Monte Carlo Machiavelli

G. Groeneweg

15th June 2005

Abstract

This paper explores the card game Machiavelli.[1] In this game, many turns have to be played and many actions can be carried out per turn, which results in a large game tree. Because traditional search methods will take too much time to play the game in a reasonable amount of time, this paper deals with applying Monte Carlo techniques with domain-dependent information to be able to develop heuristics that provide a winning strategy. In a tournament, an AI-player played against a player using Monte Carlo, where the AI-player used the heuristics that are developed in this present research. The result of the tournament has shown that the use of Monte Carlo simulation, where the evaluation function on non-terminal positions is based on playing random games from that node on, performs well. Monte Carlo simulation even outperforms the AI-player that has been developed.

keywords: Monte Carlo, Machiavelli, Progressive Pruning, Student-T test

1 Introduction

Many games, like Go [5], Poker [4] and Scrabble [14] have been studied by different approaches: using tree search, simulated annealing and Monte Carlo. This last approach has been studied by Brügmann [8], when developing an AI-player for the game Go. Monte Carlo Go programs have performed well on 9×9 boards. This paper explores the use of Monte Carlo techniques in Machiavelli, an imperfect-information card game.[2] In imperfect-information games, a part of the actual state of the game may be unknown. In Machiavelli, the cards players have in their hand are hidden from the other players. Furthermore, using domain-dependent knowledge with Monte Carlo has been very effective [7].

[1] Machiavelli is a registered trademark of Hans im Glück Verlags GmbH, 80809 München.

[2] This game is also known as Citadelles (French), Citadels (English) and Ohne Furcht und Adel (German).

The term *Monte Carlo* refers to the use of the pseudo-random function of the computer and averaging outcomes to select moves [10]. Abramson [1] has proposed the expected-outcome model, in which the proper evaluation of a game-tree node is the expected value of the game's outcome given random play from that node on. The use of Monte Carlo and combining it with progressive pruning has led to writing the present research. Monte Carlo and progressive-pruning techniques are combined to eventually develop an AI-player for the game Machiavelli, a game that has never been extensively studied to the best of our knowledge. Therefore, the knowledge about Machiavelli is still in early stages. The following problem statement has been formulated for this paper.

Which heuristics can be developed to provide a winning strategy using Monte Carlo simulation in the game of Machiavelli?

The following research questions are part of this problem statement:

- How can Monte Carlo support in studying the imperfect-information game Machiavelli?
- Which heuristics provide a better strategy than others?
- How can tree search be used in combination with Monte Carlo?
- How does a Monte Carlo simulation player compare to the knowledge-based player that is developed in the present research?

The difficulty of building a good evaluation function on non-terminal positions and the time complexity of a global tree search have caused to study tree search together with Monte Carlo in the game Machiavelli. The use of tree search and Monte Carlo enables the program (AI-player) to use both domain-dependent knowledge and Monte Carlo approaches. Domain-dependent knowledge assures the elimination of tactically bad moves. The Monte Carlo techniques enable to speed up the gameplay.

The paper starts with a description of Monte Carlo techniques in Section 2. This section also describes the simulation environment that has been developed for this paper and the techniques that have been used to speed

up the process of simulating. Section 3 describes the experiments that have been done in order to develop heuristics for a winning strategy for Machiavelli. Section 4 gathers the results of experiments proving the relevance of this approach. In Section 5, results are provided from the outcomes of the tournament that is held to determine whether the developed AI-player outperforms the Monte Carlo player. Section 6 discusses the questions that have occurred using this approach. Section 7 contains the conclusions.

2 Approach

In this section, the game of Machiavelli and the Monte Carlo techniques used are explained. To see how these two can be combined to create a strategy for a Machiavelli player, a game simulation engine has been made for the present research. Finally, some AI techniques will be described that are used to develop a winning strategy. In Section 2.1 the game Machiavelli will be explained. In Section 2.2 the use of Monte Carlo in this paper and in related work is described. Section 2.3 describes the work of Abramson [1]. In Section 2.4, an explanation is provided about the developed simulation. Section 2.5 ends with a description of the tree-search algorithm that has been used.

