, playing four times more games (column four) approximately decreases the range of the 95% interval by one half (which was to be expected due to the confidence interval being calculated with the use of a square root). Taking a look at time and memory usage, one can observe that increasing the number of simulations and increasing the number of rollouts both result in a similar increase of time usage: the time usage is very exactly ten times larger, which corresponds to the number of total rollouts being ten times larger. The reason for this is that the time needed for computation of the game itself can be neglected in comparison to the computing time of the UCT algorithm. When increasing the number of simulations, the memory usage remains constant, which is due to the fact that the trees

of each simulation are not kept in memory but just the results of each tree are stored and in the end aggregated to the final result. When increasing the number of rollouts though, a little increase of the memory usage can be observed, which is caused by the constructed trees being larger. To better compare the different versions with respect to number of simulations and rollouts, they are tested in direct play against each other.

	Player 1 Options			
# rollouts:	100	100	100	1000
# sim:	10	100	10	10
	Player 3 Options			
Expl const.:	7000	7000	7000	7000
# rollouts:	100	100	1000	100
# sim:	100	10	10	10
	95% confidence intervals for average round scores			
P1 (UCT):	6.53 ± 1.26	8.06 ± 1.23	5.00 ± 1.26	10.51 ± 1.39
P2 (Rnd):	-7.62 ± 1.20	-7.49 ± 1.22	-7.73 ± 1.23	-7.91 ± 1.14
P3 (UCT):	6.93 ± 1.20	4.97 ± 1.28	9.60 ± 1.54	4.08 ± 1.24
P4 (Rnd):	-5.84 ± 1.20	-5.54 ± 1.19	-6.87 ± 1.18	-6.68 ± 1.16

Table 6.62: Two UCT players using the first UCT version comparing the use of different numbers of simulations and rollouts (1)

	Player 1 Options	
# rollouts:	100	1000
# sim:	100	10
	Player 3 Options	
# rollouts:	1000	100
# sim:	10	100
	95% confidence intervals for average round scores	
P1 (UCT):	6.02 ± 1.23	9.47 ± 1.41
P2 (Rnd):	-7.55 ± 1.17	-7.96 ± 1.10
P3 (UCT):	8.80 ± 1.49	5.32 ± 1.25
P4 (Rnd):	-7.27 ± 1.14	-6.83 ± 1.20

Table 6.63: Two UCT players using the first UCT version comparing the use of different numbers of simulations and rollouts (2)

Tables 6.62 and 6.63 show the results of those experiments. The first table shows experiments where the standard version using values of 10/100 for the number of simulations/rollouts is used. Obviously, both increasing either the number of simulations or the number of rollouts (thus increasing the total number of rollouts computed) results in an improvement of the performance. The first table also shows that using a combination of values of 10/1000 yields a larger improvement compared to using 100/100. This supports the observation previously made in the experiments where one UCT player played

against three random players. Also the second table, where the versions using 100/100 and 10/1000 are tested in playing directly together, confirms this observation. Thus, increasing the number of rollouts has a larger impact on the improvement of the performance compared to increasing the number of simulation so that the total number of rollouts is equal. Certainly, these experiments can be refined by only slightly increasing the number of simulations and also increasing the number of rollouts at the same time.

	Player 1 Options		
# rollouts:	1000	10000	1000
	95% confidence intervals for average round scores		
P1 (UCT):	15.01 ± 1.31	17.34 ± 1.58	15.20 ± 0.71
P2 (Rnd):	-5.15 ± 0.77	-5.86 ± 0.79	-5.12 ± 0.42
P3 (Rnd):	-5.42 ± 0.75	-6.12 ± 0.76	-5.26 ± 0.41
P4 (Rnd):	-4.43 ± 0.83	-5.36 ± 0.78	-4.81 ± 0.42

Table 6.64: One UCT player using the second UCT version comparing the use of a higher number of rollouts and a higher number of games

Table 6.64 shows the results of a UCT player using the second UCT version varying the number rollouts and the number of games against three random players. The results confirm the observation made for the first UCT version: increasing the number of rollouts improves the performance of the UCT player and playing more games during a session increases the reliability of the results, as the 95% confidence intervals get smaller. I decided to omit further experiments letting two UCT players with a different number of rollouts play against each other, as the results above are clear enough and as the results for the first UCT version have shown, it is easy to predict that when compared directly, the UCT player using more rollouts will achieve a higher valuer of score points.

