Universiteit Leiden
The Netherlands

Opleiding Informatica

A Computer Program that Plays the Card Game Tysiąc

Kyran van der Laan, s2615762

Supervisor:
Prof.dr. A. Plaat
Second reader:
dr. J.K. Vis

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 11/07/2022

Abstract

The Polish card game Tysiąc has not been explored in the field of computers playing games before. In this thesis we perform a first exploration of the game, and we create a computer player that plays against humans. Humans are unpredictable opponents that pose interesting challenges for a computer to play against. The computer program is be able to beat humans that play at beginner level with it winning 5 out of 6 games against humans and it wins 100% of the games played against simple agents. This program is knowledge-based using rules and states of the game at a given time. This program can be further improved upon by creating programs based on other methods of playing games such as exploring trees or neural networks.

Contents

1 Introduction

1.1 Subject

Tysiąc is a card game which is played in many different countries around the world. The word *Tysiąc* directly translates to the number 1000, which is number of points needed to win the game. We were unable to find much information about the origin of Tysiąc. According to Wikipedia the game is believed to have come from the national Austrian card game called “Schnapsen” [Wik22]. This game is also called Russian Schnapsen among other variations. The game somewhat resembles the dutch card games of “Klaverjassen”. The game is a trick-taking card game which is easy to learn. A trick is a small round in which each player plays a card from their hand after which a winner of that trick is determined. After this, the winner of the previous trick leads the next trick. This is done until all cards have been played and the total points for each player can be counted.

Figure 1: A round of Tysiąc

In this thesis the following research question will be addressed:

> How can you construct a knowledge-based program for the card game Tysiąc that is able to beat beginner human players?

To the best of our knowledge, the Polish version of the game of Tysiąc has not been modeled in computer programs to play against others which makes it a good topic for a thesis.

1.2 Problems

The main research question can be divided into sub research questions. These sub-questions are:

1. What is known about methods of letting a computer program play a card game?

2. Based on the known methods, how can we create a program that plays the card game Tysiąc?

3. How well does the program perform against different opponents?

The first sub-problem is an important one because one must know the possible ways of letting a computer play a card game before trying to create such a program. There is a long history in computers playing card games so there are also many methods of creating such programs [SNT98].

The design question comes after the decision of what way of implementation is chosen. In order to analyse the program and understand the results the working of the code must be understood. This is why the design will be explained along an explanation of the different functions and the inner workings of the code.

Last is the question of performance. In order to evaluate the quality of the written program multiple statistics will be accumulated and evaluated. Writing a program that plays a certain game is not that difficult, the difficulty resides in writing a program that plays a certain game *well*. Because the understanding of "good" play within card games is different according to a persons perspective, multiple experiments will be performed against different adversaries to create an as objectively as possible quality of play assessment.

By answering the research questions, the broader question of *"How can you construct a knowledge-based program for the card game Tysiąc that is able to beat beginner human players?"* is answerable.

1.3 Method

The questions posed in this thesis will be answered by literature research and experiments. The main computer program will be created in the programming language C++. This is done because of personal preference and experience and also because this language is efficient for running large amounts of experiments. The supported structures within C++ also help with the creation of the program which is further elaborated on in Section 3.2. The research questions are answered by performing multiple experiments on different opponents. By creating other agents and letting the computer program play a certain amount of games against those agents, an insight in the ability of the computer program to play Tysiąc is formed. A certain number of games is also played against humans to give an estimate against real players. Further explanation on how to program is designed can be read in chapter 3 of this thesis.

1.4 Overview

This thesis consists of the following chapters:

1. Introduction.
   In this Chapter the thesis will be introduced along with the research question and the sub-questions. The method of the work done in this thesis is also given.

2. Related work.
   The related work Chapter consists of necessary background information that is important to know for our thesis. The rules of Tysiąc can also be found in this section which provide the rules and duration of the game according to the Polish version.

3. Design.
   The design Chapter explains the different design choices made in the creation of the program and how the program works. This means that the class used and the different functions will be explained and will elaborated on.

4. Experiments.
   In the experiments Chapter the complete setup and outcomes of the experiments can be found.

5. Conclusion.
   The conclusion Chapter summarizes the entire thesis and gives and reiterates the answer to the questions posed in this thesis.