2.1 Machiavelli

Machiavelli is a card game, which can be played by 2 to 7 players. It consists of 8 character cards, 64 district cards and 30 gold pieces. Every game round, a person picks a character card, with which he[3] plays that round. Every character has its own benefits. The characters come to turn in a predefined sequence, namely the Assassin, Thief, Magician, King, Bishop, Merchant, Architect and Warlord. If it is the character's turn, the player reveals himself as being that character. A turn consists of two phases, the income phase (collecting cards or gold pieces) and the building phase (character can build at least 1 district card). At any time during his turn, a player can choose to use his character's benefit. These benefits are:

- Assassin: assassinating another character
- Thief: robbing another character
- Magician: exchanging all district cards with another player, or one by one with the card deck
- King: starting next round with choosing a new character and collects from yellow districts
- Bishop: being protected from Warlord and collects from blue districts
- Merchant: always collecting an extra gold piece and collects from green districts
- Architect: getting two extra district cards and is allowed to build up to three districts per turn instead of one
- Warlord: destroying one district from another player and collecting from red districts

After all the characters have had their turn, a new game round begins by choosing new characters, and the game continues like this, until at least one player has built eight districts. Then the game ends and the winner is the one with the most district card points. Bonus points are added for being the first to build eight districts, building districts of all colors. For a complete overview of all the game rules, see Bergsma [3]. The amount of points that a district card scores, is the same amount as the gold that was paid in order to build the district. The rules of Machiavelli differ from country to country. This paper uses the original rules by Faidutti [2] and concentrates its analysis on the seven-player version of the game. The purple district cards always have an exra benefit for the player that has built this districts. These extra features are not taken into account in the game simulation engine.

2.2 Monte Carlo and related work

The principle behind Monte Carlo simulation is that the behavior of a statistic in random samples can be assessed by the empiricial process of actually drawing lots of random samples and observing this behavior [13]. The use of Monte Carlo simulation is very natural for games containing either randomness or hidden information, like Machiavelli.

Brügmann [8] was the first person to develop a Go program based on random games and Monte Carlo. The program is very easy to understand, namely, he designed the player to base his decision in a certain game state (read: tree node) by playing lots of random games from that node on and score each game. The action or move the program plays, is the move that has best performed playing the random games. Monte Carlo has already been used in computer games. The information that the opponent has, is hidden; in order to be able to say something about this information, one can simulate a lot of scenarios. Abramson [1] analysed this and his results are described in Section 2.3.

2.3 Expected-outcome model

Abramson [1] developed a method for complete, two-player, perfect-information games. He proposed the expected-outcome model which he showed to be a powerful heuristic: the proper evaluation of a game-tree node is the expected value of the game's outcome given random play from that node on. He showed that this model

[3] In this paper, the male form will be used when the person referred to could be either male or female.

is *precise, accurate, easily estimable, efficiently calculable and domain independent.* In 1990, he tried the expected-outcome model on the game of 6×6 Othello. The ever-increasing power of computers now enables Go programs to use this model.

This paper is based on the works of Brügmann and Abramson and differs in two important components. First, there are more than two players in this version of Machiavelli and second, Machiavelli is an imperfect-information game. In Section 2.4, the simulation environment will be described for this game in which several heuristics are developed to find a winning strategy.

2.4 Simulation

Together with Bergsma [3], I have developed a game simulation engine in the programming language Java which has all the game rules implemented. When at least one player has managed to build 8 districts (according to the game rules that are dealt with by the simulation engine), the game round is finished and the scores are counted (points on the district cards). All decision points in the game are played randomly, from the choice of the characters to the sequence of possible actions a player can do. This means all players have an equal chance of winning the game. This simulation is the basis upon which the heuristics for winning strategies will be developed. The outcomes of random game-play after 10,000 simulated games of Machiavelli result in an average score of 15 points for all players. If the number of games is set higher, the outcomes for the players will converge.

2.5 Progressive pruning

The technique of progressive pruning [5] has been used within the program Machiavelli and the Monte Carlo simulation environment. The basic idea is the following: in a certain game state, a player has several actions to carry out. To decide which is the best, the player plays each of these actions from that node on for a certain number of times (for example, 100 times per action). An action is being pruned in that node, as soon it is statistically inferior to another action. This process stops either when there is only one action left (then that is the one to choose) or when a maximum number of iterations has been reached. In these two cases, the action with the highest outcome is chosen. To decide whether an action is statistically inferior to another, the Student-T Test is performed, which will be discussed later in this paper.

