Universiteit Leiden
The Netherlands

Opleiding Informatica

Strategic Gambling in

DiminishingBridge

Thijs Snelleman

Supervisors:
Dr. W.A. Kosters & Dr. J.N. van Rijn

BACHELOR THESIS

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

March 3, 2020

Abstract

DiminishingBridge, or better known as "Boerenbridge" in the Netherlands, is a somewhat uncommon card game, known as the American card game "Oh Hell" as well. The goal of the game is for each player to earn as many or lose as few points as possible. At the end of the game, the player with the most points emerges as the victor.

DiminishingBridge is a trick-taking card game with the element of trump cards as well. The game is split in two separate phases, a bidding phase and a playing phase. In this paper we will focus on exploring various agents with different strategies to test their applicability within a game of incomplete information, their performance under various parameters and finally see if they are a worthy competitor against human players. The agents we will discuss are the "Random Agent", "Trumping Agent", "Rule Based Agent", "Monte Carlo Agent" and "Monte Carlo Tree Search Agent". The agents based on the Monte Carlo algorithms have specifically developed evaluation methods to allow the algorithms to work within this card game. Experimental results are presented; they show that the Monte Carlo Agent performs better than the other agents.

Contents

1 Introduction

DiminishingBridge is a trick card game in which players have to estimate the number of tricks they will make. Unlike many similar card games, such as "Rubber Bridge", DiminshingBridge requires the player to achieve the exact number of tricks which they have bid in order to earn *any* points. In this thesis, this card game will be studied, which is a specific version of the card game "Oh Hell". Although this game version's name contains "Bridge", as well as in the Dutch translation ("Farmer's Bridge"), it has less in common with "Contract Bridge" or just "Bridge" than the name would suggest. The game was first described in 1930 by B.C. Westall [Par08].

Figure 1: Example hand in the First Round of DiminishingBridge.

The Dutch version of the game, "Boeren Bridge", is traditionally played with four players and a single deck of (French) cards, starting off by dealing all the cards evenly to the players. Thus each player has thirteen cards at the starting round, see Figure 1. Each new round the players are allocated one card less than the previous round. In the final round of the game all players are given just one card. Each player's aim is to earn as many points each round as possible, by exactly winning as many tricks each round as they have bid. The player with the most points at the end of the last round, wins [BJS13].

By using strategic algorithms, such as Monte Carlo, this game will be analysed, as well as certain other variations, by altering the number of cards, players and other parameters. The agents playing the game will either use one of the strategic algorithms or randomly decide which move to make.

For this thesis, code was developed in C++ and the source code is available at the Github Repository.

This thesis will describe the rules of the game and the parameters allowing for alterations in Section 2, related research in Section 3, strategies and algorithms on DiminishingBridge in Section 4, discussion of results and statistics in Section 5 and lastly, the conclusion of this paper with a discussion for further work can be found in Section 6.

This bachelor thesis was written for the Leiden Institute of Advanced Computer Science at the University of Leiden and was supervised by dr. W.A. Kosters and dr. J.N. van Rijn.

2 Rules of the Game

This chapter will describe the set of rules regarding the game, using several parameters to allow for certain variations, both hypothetical and existing, some of which are described later in this chapter, see Section 2.5

2.1 Set Up

The game is originally played with four players. However, it is possible to play this game with more or fewer players, the minimum requirement being two players. Usually the number of players varies between three and eight. If this game would be played with only two players, one could easily calculate the first round, as any card that one does not have must be in possession of the other player, because the entire deck is dealt in the first round. A third contestant in the game forces it to be unclear which player has what card in the first round, making it impossible to calculate *which* exact player has which card. At the start of the game, a random player is selected to be the dealer, which will be explained later on.

2.2 Cards

Because all cards are dealt in the first round, the number of cards $C(p, r)$ in the game depends on how many players are present and how many rounds are played:

$$C(p, r) = p \cdot r \tag{1}$$

Here p is the number of players and r the number of rounds to be played.

Order and suit Using a French deck of cards, the cards are ordered, cards showing the number 2 as the lowest in order and cards showing the ace the highest, giving the following linearly ordered set:

$$\{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\}$$

Here J, Q, K and A represent the Jack, Queen, King and Ace, respectively. This can be represented as the set of natural numbers $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ correspondingly. The number of different card suits s is in the French deck 4, however this may vary in different (hypothetical) variations, as well as the number of cards within a suit, the order o. The number of suits s can be calculated by:

$$s = \frac{C}{o}$$

or alternatively, the order o can be calculated by:

$$o = \frac{C}{s}$$

Here C is the number of cards in the game as above, o the size of the order as a set of natural numbers. In the original game the values are $C = 52$, $s = 4$ and $o = 13$.

Trumps Trump cards are cards of the suit that will trump any card of any other suit in the game. At the start of each round (excluding the first round, which has no trump suit) the suit of trump cards is decided by picking a random card from the set of cards that is not included this round (i.e., a card that was not distributed to any player but is in the full deck). The selected card is shown to the players, yielding thus more information than just the trump suit of the round, as the set of cards that was not dealt is completely unknown *except* for the card announcing the trump suit.

2.3 Rounds

Each round contains a number of tricks t, depending on which round is being played. The number of tricks is equal to the number of cards in the hand of any player, as the cards are equally distributed, at the start of the round. So in the first round of the original game, $t = 13$, as all 52 cards are distributed among $p = 4$ players.

2.3.1 Dealer

At the start of the game, the order of players is determined, clockwise around the table. The first player in the order is left of the dealer. The dealer is determined randomly in the first round, afterwards the dealer shifts each round to the next player in the order. In the original game, each player is named the dealer three times, except for the player that is named the dealer in the first round. The player that is named the player in the first round is named the dealer in the last round as well, since four players play thirteen rounds.

At each trick, each player plays one card clockwise, starting with the player who won the last trick. In the first trick of each round, the player first in order, that is the player clockwise after the dealer, will start the trick.

2.3.2 Bidding

Bidding is a crucial activity at the start of each round. Each player states, starting at the player first in order, how many tricks they expect or want to make this round. It is the objective of the round for each player to get the *exact* amount of tricks as they bid at the start of the round. Any player that reaches the objective will be awarded a certain number of points, whereas any player that fails to do so will be deducted a certain number of points.

Bidding may technically vary between zero and infinite, although bidding above the number of available tricks in the round would be pointless as one will never be able to achieve this bid in this round of the game. Therefore this option will be excluded and from here on we will continue with $0 \leq b \leq t$, where t is the number of tricks to be made and b the bid to be chosen.

Screw the dealer In DiminishingBridge it is a matter of custom to *screw the dealer*. The dealer is always the last in the order of players, and therefore the last to bid on how many tricks there are to be made. This rule forbids that the sum of bids is equal to the number of tricks to be made, i.e., the dealer is the only player that is sometimes *not* allowed to state one certain value of b in the interval described above. For example, four players play a round of 6 tricks ($t = 6$), with bids so far being 2, 0, 3. This means that the fourth player (dealer) is not allowed to state bid 1, as $2 + 0 + 3 + 1 = 6$, but is allowed to state any other bid that is between $0 \leq b \leq 6$ in this example.

2.3.3 Following Suit

Every trick of every round forces players to "follow suit": The first player to play a card each trick determines the suit of the trick. Any player must play a card from that exact suit, unless the player is not able to do so. If that is the case, the player is allowed to play any card, including trump cards. The player is not obliged to play trump cards when possible in this situation.