	Player 1 Options	
Version:	1	2
	Player 3 Options	
Version:	2	1
	95% confidence intervals for average round scores	
P1 (UCT):	3.93 ± 1.30	9.68 ± 2.18
P2 (Rnd):	-7.71 ± 1.12	-6.92 ± 1.13
P3 (UCT):	10.41 ± 2.24	3.22 ± 1.17
P4 (Rnd):	-6.62 ± 1.20	-5.98 ± 1.14

Table 6.65: Comparison of the two UCT versions where each UCT player uses the best configuration found and a value of 1000 for the number of total rollouts used

The last experiment of this section is the second comparison between both UCT versions, this time not using the baseline UCT players as in Section 6.2, but the best found configuration for each UCT version, though only with the standard number of 1000 total rollouts. The results are shown in Table 6.65. They confirm what has been

observed at many experiments before: the best configuration for the second UCT version clearly outperforms the best configuration for the first UCT version.

6.6 Human player against best UCT player

In this section, the results of an experiment with myself playing against the best configuration found are presented. As the second UCT version clearly outperformed the first one, the second UCT version is the choice for this experiment. As it was not possible for me to "forget" the card deals played, the experiment could unfortunately not be repeated with an inversed players positioning. Anyway, the number of games played is very small so that the obtained results are not too much reliable. I chose to play with only one UCT player and two random players, to avoid any influencing of the UCT players among themselves. Furthermore, I use the tournament settings, i.e. 24 games, but no compulsory solos are played as those would just cost the two random players more negative points and the UCT player and I would have two games less to play normally.

The final standings of the experiment can be found in Table 6.66. The results show that the UCT player can compete with my playing strength (which I believe not to be too bad), although I had one significant advantage: I always knew that players two and four are random players and thus not trustworthy teammates. This information was of course not available in any form for the UCT player. Still, I also did one non-trivial mistakes when I failed to announce a marriage when having both ♣Q, thus playing a silent solo which I lost and which cost me six score points. Anyway, when increasing the number of rollouts to 1000 or even more, the playing strength would certainly be even more improved. E.g. when using 1000 rollouts, the UCT player would already play an aces solo in the very first game. The disadvantage is that one would need to wait seven to ten seconds for the computation of each move of the UCT player.

In the following, I'll summarize what kind of games have been played unless it was a regular game (just for statistical reasons and to demonstrate the quality of the UCT player):

- Game no. 1: Player 2 (random) plays a silent solo (because random players do not check for marriages), of course losing it.
- Game no. 2: I play a marriage and win.
- Game no. 5: The UCT player plays a jacks solo and wins, without having done any announcements.
- Game no. 8: I play a marriage and win.
- Game no. 10: The UCT players plays another jacks solo, announces re, I give kontra, but the UCT player wins closely (128 to 112).
- Game no. 13: Regular game, but both the UCT player and I announced re/kontra and the gamed ended 121 to 119 in favor of the UCT player's team (the re team).

	General Options
# games:	24
Comp. solos?:	no
Random cards:	yes
Random seed:	2013
Ann. version:	2
	Player 3 Options
Version:	2
SP factor:	500
Team points?:	yes
PP Divisor:	1
Expl const.:	20000
# rollouts:	1000
Allow ann.:	only +
Wrong formula?:	no
MC simulation?:	no
Action selection:	1
	Scores
P1 (Hum):	70
P2 (Rnd):	−84
P3 (UCT):	60
P4 (Rnd):	−46
	95% confidence intervals for average round scores
P1 (Hum):	11.67 ± 7.26
P2 (Rnd):	-14.00 ± 5.77
P3 (UCT):	10.00 ± 16.41
P4 (Rnd):	-7.67 ± 9.16

Table 6.66: Tournament of myself playing against the best UCT configuration together with two random players