Here is an example of how the progressive pruning algorithm works. First, as shown on the left side in Figure 1, a player has now 4 possible actions to carry out. The computer simulates, using Monte Carlo, and is able to prune two actions. These two actions are statistically worse than the other two. After the pruning at depth one, the player has four possible actions per expanded

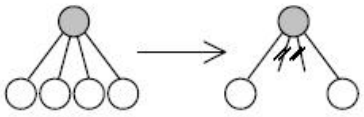

Figure 1: Performing depth 1 [6].

node, that is eight in total. On the right in Figure 2 five (sub)actions can be pruned. In Figure 3 interior nodes

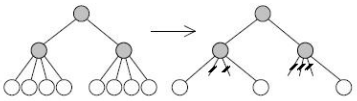

Figure 2: Performing depth 2 [6].

are pruned, which leaves the right of figure 3. There is now only one node left at the top, so the algorithm stops and chooses that corresponding action.

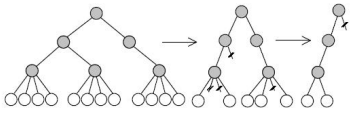

Figure 3: Performing depth 3 [6].

The trees in Machiavelli have a depth of about 14 to 15. Per node, a player can usually choose between four actions per turn:

- income
- draw district cards
- special character benefits
- build

To sum up, the algorithm is similar to iterative deepening [12]. It stops when there is only one remaining action left, or when the maximum depth has been reached. At the root of the tree the goal is to find the best move. The full tree is stored in the computer's memory in order to perform all updates. This algorithm has been used in the way described above, but also the pruning has been tested without the use of the Student T-test. That means, the best move has been chosen, even if it is not statistically better than the second best move. In Section 4 the difference between these two sorts of pruning will be shown.

Student-T test

The t-distribution is used extensively in problems that deal with inference about the population mean or in

problems that involve comparative samples (in cases where one is trying to determine whether the means from two samples are signifcantly different). In Machiavelli, this test can help to determine whether for example murdering a character is significantly better than stealing from another character, just by evalutating the outcomes of these actions. The algorithm that is used in this paper is as follows:
suppose a player has 2 possible actions, X and Y. After having simulated these two actions, X and Y have n game outcomes:

$$X = x_1, ..., x_n \quad (1)$$

$$Y = y_1, ..., y_n. \quad (2)$$

The H_0 hypothesis is that action X is not better than action Y. If it is possible to reject the H_0 hypothesis, then X and Y significantly differ from each other. The H_0 hypothesis can be translated into the following equation for the mean value μ:

$$\mu = 0. \quad (3)$$

Now we can use the t-distribution with $(n-1)$ degrees of freedom to calculate the t-value:

$$t = \frac{z - \mu}{\frac{\sigma}{\sqrt{n}}}, \quad (4)$$

where

$$z = \frac{\sum_{i=1}^{n}(x_i - y_i)}{n} \quad (5)$$

and σ is the standard deviation and n is the number of simulations. Now we have to compute $\frac{\sigma^2}{n}$. Since σ is unknown, we have to use an estimator for σ^2:

$$\widehat{\sigma^2} = \frac{\sum_{i=1}^{n}((x_i - y_i) - z)^2}{n - 1}. \quad (6)$$

If the t-value ≤ 1.8125 ($n = 10$), then the H_0 hypothesis can be rejected with a 90 percent probability and the data is significantly different from each other. This means that action X is better than action Y [9].

3 Experimental setup

To be able to create a heuristic player that plays the game based on the knowedge of Machiavelli, several heuristics have been experimented with. This section provides nine experiments (each experiment deals with one heuristic) that have been carried out in order to develop a winning heuristic player. Each of these heuristics is implemented in a Bot-player, named after that heuristic. All the heuristics have been tested with the game simulation engine that has been developed for this paper. The heuristics can be divided into two groups, general game heuristics and specific game heuristics. General heuristics are heuristics that can be applied at any time in the game, independent of the scores of the other players during the game. Specific heuristics are heuristics that can only be applied in the game in a certain game state, and most of the time these heuristics deal with the position of other players (like destroying a district from the person that has the highest score, instead of destroying at random). In Table 1 all the experiments and thus heuristics, including Bot-name for that heuristic and the goal of that heuristic, are shown.