Winning a Trick The winner of a trick is decided by firstly applying following suit: Any player that did not follow suit and did not play a trump card instead is ineligible to win the trick. If none were able to follow suit nor played a trump card, it means that the player that started the trick wins. If any cards were played of the trump suit, the player that played the highest order in the trump suit wins the trick. If no trump cards were played the player that played the highest order card that followed suit wins. The player that wins the trick, is the player that will start the next trick as well, allowing them to decide the suit of the trick.

2.4 Awarding and Deducting Points

The award and deduction system differs between variations of "Oh Hell". In the Dutch version, each player that reaches the goal earns a constant factor K, equal to ten points, plus the number of tricks made. Any player that does not reach the goal is deducted $|b - t|$, with b being the tricks bid and t the tricks made; each player is deducted the number of tricks too few or too many made that round. In other versions, more or fewer points may be awarded as starting value, other factors and/or constants are introduced for calculating the score.

2.4.1 Breaking the Rules

In (almost) every version, a certain number of points is deducted from a player when any of the rules are broken (e.g., not following suit, playing a card when it is not one's turn, peeking, ...). We will assume that each player is "perfect": no fouls or forms of cheating are considered, to reduce the possible number of games that are to be calculated by the agents.

2.4.2 Absolute Victory

We define the point system as follows: R is the number of rounds to be played; the number of tricks to be made each round is equal to the round number r, c is the constant point addition for any player that makes the number of tricks that was bid and lastly we define t_f as the "trick factor", which is commonly equal to one. The number of points that can be awarded in a round to a single player is $K + r \cdot t_f$. As the number of rounds in a game is known, we can define the number of points G_p a player can earn in the game as:

$$G_p = \sum_{i=1}^{R} (K + r \cdot t_f)$$

Note that this number may decrease, for instance when the bid of a player is lower than the number of tricks in round i. Secondly we define the maximum of points any player can lose in the game N_p by ℓ_f, the "losing factor", as follows:

$$N_p = \sum_{i=1}^{R} i \cdot \ell_f$$

Note that these formulas can be used at any moment in the game, if one is to replace R, the total number of rounds in the game, with the number of rounds that have yet not been played in the game.

Now that we know the number of points that can be maximally earned and lost by any player at any moment in the game, we can define the "Absolute Victor". This is a situation that a player has earned a certain number of points which makes it impossible to lose for this player before the game is over. Commonly, in many games this would be defined as $A_v = \lfloor G_p/2 \rfloor + 1$, where G_p is calculated with the total number of rounds R. However, earning more than half of the points does not mean one has won, as players may also lose points each round. In order to decide whether the leading player in the game is an "Absolute Victor" we need to compare the leading player's score p_1 with the second best player's score p_2. Then we consider the score difference:

$$S_d = |(p_1 - N_p) - (p_2 + G_p)|$$

The calculated score difference S_d can now be interpreted with the following set of rules:

- If $S_d < 0$ the leading player cannot be declared an "Absolute Victor" as he can still be surpassed.
- If $S_d = 0$ the leading player cannot be declared an "Absolute Victor" as the game can still result in a tie.
- If $S_d > 0$, the leading player *can* be declared an "Absolute Victor" as no player can exceed his number of points in the optimal situation.

To summarize, the concept of an "Absolute Victory" is simply trying to determine whether the leading player has won the game before it even ends. In the original card game, the game is always played until the last round. However, in this thesis a game will be stopped if a player achieves "Absolute Victory" and this will be noted, as "winning the game before it is even over", which is considered to be better than winning in the last round.

2.5 Known Variations

A certain interesting variation of the game is the "Devil's Bridge" variation, where each player is not allowed to look at one's own card in the last round, but only at all the other cards to decide their bid.

A common variation of this game in the Netherlands, the Dutch "Boeren Bridge", is to play from thirteen cards to one, back to thirteen cards. This is basically DiminishingBridge played twice in one game with a continuous scoreboard: once regularly and once inverted (so just "Oh Hell"). This version includes the version of "Devil's Bridge" as well. Although this might be very exciting for human agents, for an algorithmic agent it is just a larger game.

A less common variation is "Luciferen" ("Safety Match"), where players count down and drop a certain number of matches to let the others know how many tricks they are bidding to make. This

does not allow a player to use the information of former bids but does not check for the "screw the dealer" rule.

In Germany the game is known as "Stiche-Raten" ("Trick Guessing") where it can be played with a 64-card deck as well.

3 Related Work

Although the game DiminishingBridge is not a very frequent subject for research, trick-taking card games have been extensively researched for years now by many as "... is a popular research topic, even more so when the problem has been open for some time and the game is actually of interest to players and researchers." [BJS13]. Although DiminishingBridge differs from the regular whist [Wäs05], as it requires an exact number of tricks, not a minimum number of tricks; any trick scored above the bidding is negative for the player.
However, there are some trick-taking card game papers that are interesting for discussion as many other trick-taking games have very similar methods on incomplete information [FB97].

3.1 Search in Games with Incomplete Information

The article [FB97] by Frank and Basin from 1997 approaches the greatest similarity that "Contract Bridge" and DiminishingBridge have, aside from the fact that they are both referred to as "Bridge". As noted before in Section 1, the two trick-taking card games have less in common than one might think, but as both are trick-taking card games they do have common ground when it comes to having *incomplete information*. Frank and Basin point out correctly that *strategy fusion* is simply not possible due to the unknown current state of the world which is "... a constraint typically broken when combining strategies designed for individual worlds." Another observation that we can take from this paper is that the issue of *non-locality* may occur: due to the incomplete information some game states may never be reached and therefore the partial evaluation of the decision tree (the evaluation of the sub-tree) is "considering payoffs in world states that are in fact of no consequence at that position in the tree."

Aside from the manner to treat incomplete information and therefore the game, these two trick-taking card games have very little in common, mainly due to the two-player team system, instead of a "free-for-all", which gives a whole different approach to the game (and therefore game-tree). Similar differences are found quite often with other trick-taking games [Par08].

3.2 The Trick-Taking Game of Klaverjas

In the article by Van Rijn, Takes and Vis the Dutch trick-taking card game "Klaverjas" is discussed [vRTV18]. Although "Klaverjas" is different from DiminishingBridge due to the two-player team concept, the different manner of earning points through certain trick combinations and the card deck containing only 32 cards, they "... consider the task of determining and predicting whether an instance of the game of Klaverjas is winning for a variant of the game" where perfect information is available to the algorithm.

The similarity in "Klaverjas" and DiminishingBridge in this article can be found in the calculation of different configurations (in the initial round), which in our case would be the possible distributions of 52 cards over 4 players (called *hands* in the article) we get a slightly different *Combinatorial Number System*. In our case this would result in: $\binom{52}{13}\binom{39}{13}\binom{26}{13}$. However, this may vary due to the number of players (and suit/order size).

In order to evaluate a certain game state, the authors have used classical minimax search with α-β pruning [RN16], without any additional heuristics. However, they also state that this method

is not a practical method, as they are simply searching for game-state evaluations with perfect information, due to the impractical size of the state-space. As the number of configurations of the game in this thesis is even larger, due to the card deck size, this is applicable to the situation in this thesis as well.

3.3 Monte Carlo on Skat

"Skat", Germany's national card game, is quite similar to DiminishingBridge as well as "Klaverjas". This game is played with 3 players, 4 suits and 32 cards. The number of cards and extensive point calculation system has quite a lot in common with the one for "Klaverjas", however this game is not played in determined teams: during the bidding phase it is decided how for each hand (or round) the teams are formed. Furthermore, instead of randomly selecting the trump, the player that won the bidding phase decides the trump suit, which is also quite different from the DiminishingBridge trump selection method.