- Game no. 17: The UCT players plays a clubs solo, announces re and wins.
- Game no. 18: I play a diamonds solo, announce re, the UCT player replies kontra but I win quite distinctly.
- Game no. 19: The UCT player plays a third jacks solo, announces re, I say kontra (holding three jacks), but the UCT player still manages to win.
- Game no. 20: The UCT players plays an aces solo, announces re and wins by making all tricks! One has to admit though that he was very lucky, as one of the random players started by playing the wrong suit and in the end, I chose to keep the Ace of the wrong suit (having a 50% chance to keep the right one).
- Game no. 21: I play a spades solo, announce re, the UCT player says kontra but I win.
- Game no. 22: I fail to announce a marriage and lose silent solo.
- Game no. 24: I play a hearts solo, announce re, the UCT player announces kontra but I win.

Note that I accidentally started playing without limiting the number of games to 24 and thus the 25th game was already started and the UCT player would have played another solo, namely a hearts solo. I think the large number of solo plays in only 24 games is more than amazing, even at a level of good human players, this is not common at all. The UCT player played five solo games, I played three solo games and all of them were won by the soloist. I thus assume that computing whether there is a chance of winning a solo game or not is a strength of the UCT player. A general observation is that the UCT player often did an announcement only in reaction on one of my announcements, i.e. when I did a (sometimes risky) announcement of kontra, he said re, or the other way around. Sometimes, the UCT player's team then also won the game. It happened rarely though that the UCT player did an announcement on his own, at least in a regular game. Whenever he played a solo game, with the exception of the first one, he also announced re.

In order to test if the given configuration of the UCT player is also useful when playing with four "good" player, i.e. when replacing the remaining two random players by two copies of the UCT player, I played another tournament against these three players. No parameters have been changed except that another random seed was used so that I would not know the card deals in advance. The results are shown in Table 6.67. Unfortunately, the results are less in favor of the UCT players, especially not in favor of the one positioned behind me. The reason for the worse results compared to the experiment above is that 17 out of the 24 games played were solo games, and only 6 of those were won. Thus UCT players tend to decide to play a solo game too often; I assume this is due to the very large exploration constant (20000) chosen in the "best" setup. This works as long there are some "bad" players involved as the random players, but soon as the other three players play well together to stop the soloist, they often

fail when playing solo. I assume that optimizing the parameters of the UCT players when always using four UCT players could adjust the problem at hand. Another idea is to prevent the UCT players from playing solo games too often by implementing a rule similar to the announcing rule, which would allow UCT players to play solo games only if all corresponding UCT rewards are positive.

	General Options
Random seed:	2024
	Scores
P1 (Hum):	58
P2 (UCT):	-52
P3 (UCT):	8
P4 (UCT):	-14
	95% confidence intervals for average round scores
P1 (Hum):	9.67 ± 9.94
P2 (UCT):	-8.67 ± 15.75
P3 (UCT):	1.33 ± 8.98
P4 (UCT):	-2.33 ± 15.11

Table 6.67: Tournament of myself playing against the best UCT configuration

Note that the results seem to be worse than they actually are: players three and four are nearly on a same level around 0 score points. In a perfect world, when all players have the same playing strength and play a number of games large enough, the resulting score points should all lie very close to 0. This experiment deviates from this ideal only in the sense that the UCT player on second position lost too many solo games, whereas the other two UCT players managed to overall play in a way that they stayed near the 0 points mark. I played very defensively, only playing a solo once and winning it, therefore explaining why I have a comfortable advance in the final points. It is also important to note that the card playing behavior of the UCT players is most of the time excellent; they start by playing aces of non trump suits if possible, often play the ♡10 on top of the ♣Q, play foxes when their teammate already secured the trick, use a fox or a ♢10 when trumping a non trump suit which is played for the first time and generally play valuable cards if the trick will (probably) be taken by a teammate and play low valued cards if this is not the case. Also the announcing behavior is very passive and reactive, i.e. they mostly make an announcement as a reaction to an announcement made by another player. Additionally, they (nearly) always announce re when playing a solo (which is in 99% of the cases a very good idea because one does not play a solo game if one does not expect to win it).