Nr.	Bot-Name	Goal of heuristic
1	Bot-MonteC	Using Monte Carlo simulation as evaluation function
2	Bot-Frequency	Determining the power of each character
3	Bot-Gold	Deciding whether to choose money or district cards
4	Bot-Build	Determining a winning building strategy
5	Bot-Color	Choosing a character based on the colors of built districts
6	Bot-Missing	Deciding which district card to choose in income phase
7	Bot-Trading	Determining the player to trade cards with
8	Bot-Destroy	Determining the district to destroy
9	Bot-Choice	Choosing a character based on the game-state

Table 1: Experimented heuristics.

In Section 4, a clear description of each heuristic and its results are provided. The tests are conducted in such a way that the first player uses the described heuristic and the other players in the experiments are named Bot-soFar, which means these players have the most updated knowledge so far. The only difference between player one and Bot-soFar is the heuristic to be tested. If the Bot with the new heuristic defeats the other players (average outcomes over 5000 games), the new heuristic is implemented in all the players (Bot-soFar). This way of experimenting has been used throughout this paper. At the end, Bot-soFar has all the knowledge that has been developed in this present research.

4 Heuristic results

This section describes each experiment from Sectionexperimentalsetup and provide the results of each of these experiments.

General heuristic 1: Bot-MonteC

In this experiment, Bot-MonteC has the ability to "freeze" the game just before he has to choose a charac-

Player	No PP	PP
Bot-MonteC	24.01	23.04
Bot-soFar	14.90	12.25
Bot-soFar	16.29	10.55
Bot-soFar	15.93	11.60
Bot-soFar	16.13	14.60
Bot-soFar	17.38	15.45
Bot-soFar	18.04	11.85

Table 2: Effect of using Monte Carlo simulation during the game. PP: Progressive pruning.

Character	Power
Merchant	20,22
Thief	15.98
Bishop	15.79
Architect	13.71
Warlord	13.18
Assassin	13.07
Magician	12.61
King	11.03

Table 3: The power of the characters.

ter. After he has "frozen" the game, he plays the game 120 times per character he can choose. The character with the best average outcomes will be chosen. In this way, pruning the game tree is very easy, the character with the highest outcome will be chosen, the other characters are directly pruned. This is not progressive pruning. That method is used when Bot-MonteC wants the character which is significantly the best. To be able to choose the significantly best player, more games have to be played, which takes more time. A comparison is made between using progressive pruning during game-play or not.

In Table 2 the results of using Monte Carlo as an evaluation function can be found. This shows that when using Monte Carlo in this way, a good computer opponent can be created. Bot-MonteC is doing much better than the other players. In the second column, the average outcomes can be found. These outcomes are the outcomes without using progressive pruning. The third column shows the outcomes when using progressive pruning with significance. As can be seen, the difference between Bot-MonteC and the other players are more visible: the difference in the second column between best and second best player is smaller than in the third column. Still, the third column, where progressive pruning has been used, is not significantly better than the second one.

General heuristic 2: Bot-Frequency

Another general heuristic is choosing a character with a certain frequency. For example, what will the outcomes be if the merchant is always chosen by a player? To carry out this experiment, a player chooses the same character everytime, when available. This experiment shows that in the 7-player version of Machiavelli the Merchant and the Thief are the best characters. This is shown in Table 3. This results from their capability to get more gold pieces. The Merchant always gets an extra gold piece and the Thief has a chance of $\frac{6}{7}$ on stealing extra gold from another player. A reaction of other players can be to assassinate these characters during the game. It is remarkable that the Bishop is on the 3rd place in this list, because this character has no strong benefits, except for being protected from the Warlord.

General heuristic 3: Bot-Gold

In the beginning of a player's turn, he can decide to take two gold pieces or take two district cards, of which he can keep one. A natural assumption is that taking only gold or only district cards at the beginning of a player's turn, is not effective. A player needs gold to build districts and districts to pay gold for. The only other way to collect gold is choosing the Thief character or to collect extra bonus gold for built districts. To receive district cards is not as difficult. The magician can trade hands with a player of its own choice (not a character, which would be harder), or by choosing the architect, who gets two free district cards. This experiment shows if it is preferable to choose gold over district cards. Bot-Gold implements heuristics that can be formulated to decide whether to choose gold or district cards. For example, always take gold, unless no district cards available, or always take gold, unless enough gold is available to build the most expensive building.