In the paper by Kupferschmid and Helmert [KH06] the authors mainly place the focus on the difficulty of the "unknown" or "random" factor in the game, which is similar to DiminishingBridge, due to incomplete information, as discussed before. This is very relevant to this thesis as well, as they correctly address the issue of "statistical error and the inaccuracies" in which an algorithm, in this case Monte Carlo, might prefer a choice in which victory is not guaranteed to a choice in which it is guaranteed, because of the number of points that is to be gained. This translates into our situation only in the bidding part of the game, afterwards the algorithm only needs to worry to try and make as many tricks as was bid, where the number of points to be *earned* is already determined. Another noteworthy point in this article is the differentiation of the card-playing engine and the bidding engine as this effects DiminishingBridge as well in a different manner.

3.4 On Random Forest and AlphaZero

Two related concepts to this thesis are Random Forest [Bre01] and AlphaZero [SHS+18]. For starters, Random Forest is applicable as a possible algorithm for an agent, due to its characteristic that counters overfitting, for instance. This is a useful trait as the game is within incomplete information. Likewise, AlphaZero, could be used as well as it "... can achieve, *tabula rasa*, superhuman performance in many challenging domains." [SHS+18]. They are very potent and intelligent constructs. However, neither of these are used to create an agent for DiminishingBridge in this thesis as they require a great consumption of time for not only their application to this specific situation, but for their calculations for simulating and training as well, not to mention the highly diverse numbers of approaches that could be used to construct any such agent that they could perhaps be a thesis subject on DiminishingBridge on their own.

4 Strategies and Algorithms

This section will discuss the outline of different agents on DiminishingBridge. Most importantly certain decisions will be made clear and will be explained in this section. Note that each agent is split into two parts: the bidding engine and the playing engine.

4.1 Strategic Bidding Engine

The Strategic Bidding Engine is inspired by the "Milton Work Point Count System"[DL01] from bridge and is based on the hand evaluation at the start of each round. In order to decide which bid to make, a pointing system is used to approximate the value of a hand, a high value meaning the hand will probably win many tricks and a low value meaning the opposite. The value of a hand is defined as the sum of the values of all its cards. The value V of a card c is determined using the number of cards in the game C, the card suit s, the order size o, the card rank of the card c_{r}, the order and the current trump suit of this round s_0. In case that $s \neq s_0$ the formula is:

$$V(c) = 1.0 - |c_{\mathrm{r}} - o|/C$$

In case that $s = s_0$ the formula is:

$$V(c) = \max(0, 1.0 - ((c_{\mathrm{r}} - o)^2 + o)/C)$$

The general idea for this pointing system is based on the fact that the highest card in the original game, the ace of the trump suit, is *always* winning. Any player that is dealt this card will never bid lower than one because the ace guarantees that at least one trick will be won, since it can participate in any trick. This explains the 1.0: This card is the base case. However, any trick card after that still has a very good chance of winning; It beats all the other cards of the other suits and not all the other cards of the trump suit *have* to be dealt, hence its effective rank could be higher than its technical rank. Therefore, the quality of cards in the trump suit is linear: $|c_{\mathrm{r}} - o|$ reverts the order and divides it by the number of cards in the game, i.e., it deducts the number of cards that can (technically) beat itself.

In case of cards where their suit is not equal to s_0, the calculation is done quadratic: The order is reversed and then squared so any card further from the ace will become worth less and less. Secondly, we always add the order size o to this value, since any trump will always beat this card. Note that this formula is also used in the first round, when there is no trump suit as it is still a good estimation: The addition of o then works as a correction that owning a card does not mean that a player will be able to play it when desired, e.g., owning the ace of spades when diamonds is trump is useless in the last round if the trick starts with spades (or any suit that is not spades for that matter).

4.2 Monte Carlo Bidding Engine

The Monte Carlo Bidding Engine is a simplified Monte Carlo simulation [RN16], which uses the same budget as the Monte Carlo Agent below, using as input the agent's cards, the starting player order, the round number, the illegal bid if available, and the trump suit, if available. The engine

runs as much simulations as the budget allows and counts each time how many tricks the player has won in the simulation. Afterwards the number of tricks made in total is divided by the number of simulations, selecting the integer closest to the floating point value. If this integer is equal to the illegal bid, the floating point value is rounded off the other way around.

4.3 Random Agent

The Random Agent is not a truly pure random player. In the bidding phase its bidding engine uses the bidding engine as specified in Section 4.1. In the playing phase, however, each trick the playing engine is told which of the cards in its possession are legal moves, from which a card is selected uniformly random. This decision was made due to the fact that a pure random player simply strays too far from any sensible results.

4.3.1 Trumping Agent

In contrast to the random agent, the Trumping Agent is not very pure in its playing phase as it always prefers playing trump cards when possible. They do share the same bidding engine but in the playing phase, the playing engine of the trumping agent is told which moves are legal but then it first tries to find all possible trump cards within the set. If any are possible, a random trump card is selected, otherwise it plays a random card within the set of playable cards.

4.4 Rule Based Agent

The Rule Based Agent is a simple agent using a set of rules to determine which card to play. The idea of the Rule Based Agent is to mimic a human agent with perfect memory. The Rule Based Agent uses the bidding engine from Section 4.1.

After the bidding has been done, the Rule Based Agent will have to decide which card to play each trick. The rule set is split into two parts, based on the number of tricks made t by the agent so far and the agent's bid b; The first set applies if $t < b$, the second applies if $b \geq t$. First the legal moves are calculated at the given point. Secondly, for each card c its *current* card value $V(c)$ is calculated using the perfect memory of the agent.

The card value is calculated using the following formula, with OP the number of cards higher in order played within the suit, $Tr(c)$ that determines as a Boolean whether the card c is a trump card. It is equal to zero if the card is a trump card, or if there is no trump suit present like in the first round, one otherwise. PT is the number of trump cards played so far. Lastly, c_{r} represents the rank of the card and o represents the number of cards in a suit as discussed in Section 2. We put:

$$V(c) = 1.0 - \frac{(o - c_{\mathrm{r}} - OP) + Tr(c) \cdot (o - PT)}{(o - OP) + Tr(c) \cdot (o - PT)}$$

Note that division by zero cannot happen, as we only evaluate cards that have not been played yet. Secondly, note that it is possible for cards to have the same evaluation. Thirdly, in case that a card has been played in the current trick that beats the current card, its value $V(c)$ is multiplied with -1 because this card is definitely not able to win this trick. Fourthly, note that a card valued at

zero could still win a trick when the card is the first card to be played in a trick. For example, if a player is the first to play in the trick and plays the lowest card of a non trump suit, this card is evaluated at zero ("This card cannot win from any other *relevant* card"). But if none of the other players plays this suit or the trump suit, this card still wins. Only cards with a negative value are "guaranteed" to lose. Lastly, note that all the other cards in the possession of the player are considered "played" whilst evaluating $V(c)$, as they are not able to influence the outcome of trick because the player can only play one card. In later rounds, many cards do not partake in the game, influencing the accuracy of this method.

After these calculations, using perfect memory, the agent checks if any of the players that still have to play a card this trick are not able to play this suit *and* are not able to play a trump card, i.e., these players are certainly not able to win this trick.

The card values represent an estimation within incomplete information of the probability that the card wins the current trick. Using these estimations, we apply the following rule sets to decide which card to play. If the rule set determines that two or more cards are eligible, one is selected randomly.