7 Conclusion

This chapter summarizes this thesis in a first section by pointing out what has been done and by showing the contributions of this work. A second section gives an outlook of what can and still should be done in the future in order to further improve the quality of playing doppelkopf when using the UCT algorithm.

7.1 Summary

This work first introduced doppelkopf, a trick-taking card game with incomplete information. Focusing on its official rules, a motivation for solving doppelkopf was given: the game has a huge state space and and very large strategic depth. Furthermore, there are several other aspects of playing the game which makes it more appealing to investigate the game: the separation into a "game type determination phase" and the "card play phase" in which additionally the problem of good announcing comes into play and the fact that the teams are not known in advance for most of the games played. Solving doppelkopf thus is a non-trivial problem and can be compared to playing bridge or skat. This thesis focused on solving the game using the UCT algorithm, an algorithm based on MCTS with guided action selection, which had shown success in previous applications to other games such as go, Klondike solitaire and skat.

In order to use UCT in combination with an imperfect game, one could just use the belief state space rather then the concrete state space, thus operating on information sets rather than real states, or the missing information could be replaced by sampling worlds. The second solution was the one chosen for this thesis and an algorithm to generate consistent card assignments was presented.

The next part of this thesis explained the implementation of doppelkopf and of the UCT algorithm. Especially the latter one provides a set of numerous options that modifies different aspects of the UCT algorithm, ranging from modifications to how the UCT rewards are calculated over how many iterations of the algorithm should be computed up to the usage of a heuristic to guide action selection in unexplored parts of the search tree.

The final part was dedicated to the presentation of a large number of experiments. First experiments were used to determine two reasonable baseline UCT players, one for each of the two different implemented UCT versions, by tuning the parameters concerning the calculation of UCT rewards and of the UCT formula. The first block of experiments also gave evidence that the "first announcement style version" which results in game trees with a larger branching factor than the second version cause the UCT players to play a lot worse. Furthermore, a first comparison of the two UCT versions

was conducted. The second block of experiments was then dedicated to testing the use of single enhancements, i.e. only one option at the time was changed in comparison to the baseline players. These results were interesting in the sense that the two different UCT versions showed different behavior when enabling the same options. Follow-up experiments tried to combine several of the "good" found options during the previous experiments in order to find a "final" best configuration for each UCT version. Those were then used to test the influence of increasing the number of samples and the number of games played and to compare the two UCT versions again. The final experiment then consisted of myself playing against the best found UCT player. When playing together with two random players, the results were very good and promising, but when replacing the remaining two random players, performance of the UCT players degraded a bit, at least for one of the tree players, namely the one positioned behind me. Still they showed very good card playing skills.

Concluding, the primary goal of this work – which consisted of showing that the UCT algorithm can be applied to doppelkopf and that a player using this algorithm can achieve some good results when compared to a baseline approach, due to the lack of previous research that would have provided better comparison data – was reached. Additionally, a simple UCT player was improved by using several modification of the UCT algorithm. Finally, the best configuration of the UCT player achieved some promising results when playing against a decent human player.

7.2 Outlook

As the approach of using the UCT algorithm for playing doppelkopf was shown to achieve promising results, it is certainly worth to think about how it can further be improved.

A first idea is always to check if the current implementation can be made more efficient in some way, so that the number of rollouts used can be improved without the loss of a player that computes his next move within a few seconds only. More precisely, the memory usage of the program is quite low for the moment and could be increased for the sake of making it faster. Currently, the nodes of the search tree only store the data relevant for the application of the UCT algorithm itself, e.g. the accumulated rewards, which player moves and which move he chooses. Especially, the corresponding belief game state of the UCT player using the algorithm is not stored but always recomputed again in every iteration. If those states could be stored in addition to currently stored information without exhausting the available memory, then this could very well yield a speed up of the computation of UCT players.