Figure 4 shows that choosing gold nine out of ten times in the income phase leads to the highest outcome of the game. This means collecting gold is more important than collecting district cards in the income phase. It is much easier to get district cards during the game than gold. On the other hand, when all players choose gold

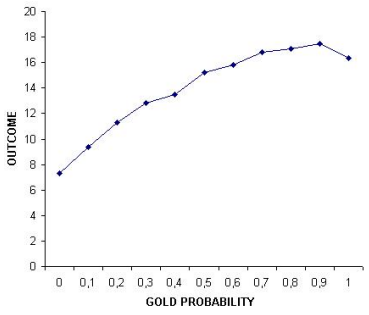

Figure 4: Outcomes of the game as a function of collecting gold pieces.

pieces nine out of ten times, the results of experimenting

Player	Expensive	Cheap	Gold vs. Build
Bot-Build	15.23	13.47	22.04
Bot-soFar	17.48	17.65	18.7
Bot-soFar	17.44	17.29	18.56
Bot-soFar	17.75	17.26	17.35
Bot-soFar	17.45	17.94	16.76
Bot-soFar	17.44	17.83	18.51
Bot-soFar	16.97	17.46	19.16

Table 4: Effect of building most expensive or cheapest districts, or both.

Player	Choose by color
Bot-Color	13.47
Bot-soFar	17.65
Bot-soFar	17.28
Bot-soFar	17.26
Bot-soFar	17.94
Bot-soFar	17.83
Bot-soFar	17.46

Table 5: Effect of choosing a character, based on colors of built districts.

with this issue have shown that it is also best for player 1 to choose gold as much as the other players, in order to perform as well as possible by just using this heuristic.

General heuristic 4: Bot-Build

Two possible heuristics can be to either build only the cheapest districts, or to build only the most expensive districts. Another one can be to build the most expensive district if possible, otherwise, to build the cheapest. Experiments can show whether there is a winning strategy or which strategy is most favourable.

If Bot-Build always builds the most expensive districts, he will not accomplish the building of eight districts before anyone else does. If Bot-Build always builds his cheapest districts, building will go fast, but the built districts are easy targets for the Warlord. As expected, Table 4 shows that both of these heuristics are not very strong. Comparing the two heuristics, building expensive districts is a better strategy (Bot-Build ends up with an average outcome of 15.23, instead of 13.47). In the 4th column is shown that a combination of two general heuristics can lead to a winning heuristic: Bot-Build always takes gold, unless he has collected enough gold to pay his most expensive district (which he will build).

General heuristic 5: Bot-Color

The final general heuristic consists of choosing a character in a certain situation. An experiment that has been conducted is choosing the character based upon the districts that are already built. This assumption can be a good one, because if one has built three yellow districts and he chooses the King as character for that game round, he collects three extra gold pieces. Of course, the Assassin plays an important role in this heuristic. If a player has three blue districts in front of him, there is a pretty good chance he has chosen the Bishop character. The Assassin knows this. The same goes for the Thief. The hypothesis to be examined will be that choosing a character purely based upon the built districts from a player, will not be a very good heuristic.

As can be seen in Table 5, this is indeed not a very good heuristic to play at all times. This can be explained by the fact that Bot-Color is an easy target for the Assassin or the Thief. If Bot-Color has three yellow districts in front of him and the King is available, all the other players will know the character choice of Bot-Color.

This section shows that there is no single general heuristic with a winning strategy. If there was one, the game should be too easy to win, just by always applying the same action. The merchant shows to be a powerful character, but a player is never capable of choosing always this character, and if he is, he has a good chance of being assassinated, because choosing the merchant is a very obvious and easy to detect strategy. The following four experiments are done with specific heuristics, which are more interesting heuristics for an AI-player

Specific heuristic 6: Bot-Missing

Every district card has a certain color; in total there are five colors. At the end of the game, if one has at least one district of each color, the player gets a bonus of 3 points. An experiment has been done to see what happens if a card with a missing color (if available) is chosen. The assumption that can be made with this experiment is that it is always better to choose such a missing card. But if the value of such a missing district is not high, it can be more preferrable to choose a card with a color that is already built.

As experiments have shown, th heuristic where you always choose a missing color card is not a very good one, because these missing cards can sometimes have a low score. But, the heuristic where a missing color card is chosen unless the other available card is worth 2 or more points, is much more interesting. This can be seen in the second column in Table 6. Here, Bot-Missing has, together with one other Bot, the highest outcome.