The first rule set is as follows:

1. If all card values are equal to zero or less than zero, a lowest valued card is played.
2. If none of the other players that have not played yet are able to play a winning card or the agent is the last player in the trick, the card with the lowest value, higher than zero is played.
3. Otherwise, a highest valued card is played.

The second rule set is as follows:

1. If there are any cards valued at zero or less, the highest valued card of this set is played.
2. Otherwise, the lowest valued card is played.

4.5 Monte Carlo

The Monte Carlo method [RN16] is a very well-known and popular set of algorithms based on casino statistics. The method is very useful for the research environment due to its vast size: the upper limit of possible rounds is $C!$, which is 52! in the classical variation, approaching $8.07 \cdot 10^{67}$ just for the first round. Note that this number is not a tight upper bound, as some variations may be illegal (or even isomorphic).

There are two separate agents that can be produced with this method: A regular version that gives each option the same number of evaluations, and the Tree Search version which uses a weighting scheme to decide which option should be investigated further at the current situation.

4.5.1 Monte Carlo Agent

The Monte Carlo Agent bids using the Monte Carlo Bidding Engine, described in Section 4.2. First it is important to consider that the agent will start over each round, as each round new cards are dealt: the agent will not search beyond the current round. Secondly, the agent will *not* consider achieving the number that was bid as "winning" and not achieving it at the end of the round "losing". Although this may seem very logical, there is much discrepancy between the two, so we will estimate the quality of a leaf with an evaluation function. The agent will simulate per option a number of games k, the so-called play-outs, to evaluate each move. The game will be continued using only the Random Agents with the input of the current game state. When a leaf has been reached, the round must be evaluated.

This can be done, for instance, by evaluating the scoreboard of the game at the end of the round. We calculate the scoreboard value SV using the score of all players p_1, p_2, ..., p_n, where p_1 has the highest score and p_n the lowest. First we normalize the scores around the agent's score p_i. This method results in any player p_j that has the exact same score as our player p_i to be set to zero as well. This, however, this is losing as well as the game results in a loss for all of the players in case of a tie. Therefore we add one point to any $p_j = 0$ where $p_j \neq p_i$ to represent the fact that this "losing" as well.

Now we must adjust the player or players that are in the first place to have a bigger impact, as these are currently winning, so we replace the highest score p_1 with $p_1 = p_1^{1+(p_1/2 \cdot i + K - 1)}$. Here i is the current round and K the winning constant as described in Section 2.4. This replacement of the highest score is based on the fact that *if* the score distance between the winning player and the agent increases, the impact of this distance on the scoreboard value SV will start to increase exponentially. This impact has been gauged to the maximum distance between the two: if the distance is equal to this number of points, the impact of p_1 will be squared. If the player is in first place, the value of p_1 is equal to zero and therefore nothing happens. Then we calculate SV as follows:

$$SV(s) = -\sum_{P=1}^{p} p_i / i$$

Here the value of player p_i is divided by its rank i and the sum's resulting value is made negative as we are evaluating the scoreboard for a specific player: positive values mean another player has more points than this player and negative points vice versa. However, Monte Carlo is based around "winning" and "losing" results from evaluations, so the SV must be slightly adjusted to estimate whether it is winning or losing. Consider W as the lower limit, i.e., "the worst possible" scoreboard, and B as the upper limit, i.e., "the best possible" scoreboard for the evaluating player. Then we can calculate their scoreboard values SV and adjust the scores accordingly. Considering $SV(W)$ to approach "losing" a game and $SV(B)$ "winning" a game, we can calculate the game value GV of scoreboard s as follows:

$$GV(s) = \frac{SV(s) - SV(W)}{SV(B) - SV(W)}$$

Here $0 \leq GV(s) \leq 1$. Note that scoreboard W is constructed by taking the scoreboard from the root node and adding the maximum achievable points for any player that round to each player that

is not the agent and subtracting the most points that can be lost from the player that is the agent, and B is constructed vice versa. Secondly, note that W is not reachable, as it is impossible for each player to win all the tricks of a round, whereas B is only possible if each player bids the maximum number of tricks in a round, which is very unlikely. Lastly, there is an exemption in which the formula above is not applicable: if in any scoreboard an "Absolute Victor" can be declared, i.e., a player has won the game before it is over, as stated in Section 2.4.2, either GV is equal to one if this victor is the player or zero otherwise.

4.5.2 Monte Carlo Tree Search Agent

Similar to the Monte Carlo Agent, the Monte Carlo Tree Search Agent will bid using the Monte Carlo Bidding Engine, described in Section 4.2.
The Tree Search variant of the Monte Carlo Agent will start out as usual using the options of Selection, to decide which move to investigate further, Expansion, to expand a node with a new child, Simulation, playing k random play-outs and Back propagation, updating the nodes value from the simulation node all the way back to the root. The Monte Carlo Tree Search Agent uses the same evaluation function for leaves as Monte Carlo, however it evaluates nodes differently by conducting the tree search. The node evaluation function is adapted from Kocsis and Szepesvári [KS06]. The original formula, which calculates the probability of each node to be selected, is stated below:

$$\frac{w_i}{n_i} + c_0 \cdot \sqrt{\frac{\ln(N_i)}{n_i}}$$

Here w_i is the number of wins for this node after the move i, n_i is the number of simulations considered for the i-th move, c_0 is the exploration parameter, set to $\sqrt{2}$, and N_i is the total number of simulations in this node as considered from the parent node. However, in our situation it is not only impossible to decide after each round whether a player has won or not, since the game does not end after each round, but the evaluation function returns an evaluation of the scoreboard. In order to decide if a node should be explored, we need a slightly adjusted formula: we replace w_i simply with the sums of the game values GV from the previous section.

4.6 Human Agent

In order to properly estimate the performance of these agents will be illustrated through mass testing against each other. However, the most noteworthy opponent to test these algorithms and strategies on is the Human Agent. To truly illustrate the quality of the Human Agents versus the Computing Agents, these Human Agents will be hand picked from different faculties of Leiden University with some experience in card games. The most interesting of these human agents is Tim van de Paverd, second year student in Economics and Business Economics at the University of Amsterdam, whose skills in card games are a true match for the computer. Although these matches will not be as legendary as Kasparov vs. Deep Blue 1997 [Pan97], van de Paverd does hold several titles in "Contract Bridge" to his name, such as the European Championship in Stokke, Norway in U21 in 2019 [Bon19a] and second place in the World Championship in Opatija, Croatia U21 in 2019 [Bon19b].

5 Results

This section will illustrate the outcome of the various agents described in Section 4, with various parameters where possible. All of the results are focused on a game with four players, as this is the regular playing style of the game where performance of the agents should be optimal.