Another approach consists of improving the heuristic to better guide the algorithm when choosing between several unvisited successors at a node during the search. For now, the heuristic only affects card play and it is probably not very sophisticated right now. By extending the heuristic to be usable for all move types, the results of an MC simulation or the search in general if all nodes should be added to the tree should be improved, thus enhancing the overall performance of the UCT algorithm. Concrete ideas for the heuristic include to always first choose to make no announcement at announcement

move nodes and to choose not to play solo games at nodes where the player is asked for a reservation. Another good idea would certainly be to first choose to play a marriage when having both ♣Q before exploring any other options like playing a secret solo or another solo game type. Also the card playing heuristic could be extended not to only search for "safe cards" but also to consider more information such as if a non-trump suit can be played twice without being trumped or if playing a specific color is reasonable because the teammate can get rid of a last card of another suit and thus trump this suit later.

A third idea is to use any kind of initializing for unvisited nodes, i.e. as soon as a leaf node is reached or added to the tree, all of its successors receive an initial value of expected UCT rewards. That way, the action selection could always be done by applying the UCT formula for all successors. Such an initialization could be done with the help of heuristics again or with help of expert knowledge. It is not clear though, how any kind of expert knowledge can be that broad that it covers all cases.

Another idea consists of using a kind of "ensemble-UCT" approach also for the second UCT version, similar to the simulation/rollouts approach of the first UCT version. The difference would then consist in the fact that a simulation of the second UCT version would *not* fix a card assignment for all rollouts, but still use a new one for each iteration. Thus the second UCT version would be changed in the way that not only one search tree is constructed, but several. Their result could then be aggregated in a manner similar to how the first UCT version does. Bjarnason et al. [3] tried using this approach in their work on Klondike solitaire, and their results are promising in the fact that the ensemble approach slightly surpasses the normal UCT variant both in the winning rate for Klondike solitaire and the average time needed, however the HOP-UCT approach (which corresponds to the first UCT version in the case of doppelkopf) is a lot faster and obtains nearly as good results. Nevertheless, using an ensemble approach for UCT could be tested also motivated by having different version to trade-off between speed and performance.

Last but not least, it would be interesting to compare the performance of the UCT players to a totally different approach. Due to the lack of previous research on this topic, one could test to play against one of the several commercially available programs or, probably better, test against the open source program FreeDoko (see Chapter 5) which plays solely based on rules and heuristics.

Bibliography

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47:235–256, May 2002.

[2] Ivona Bezáková, Daniel Štefankovič, Vijay V. Vazirani, and Eric Vigoda. Accelerating Simulated Annealing for the Permanent and Combinatorial Counting Problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '06, pages 900–907. ACM, 2006.

[3] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. Lower Bounding Klondike Solitaire with Monte-Carlo Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, ICAPS '09, 2009.

[4] Michael Buro, Jeffrey R. Long, Timothy Furtak, and Nathan Sturtevant. Improving State Evaluation, Inference, and Search in Trick-Based Card Games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 1407–1413, 2009.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[6] Patrick Eyerich, Thomas Keller, and Malte Helmert. High-Quality Policies for the Canadian Traveler's Problem. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 51–58. AAAI Press, july 2010.

[7] Hilmar Finnsson and Yngvi Björnsson. Simulation-Based Approach to General Game Playing. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, pages 259–264. AAAI Press, 2008.

[8] Hilmar Finnsson and Yngvi Björnsson. CadiaPlayer: Search Control Techniques. *KI Journal*, 25(1):9–16, 2011.

[9] Sylvain Gelly and David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[10] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Technical Report 6062, INRIA, November 2006.

[11] Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, ICAPS'12, 2012. To Appear.

[12] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*. Springer Berlin Heidelberg, 2010.

[14] Sebastian Kupferschmid and Malte Helmert. A Skat Player based on Monte-Carlo Simulation. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 135–147. Springer-Verlag, 2007.

[15] Jeffrey Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, 2010.

[16] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3rd edition, 2009.

[17] Jan Schäfer. The UCT Algorithm Applied to Games with Imperfect Information. Master's thesis, Otto-von-Guericke-Universität Magdeburg, July 2008.

[18] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing UCT versus CFR in Simultaneous Games. 2010.

[19] Wikipedia. Doppelkopf — Wikipedia, Die freie Enzyklopädie, 2012. URL `http://de.wikipedia.org/w/index.php?title=Doppelkopf&oldid=99972145`. [Online; accessed 6-March-2012].