Specific heuristic 7: Bot-Trade

The Magician's benefit is to trade cards one by one with the bank, or all together with another player. This can be very useful if a player has none or just one card, because then he can trade with another player who has more district cards in his hand. An experiment should

Player	Missing Colors	Player	Trading
Bot-Missing	17.42	Bot-Trade	22.80
Bot-soFar	17.33	bot-soFar	22.38
Bot-soFar	17.40	bot-soFar	22.47
Bot-soFar	17.38	bot-soFar	22.19
Bot-soFar	17.24	bot-soFar	21.02
Bot-soFar	17.42	bot-soFar	21.97
Bot-soFar	17.30	bot-soFar	22.18

Table 6: Effect of cards- and trading-heuristics.

Player	HS	CD,HS	ED,HS	MC	BD
Bot-Destroy	19.17	19.48	18.40	21.61	21.48
Bot-soFar	19.05	19.26	19.39	22.82	22.05
Bot-soFar	19.09	18.84	19.20	22.44	21.89
Bot-soFar	19.44	19.05	19.21	21.89	21.96
Bot-soFar	16.16	18.90	18.81	22.03	22.21
Bot-soFar	19.36	19.27	18.94	21.98	22.05
Bot-soFar	19.13	18.86	19.01	22.06	21.68

Table 7: Effect of destroying heuristics. HS: highest score; CD,HS: cheapest district and highest score; ED,HS: expensive district and highest score; MC: missing color; BD: built districts.

show the benefit of not just trading with a random player, but with the player that has the largest number of cards in his hand.

Trading cards with the player that has the largest number of cards in his hand is a very good one, it directly leads to a winning advantage, as can be seen in the third column in Table 6.

Specific heuristic 8: Bot-Destroy

The Warlord has a powerful benefit, namely, he can destroy any built district of any player. The price he has to pay for this is the price of the district to destroy minus 1. If a player has chosen the Warlord, he has several targets. To determine the target to destroy is a complex choice and there are many options. Some simplistic destroying-heuristics are studied.

Table 7 summarizes the results. Destroying a random building from the player with the highest score has an average result (second column), but destroying his cheapest district results in a winning strategy (third column). The advantage of destroying a cheap district is that the Warlord keeps enough gold to build his districts. That is why destroying his most expensive building does not result in a good strategy (fourth column). The fifth column shows the results of the heuristic to destroy the building of the player that has the largest number of different colored districts. The last column contains the results of destroying a district from the player that is closest to building eight districts.

Specific heuristic 9: Bot-Choice

Probably the most important part of the game Machiavelli is the choice of a character. If one chooses a character that everybody wants in that round, it is probable that this character will be assassinated or robbed. If a player chooses a character that has not many benefits, he will probably not score many points. This means there is a trade-off between the choice of a character and the danger of choosing that character. An experiment has been set up to develop a heuristic that will choose a character in an "intelligent" way. Every character has its own benefits at each game state. This state determines which character is best to choose at that time. To determine the game state, all variables have to be screened in order to make a good character choice. For example, the colors of the built districts help determine the character choice, but also the amount of gold and the characters that are still available. All these variables determine the weight each character gets. These weights determine the character to be chosen (character with highest weight is chosen). The following are examples of knowledge that are checked each time Bot-Choice has to choose a character. These form the basis of the weight update.

- Colors of built districts
- Colors of districts in the hand of the player
- Amount of gold
- Number of built districts
- Number of district cards in hand
- Number of available characters
- Utility of characters at that game state

An experiment has been carried out where player 1 (from now on the player with this heuristic is called Bot-Choice) has all this knowledge and the other players do not. The hypothesis is that the player that has all this knowledge, can reason much better than the players that do not have this knowledge.

Bot-Choice implements the above heuristic. When Bot-Choice has to choose a character, he checks all these variables and if one of these variables is positive for some character, 1 point is added to the weight of this character. In Table 8 the results are shown where Bot-Choice has all this game state knowledge, and the other players do not. Bot-Choice outperforms the other players.

5 Tournament results

To make a final judgement of the several heuristics that have been experimented and tested, a final tournament shows the final results. In this tournament, several players compete against each other. In total, 5000 games are played, the player who has on average the highest score wins. Bot-KnowAll is the final updated Bot-soFar, with

Player	Outcome
Bot-Choice	23.61
Bot-soFar	21.74
Bot-soFar	21.83
Bot-soFar	21.78
Bot-soFar	21.63
Bot-soFar	21.81
Bot-soFar	21.24

Table 8: Effect of knowing the complete game-state.