5.1 Analysis

The various agents are all very different, but a quick selection can be made which agent will outperform another agent through some small analyisis. The results shall suggest the following properties, where $x < y$ means y performs better than x, as follows:

$$RA < TA < RBA < MCTS < MC$$

Here RA is the Random Agent, TA the Trumping Agent, RBA the Rule Based Agent, $MCTS$ the Monte Carlo Tree Search Agent and MC the Monte Carlo Agent. The Random Agent is placed the lowest and each other agent that does its moves strategically should at least perform as well as the Random Agent. The Trumping Agent is a special form of the Random Agent and performs better than the Random Agent, as it has some form of strategy, but very slightly. The Rule Based Agent performs better than either of those as it is an extensive strategic agent. Both the Monte Carlo and Monte Carlo Tree Search Agent should, in general, out perform all the other agents due to their extensive analysis of game states. However, the Monte Carlo Agent performs better than the Monte Carlo Tree Search Agent, due to the fact that the latter treats the game as deterministic even though the game takes places in an environment of incomplete information. Lastly, the Human Agent is not placed anywhere, as the data set is insufficient to make any proper conclusions, which is elucidated in Section 5.7.
Each section below, excluding Section 5.2, is concluded with a significance tests, for which we make a two assumptions [AA18]. The most important assumption is that the population is distributed normally. Secondly, we assume that the obtained data is collected using randomization. We will test the significance using a one sided significance test, where each agent is expected so exceed its regular number of victories, giving the research hypothesis $H_a : \pi > \pi_0$. As the number of players is equal to four, it should properly exceed its fair share against the other three agents, thus $\pi_0 = 0.25$. Furthermore, this agent must have the highest percentage of victories within the data set. Although this method is not perfect, it does give some argument for an agents quality of performance. We assume an α-level of 0.01, which is rather strict but has been chosen because of the great sample sizes. Afterwards a confidence interval is calculated with a z-value of $z = 2.576$ [AA18].

5.2 Random Agent

The Random Agent is the most simplistic agent of all. In Table 1 are the results of 1,000,000 games of four Random Agents.

1,000,000 games	Points Earned	Wins	Absolute Victories
Random Agent 1	29,662,799	244,497	52,776
Random Agent 2	29,652,884	243,968	52,837
Random Agent 3	29,637,495	243,690	52,969
Random Agent 4	29,704,806	244,801	52,768
Total	118,657,984	976,956	211,350

Table 1: Four Random Agents against in one million simulations.

Note that the total number of wins and the total number of games do not necessarily match: Ties are counted as a no win situation for every player. Therefore each agent performs slightly under the 25% mark, with an average proportion over all four agents of 0.24.

5.3 Trumping Agent

The Trumping Agent is slightly different and has been tested by playing it against three Random Agents. The results of 1,000,000 games are in Table 2.

1,000,000 games	Points Earned	Wins	Absolute Victories
Trumping Agent 1	29,924,240	258,732	57,052
Random Agent 2	28,496,628	239,057	50,870
Random Agent 3	28,528,229	239,354	51,152
Random Agent 4	28,572,223	239,658	50,853
Total	115,521,320	976,801	209,927

Table 2: Trumping Agent vs. Random Agents in one million simulations.

If we apply the significance test from Section 5.1 on the number of games won, we get a p-value of $p < 0.00001$, thus $p < \alpha$. The confidence interval for this sample is 0.258732 ± 0.0011, which does not include $H_0 = 0.25$. Thus we can conclude that the Trumping Agent does indeed play better than the Random Agent for this sample with a confidence of 99%. Another interesting note is that the Trumping Agent's Absolute Victories strongly exceeds any of the Random Agent which also shows its superior quality compared to the Random Agent.

5.4 Rule Based Agent

The Rule Based Agent is a far more tactical agent than those described before. Therefore we test it against both the Random Agent and the Trumping Agent. The results of the Rule Based Agent against three Random Agents in 1,000,000 games are in Table 3.

1,000,000 games	Points Earned	Wins	Absolute Victories
Rule Based Agent 1	39,835,140	406,419	128,619
Random Agent 2	27,463,603	190,353	35,658
Random Agent 3	27,460,410	190,137	35,737
Random Agent 4	27,554,878	191,524	35,569
Total	122,314,031	978,433	235,583

Table 3: Rule Based Agent vs. Random Agents in one million simulations.

By applying the significance test from Section 5.1, we find that the p-value in this sample has a value of $p < 0,00001$ and the confidence interval is equal to 0.406419 ± 0.0012 and can therefore conclude with a 99% confidence level that the hypothesis H_a is true, thus the Rule Based Agent exceeds the Random Agent for this sample. Another note of its overall quality compared to the Random Agent is that the Rule Based agent is achieving approximately 31.65% of its wins as absolute victories, compared to the three Random Agents average of 18.70%. Lastly, the Rule Based Agent definitely performs well enough to receive its fair share of points in the game, earning 32.57% of all points. This is considered a good quality in an agent, as this suggests that this agent was able to either reach its bid many times, let others fail theirs or a combination of the two.

Secondly, the Rule Based Agent has been simulated against three Trumping Agents in 1,000,000 games. The results are in Table 4.

1,000,000 games	Points Earned	Wins	Absolute Victories
Rule Based Agent 1	29,084,138	281,947	66,328
Trumping Agent 2	26,392,286	231,117	46,944
Trumping Agent 3	26,448,052	232,556	47,528
Trumping Agent 4	26,445,237	231,620	46,869
Total	108,369,713	977,240	207,669

Table 4: Rule Based Agent vs. Trumping Agents in one million simulations.

As before, we apply the significance test from Section 5.1, giving us a p-value of $p < 0,00001$ and a confidence interval of $0,281947 \pm 0.0012$ which does not include H_0, concluding that for this sample H_a holds as $p < \alpha$ and thus with a confidence of 99% that the Rule Based Agent is better than the Trumping Agent.

The results of these matches have been illustrated in Figure 2, orientated on the Rule Based Agent. Note that the bar on absolute victories is based on how many of the agent's wins were absolute victories.

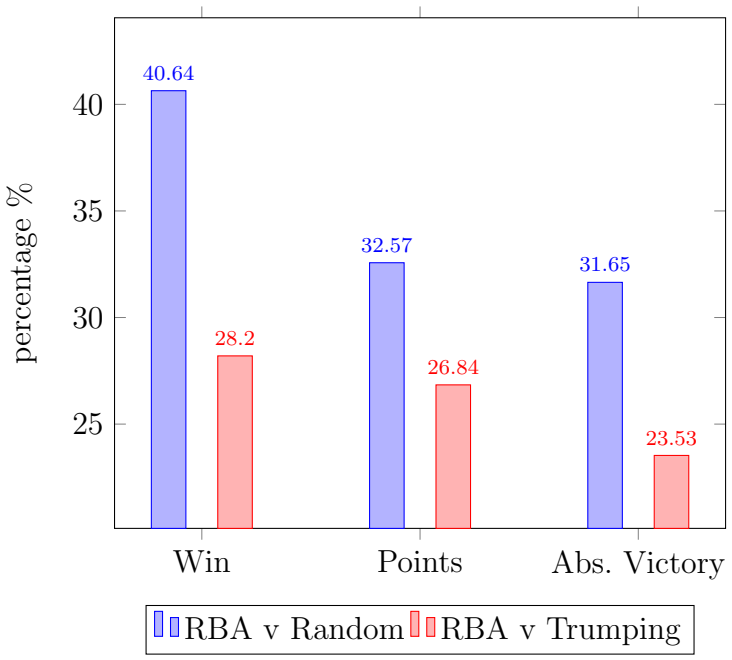

Figure 2: Illustration of the Rule Based Agent's performance.

In Figure 2 it becomes quite clear that the Rule Based Agent had more issues with defeating the Trumping Agent than the Random Agent, which corresponds with the conclusion from the previous section. Overall we can conclude that the Rule Based Agent is a quite capable of winning against either of these agents, and the percentage of points earned suggests that even in losing matches the agent is still able to earn its fair share of points to be gained.

5.5 Monte Carlo Tree Search

The Monte Carlo Tree Search Agent has been tested against three Random Agents. In Table 5 are the results for the agent with various budgets, the number of simulations the algorithm is allowed to do before it is forced to make a decision on which move to make. For every budget we have simulated 10,000 games each. Note that the sample size is significantly smaller than in the previous sections, due to a great increase of consumption time where some calculations took over 48 hours.

MCTS 10,000 games	Points Earned	Total Points	Wins	Absolute Victories
10 Budget	584,626	1,398,648	6,780	3,131
100 Budget	686,886	1,382,381	8,302	5,297
1,000 Budget	747,774	1,377,556	9,077	6,625
10,000 Budget	748,633	1,367,814	9,141	6,686

Table 5: Monte Carlo Tree Search vs. Random Agents over different budgets.

The first thing to be noted is that each column's value, except for total points, increases significantly between the budgets of 10, 100 and 1,000. The increase between 1,000 and 10,000 is only very slight. The significance test is applied to each of the various budget settings, giving each budget the p-value

of $p < 0,00001$. Looking into the confidence interval on each of these budgets, we get 0.678 ± 0.0120, 0.8302 ± 0.0097, 0.9077 ± 0.0075 and $0,9141 \pm 0.0072$. Each of these confidence intervals excludes H_0, and thus we can conclude with 99% confidence that for this sample the Monte Carlo Tree Search Agent is better than the Random Agent for each of these budgets. In Figure 3 the quality of the agent is illustrated with the different budgets. The percentage of absolute victories denotes the percentage of how many wins were absolute victories, the same as before as in Figure 2.

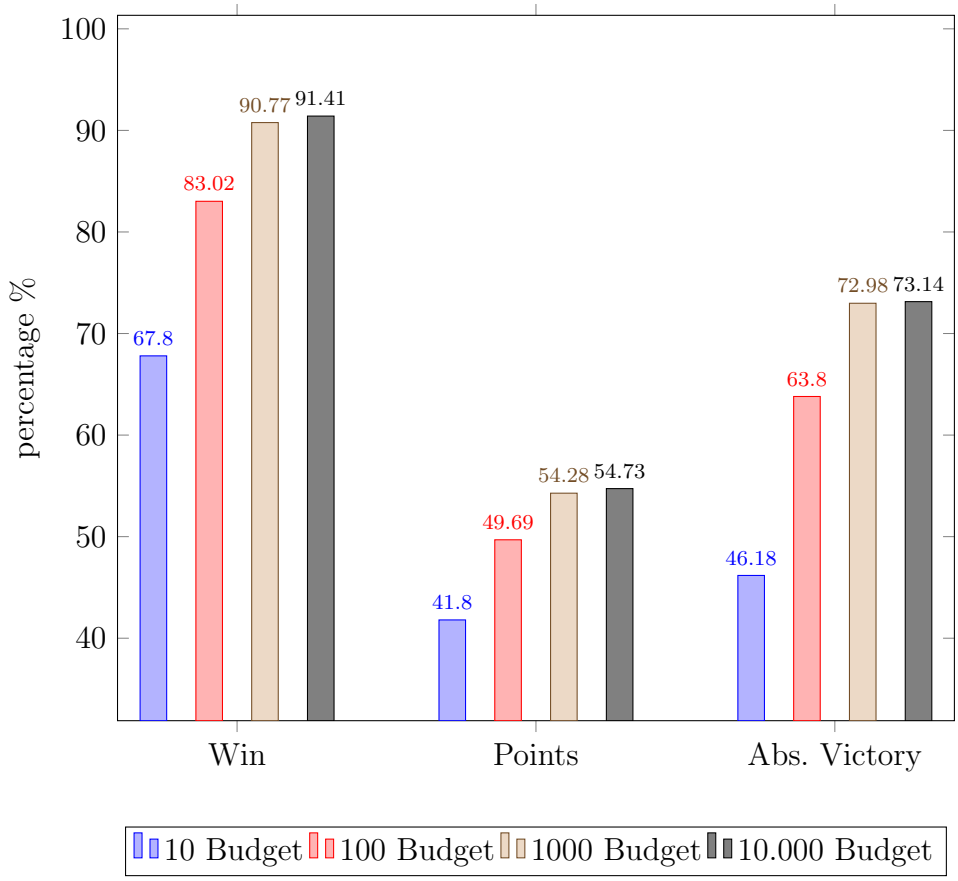

Figure 3: Monte Carlo Tree Search Agent versus three Random Agents.

In the graph the increase of quality of the agent becomes quite clear, with a somewhat visible approach of a pinnacle between the results of the 1,000 budget and the 10,000 budget.

Now we shift the focus for the quality of the Monte Carlo Tree Search Agent to how well it performs against the Rule Based Agent. In Table 6 are the results of 10,000 simulations each on the Monte Carlo Tree Search Agent with various budgets against three Rule Based Agents.

MCTS 10,000 games	Points Earned	Total Points	Wins	Absolute Victories
10 Budget	572,202	1,540,295	5,805	2,253
100 Budget	667,535	1,530,373	7,458	4,036
1,000 Budget	713,269	1,514,534	8,290	5,105
10,000 Budget	710,539	1,511,250	8,304	4,997

Table 6: Monte Carlo Tree Search vs. Rule Based Agents with various budgets.

Like in the previous sample we can denote that the increase between 1,000 budget and 10,000 budget is so slight, and even a minor decrease in the percentage of points and the absolute victories, that the results suggest the agent is approaching an apex. We apply the the significance test of Section 5.1 to each of the budgets. This gives for each of the budgets a p-value of $p < 0.00001$ and a corresponding confidence interval of 0.5805 ± 0.013, 0.7458 ± 0.011, 0.8290 ± 0.010 and 0.8304 ± 0.010, none of which includes H_0, thus concluding that for this sample holds that $p < \alpha$ with a confidence level of 99%. The Monte Carlo Tree Search Algorithm is therefore a better agent than the Rule Based Agent.

In Figure 4 the results of the different budgets are illustrated. As before, the percentage of the absolute victories denotes how many of the wins of the agent were an absolute victory.

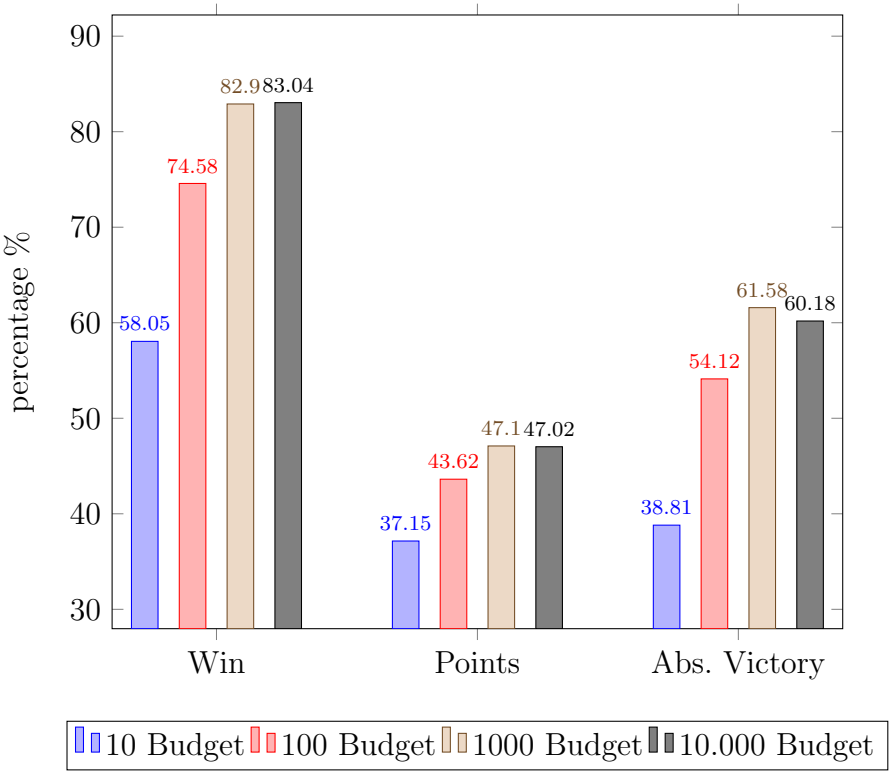

Figure 4: Monte Carlo Tree Search Agent versus three Rule Based Agents.