Player	0 sims	3 sims	10 sims	50 sims
Bot-KnowAll	23.24	23.04	22.95	22.47
Bot-Chad	22.61	22.70	22.30	21.04
Bot-Choice	15.31	15.13	14.87	15.03
Bot-Gold	19.52	19.22	19.58	18.56
Bot-Destroy	18.31	18.74	18.32	18.07
Bot-Build	19.38	19.45	19.19	18.86
Bot-MonteC	18.09	21.63	23.59	25.69

Table 9: Final tournament: different players (different heuristics) compete against each other. Sims is the number of times Bot-MonteC can simulate during the game.

all the up-to-date winning heuristics implemented. This player does not act randomly, all actions are dealt with using knowledge of the several game states. Bot-Chad is a player that is based partly on the player developed by W.L. Sims, who wrote the intelligence behind this Bot. The character-choice heuristic has been used, the other actions a player can do, are copied from Bot-KnowAll. All the other bots are described in this paper. The results of this tournament can be found in Table 9.

Bot-KnowAll wins the tournament, if Bot-MonteC is not able to simulate at all or to simulate the game at each tree node three times (see column 2 and 3). If more simulations per available action are allowed, for example 10 (column 4) or 50 (column 5) simulations, then Bot-MonteC beats Bot-KnowAll.

6 Discussion

Frank and Basin [11] investigated Monte Carlo sampling in games with incomplete information. They showed that even for very simple game trees the chance of finding the optimal strategy with Monte Carlo sampling rapidly approaches zero as the number of moves in the game increases. In their paper, they make use of binary trees and two-player games, which makes it different from Machiavelli. In the test results of Machiavelli, the opposite can be found. A player that uses Monte Carlo sampling without any further knowledge of Machiavelli defeats a player who possesses all the knowledge of the game and uses that knowledge. In order to win, that player has to, at least, simulate every action 10 times. This is an interesting result, which means that the use of Monte Carlo in Machiavelli provides a winning strategy and therefore a very powerful player (although the simulating has a time penalty). Therefore, the conclusions of Frank and Basin do not hold for Machiavelli, assuming the given heuristics are good enough.

The main advantage of Monte Carlo is that it uses very little knowledge. The Monte Carlo simulation has been developed within 2 weeks. One only has to efficiently implement the rules of Machiavelli. Without using Monte Carlo simulation, the player that combines all the heuristics in order to choose his action, is an overall winner. Most important in this winning strategy is the choice of a character. The character is very important in Machiavelli. It determines when it is the player's turn, what he can do and how much he can possibly score in that game round. If he chooses an attractive character (overall, or attractive in his situation), the chance of getting assassinated or robbed is high, but if he survives, he can make considerable progress. If he chooses a non-attractive character, the chance of survival is high, but his progress is slow. Machiavelli is all about this trade-off between greediness and preservation. The best heuristics can be found if:

1. the player keeps in mind this trade-off
2. the player can do useful actions with an acceptable amount of risk

For both of these heuristics an average score of at least 2 points higher was found, compared with not using this strategy. An example of the first can be found in the choice of a character. An example of the second one is trading cards with a player who has more cards.

7 Conclusions and further work

In this paper, a Monte Carlo approach to the computer version of Machiavelli has been described. Next to this approach, several heuristics have been developed that, if combined together, form an AI-player. In the beginning of this paper, the problem statement was to find heuristics that can be developed to provide a winning strategy and how Monte Carlo supports the development of a Machiavelli computer player. On the one hand, there is the Monte Carlo player who uses very little domain-dependent knowledge. This knowledge only consists of preventing the player from taking actions that have a negative effect on that player. On the other hand, there is a player who has far more knowledge, Bot-KnowAll. This player knows the winning heuristics, like finding a good combination of collecting gold pieces and building districts, destroying buildings of the player with the highest score, trading cards with a player that has the largest number of cards in his hand and trying to collect district cards of a color that the player does not have in

his hand and has not yet been built. Finally, powerful characters are the Merchant and the Thief, because they both have the ability to collect extra gold pieces, next to their regular features. Heuristics that have a negative effect on the outcome of a player are to always build the cheapest or most expensive districts or to choose one's character only by looking at the colors from the districts one has built so far.