Overall we can conclude that the Monte Carlo Tree Search Agent is an agent that exceeds all previously tested agents and that the budget approaches somewhat of an apex when increasing it further, suggesting that improvement is not to be found per se in higher budgets.

5.6 Monte Carlo

The final agent we investigate is the Monte Carlo Agent, hypothesised to be the best agent of all. In Table 5 are the results of the Monte Carlo Agent with various budgets against three Random Agents.

Monte Carlo 10,000 games	Points Earned	Total Points	Wins	Absolute Victories
10 Budget	651,161	1,396,151	7,831	4,463
100 Budget	742,923	1,355,448	9,110	6,737
1,000 Budget	782,501	1,338,886	9,481	7,644
10,000 Budget	793,910	1,325,708	9,570	7,858

Table 7: Monte Carlo Agent vs. Random Agents with various budgets.

From these results we calculate the significance test, giving us a p-value of $p < 0.00001$ for each budget. This gives us a confidence interval of 0.7831 ± 0.012, 0.9110 ± 0.007, 0.9481 ± 0.006 and 0.9570 ± 0.005, none of which include H_0, thus concluding that the Monte Carlo Agent is, for this sample, a better agent than the Random Agent.
The information in Table 8 is visualised in Figure 5, comparing the qualities of the different budgets for the agent.

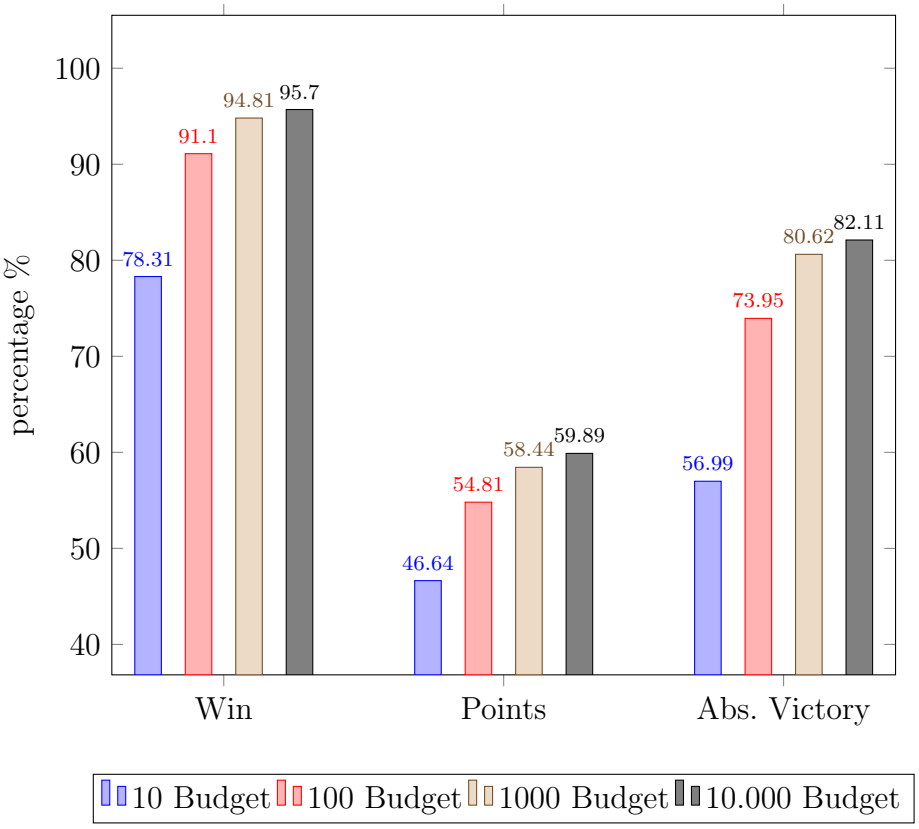

Figure 5: Monte Carlo Agent versus three Random Agents.

Now we have seen in Section 5.5 that the Monte Carlo Tree Search Agent heavily outperforms the

Random Agent, but in Table 8 we can see that the Monte Carlo Agent does as well, and seemingly even slightly better. If we compare the two proportions [AA18] on their best results, thus placing the focus solely on the 10,000 budget samples, we get an estimated standard error se of $se = 0.0035$ for which a confidence interval can be calculated, that is equal to $(0.9570 - 0.9141) \pm 2.576 \cdot (se)$, giving 0.429 ± 0.0089 as a 99% confidence interval. As this contains only positive values, we can conclude that the Monte Carlo Agent's proportion in winning games against the Random Agents is indeed larger than the Monte Carlo Tree Search Agent, suggesting that the first agent is better.

It has become quite clear that the Monte Carlo Agent and Monte Carlo Tree Search Agent outperform any other agent discussed so far. The results from these agents against the three Random Agents point out that the Monte Carlo Agent is objectively better at defeating Random Agents than the Monte Carlo Tree Search Agent. However, it is also important to test these agents against one another. Since it is hypothesised that the Monte Carlo Agent will be a significantly better agent, the results in Table 8 are on one Monte Carlo Agent versus three Monte Carlo Tree Search Agents. Note that all of the agents have the same budget in each simulation of 10,000 games. Furthermore note that only the results of the Monte Carlo Agent are recorded, as it is hypothesised that this agent should be better than the other three and therefore we keep the focus on these results.

MC v MCTS 10,000 games	Points Earned	Total Points	Wins	Absolute Victories
10 Budget	680,293	2,407,151	3,798	1,046
100 Budget	770,891	2,828,870	3,459	775
1,000 Budget	820,787	3,111,627	3,208	491
10,000 Budget	834,771	3,119,666	3,558	504

Table 8: Monte Carlo Agent vs. Monte Carlo Tree Search Agents with various budgets.

The first note to make on this data is the surge between the different budgets, as the Monte Carlo Agent seems to have a harder time to compete with the Monte Carlo Tree Search Agents as the budget increases, but has a slight resurge at 10,000 budget. If we apply the significance test from Section 5.1 we get the p-value of $p < 0.00001$ for each of the budgets and the confidence intervals of 0.3798 ± 0.013, 0.3459 ± 0.012, 0.3208 ± 0.012 and 0.3558 ± 0.012 respectively. As none of these contain H_0, we can say with 99% confidence that H_a holds and therefore can conclude that for this sample, for each of the budgets the Monte Carlo Agent is a better agent than the Monte Carlo Tree Search Agents.

In Figure 6 the results of the Monte Carlo Agent versus the three Monte Carlo Tree Search Agents is illustrated. As before, the absolute victory is the percentage of games won by the Monte Carlo Agent that were an absolute victory.