The most important aspect of this game is the choice of a character. This action depends on many variables and all these variables should be weighted in order to choose the right character. When it is obvious to choose a certain character, the other players probably also know this character is precious for one player and they can prevent him easily from scoring points by assassinating or robbing. Playing the game Machiavelli does not mean to constantly do one thing or the other. In order to win, a combination of strategies is needed. This results in a game-play where one should react differently under the same circumstances in order to win. The heuristics that are dealt with in this paper always react in the same way in the same game-state, therefore it is important to introduce a random input. Otherwise a player's strategy would be too easy to predict.

When playing 1000 games of Machiavelli with seven players, each using some heuristics, and one player using them all, the result is that the player that has the most knowledge wins. If one of these opponents is a player that is capable of using Monte Carlo simulation and progressive pruning, the situation changes. The Monte Carlo simulation outperforms the knowledge-based player if it can simulate a game node for at least ten times, in order to choose the action that has on average the highest outcome.

Machiavelli is a game that has not yet been extensively studied, to the best of my knowledge. This makes it hard to compare findings and results. One way of being more capable of comparing is to implement the simulation engine in such a way that human players can play too. If they can compete, a far better measurement can be made about the AI-player that has been developed now. When playing against humans, the introduction of *moods* can be useful in order to remember what players did. These moods help dealing with the trade-off between being careful and choosing a daring character. Is it angry, and should it choose the Assassin in order to kill his opponent? Or has the player almost won the game and should the player be afraid of getting assassinated or robbed? Or, the final distinction in moods, is the player loosing, and should he show more courage? By choosing more daring characters, he can still win.

At this moment, the weight-updating in the choice of a character heuristic is very straight-forward. If one of the variables has a positive feature for a character, its weight is always updated with 1. This means, that having a yellow building (which means the weight of the King's character gets higher) is just as important as choosing the Thief when there is a lot of gold in the game. No research has been done to determine the best weight update for each variable. It would make more sense to give more weight to the Architect, with six built districts (probability of winning the game is high), than to the King (because there are only two yellow districts in front of him).

For now, game-play with Bot-MonteC is still slow, but with the ever increasing power of computers, the Monte Carlo approach is promising for the future. Bot-KnowAll defeats the Monte Carlo player when Bot-MonteC is not able to simulate games during game-play for more than nine times. When Bot-MonteC is able to simulate more games during game-play, he outperforms the heuristic AI-player which can use all the knowledge there is to know about the game and the game-state.

References

[1] Abramson, B. (1990). Expected Outcome: a General Model of Static Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 182 – 193.

[2] B. Faidutti (2005). Citadels. `http://www.faidutti.com/`.

[3] Bergsma, M. (2005). Opponent Modeling in Machiavelli. *Submitted to BA-KECS 2005.*

[4] Billings, D., Davidson, A., Schaeffer, J., and Szafron, D. (2002). The Challenge of Poker. *Artificial Intelligence*, Vol. 134, No. 1-2, pp. 201 – 240.

[5] Bouzy, B. and Helmstetter, B. (2003). Developments on Monte Carlo Go. *Advances in Computer Games (9)*, pp. 150 – 165, Graz, Austria.

[6] Bouzy, B. (1999). Complex Games in Practice. *Game Programming Workshop in Japan '99, editors: R. Grimbergen, I. Frank, M. Mueller*, pp. 53 – 60, Hakone, Kanagawa, Japan.

[7] Bouzy, B. (2004). Associating Domain-Dependent Knowledge and Monte Carlo Approaches within a Go Program. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 182 – 193.

[8] Brugmann, B. (1993). Monte Carlo Go. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 182 – 193.

[9] Cohen, P.R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA.

[10] Fishman, G.S. (1995). *Monte Carlo: Concepts, Algorithms and Applications*. Springer-Verlag, New York.

[11] Frank, Ian, Basin, David, and Matsubara, H. (1998). Finding optimal strategies for imperfect information games. *Proceedings of AAAI-98*, pp. 500 – 507.

[12] Korf, R.E. (1985). Depth-first Iterative Deepening: an optimal admissable tree search. *Artificial Intelligence*, Vol. 27, No. 1, pp. 97 – 109.

[13] Mooney, C.Z. (1997). *Monte Carlo Simulation*. Sage Publications, Thousand Oaks, London.

[14] Sheppard, B. (2003). Artificial Intelligence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 182 – 193.