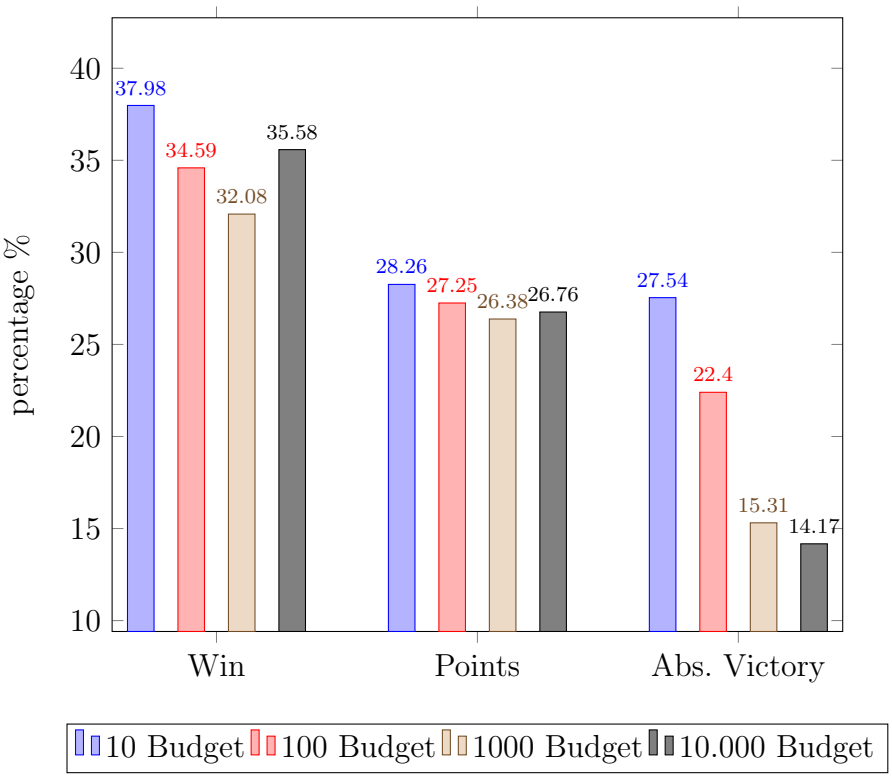

Figure 6: Monte Carlo Agent versus three MCTS Agents.

The surge of wins through the different budgets is quite clear in Figure 6, especially in the percentage of games won but also slightly in the percentage of points. In the absolute victory the reduction in shrinkage seems to come to a halt.

Overall we can conclude that the Monte Carlo Agent is the best agent of all, with a highly superior benchmark against the Random Agent, and defeating three Monte Carlo Tree Search Agents with a proper distance.

5.7 Human vs. Monte Carlo

Finally, the Monte Carlo Agent, nicknamed "Tessa" during the matches by the other players, has been tested against three human players in four matches with a budget of 10,000. Even though is is a rather small size for a number of tests, especially in comparison with the number of simulations ran in the previous sections, it should be noted that playing a match of DiminishingBridge is rather exhausting and each match takes quite a long time, each match taking about one hour to an hour and a half. The results of these matches can be found in Table 9. Note that the column "Earnable Points" refers to how many points the individual could achieve and "Rounds" refers to the number of rounds this player succeeded in achieving their bid. Lastly, there is no column on Absolute Victories as none occurred in any of the matches.

Four games	Points	Earnable Points	Rounds	Games Won
Monte Carlo Agent ("Tessa")	362	589	35/52	2
Paverd	293	602	24/52	0
Student 1	292	622	28/52	1
Student 2	300	608	29/52	1

Table 9: Monte Carlo Agent "Tessa" vs. Human Agents.

Although the population size of the data is far too small to make any good statistical inferences [AA18], it is proper to point out that "Tessa" has achieved more points, 61.4%, than any other player has, achieved the most rounds, 67.3%, and won two of the four games. It would be too eager to hypothesise that the agent is better than the human player, but these results definitely suggest that it can hold its own against a human. Finally, the performance of Paverd is not a very expected result as every other player has a better results than Paverd. This, however, can be explained by the fact that the other two human players at the start realised that they were at a disadvantage against the other two players, resulting in, as far as this is possible within DiminishingBridge, team play, preferring to see either of them not achieving their bid more than the third player.

6 Conclusions and Future Work

In this thesis we analysed various agents for DiminishingBridge. The agents that were developed had very different tactics and play styles. Overall it can be concluded that each agent has their own qualities but these samples suggest that certain agents perform better than others when simulated to play against other agents. Furthermore, the Monte Carlo Agent, the best agent, has achieved victory against humans twice out of four games and has a victory percentage against other agents of 32.08% to 37.98% against the Monte Carlo Tree Search Agents and 78.31% to 95.7% against the Random Agents.

In future work some simple exploration could be done in empirically inferring different possible exploration constants for the Monte Carlo Tree Search Agent. Another possible options is to develop a new Monte Carlo Tree Search Agent using Information Set Monte Carlo Tree Search [CPW12], as this version of the Monte Carlo Tree Search Algorithm is dedicated for environments within incomplete information.
Another possible approach would be to search for a different evaluation function, as described in Section 4.5.1. This evaluation function is highly biased to achieve more points than the player in first place, where it might be more interesting to introduce a stronger favor to not allowing other players to achieve their bid as well, which could possibly correspond more to the playing style of human players. Whether the latter is favorable or not is unclear. Another option would be to try and deduce even more information from cards that have been played by other players, to predict what cards are logical that they will play next using the information on whether or not they have achieved their bid currently, which is an even more humane play style. Lastly, in any future work the effect of humans playing against a computer could be analysed, as humans may have a conscious or subconscious bias against a computer player .

For further research, the source code is available in C++ at the Github Repository, `https://github.com/thijssnelleman/DiminishingBridge`.

References

[AA18] J. Miller A. Agresti, B. Finlay. *Statistical Methods for the Social Sciences*. Pearson, 4th edition, 2018.

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

[Bon19a] Nederlandse Bridge Bond. Nederlandse aspiranten Europees Kampioen. https://www.bridge.nl/nieuws/nederlandse-aspiranten-europees-kampioen/, 2019. [Online; accessed 2-December-2019].

[Bon19b] Nederlandse Bridge Bond. Twee keer zilver, een keer brons bij WK jeugd 2019. https://www.bridge.nl/nieuws/wk-jeugd-2019/, 2019. [Online; accessed 2-December-2019].

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[CPW12] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set Monte Carlo tree search. *Transactions on Computational Intelligence and AI in Games*, pages 120–143, 2012.

[DL01] N. Kantar, D. Dimitrescu and M. Lundqvist. *Bridge Classic and Modern Conventions, Volume I.* Example Product Manufacturer, 2001.

[FB97] Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 87–123, 1997.

[KH06] Sebastian Kupferschmid and Malte Helmert. A Skat player based on Monte Carlo simulation. In *Proceedings of the International Conference on Computers and Games*, pages 135–147. Springer, 2006.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, pages 282–293, 2006.

[Pan97] Bruce Pandolfini. *Kasparov and Deep Blue: The Historic Chess Match Between Man and Machine*. Touchstone, 1997.

[Par08] David Parlett. *The Penguin Book of Card Games*. Penguin Books Ltd, second revised edition, 2008.

[RN16] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, third edition, 2016.

[SHS+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[vRTV18] J.N. van Rijn, F.W. Takes, and J.K. Vis. Computing and predicting winning hands in the trick-taking game of Klaverjas. In *Proceedings of the 30th Benelux Conference on Artificial Intelligence*, pages 106–120, 2018.

[Wäs05] Johan Wästlund. A solution of two-person single-suit whist. *The Electronic Journal of Combinatorics*, 12:291–320, 2005.