2 Related Work

In this chapter a short survey of relevant literature is provided. The background of Artificial Intelligence (AI) and games is an important factor to consider before designing our computer program. Furthermore, the rules of the game Tysiąc are explained. At the end the characteristics of Tysiąc will be explained.

2.1 AI and games

An important realizations to be made before starting on our project, is that this isn't the first research of this type. Much previous work has been performed in the field of AI and games.

As stated in the book "Artificial intelligence and Games" [YT18], this all started when A. Turing used the MiniMax algorithm to play the game of Chess after which A. S. Douglas successfully created the first program that mastered a game. The game in question was tic-tac-toe [YT18]. After these programs, people went on to create more algorithms for playing computer games. In 1959, A. L. Samuel released a paper in a journal about the first program that used machine learning to play checkers [Sam59]. Another success came with Deep Blue, the computer that defeated the then reigning world champion chess player Garry Kasparov in 1997 using the alpha beta algorithm [CHJH02]. Many of these programs explore large search trees to find possible outcomes of certain moves. These different outcomes are evaluated and the move with the best evaluation is made. These type of programs work best with games of perfect information as is apparent by the amount of games which are being solved or played at high levels by AI even years ago [vUv02].

2.2 Perfect and Imperfect Information

Games of perfect information are games where the game "state" is fully observable. This means that for each state the game can be, each possible move can be observed. Good examples of this are games like chess and checkers where every possible move can be easily determined by looking at the board. For these perfect information games the search tree that is used to evaluate possible moves can be created up to a reasonable depth. This depth is determined by the constrains on the system which can consist of time needed to calculate or computational power which relies on the hardware used in the system. Games in which the game is not fully observable are games with imperfect information. These are games such as Contact Bridge or Poker in which one does not know what cards the other players hold. Solving games with imperfect information are computationally more expensive to solve than games with perfect information [BML93]. The game of Tysiąc is a game of imperfect information because one does not know which cards the other players hold. For such games trees can be made. The trees for perfect and imperfect information games, see Figure 2 , look quite similar with one difference. In the imperfect information tree there is a dotted line pairing different nodes. This represents the uncertainty of a certain state. A player does not know which node has been chosen and must act using other strategies such as chance calculations. The white and black nodes correspond to two different players respectively and an edge is a move.

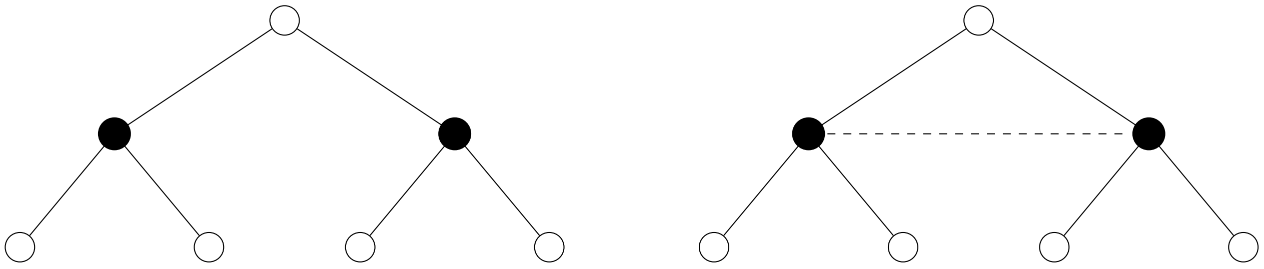

Figure 2: Perfect and imperfect information trees. The imperfect information tree possesses an uncertainty of the black- state

2.3 Imperfect Information Games

As stated earlier, the game of Tysiąc is not a game of perfect information. This significantly alters the way one should design a program. This is because games of imperfect information are games in which one does not all the information such as the cards the opponents are holding. As a direct result of this, the number of possible moves and outcomes can become large. In the magazine titled “AI magazine”, the following is stated about the game bridge: “it would include about 5.6×10^{44} leaf nodes in the worst case and about 2.3×10^{24} leaf nodes in the average case ” [SNT98]. The card game bridge is not the same as the card game Tysiąc, but this is a clear indication of the size some of the trees can become. This is not a reason why a program using search is not possible, it means it would be computationally more challenging than a program with perfect information. The reasoning for not choosing such a program lies within the boundaries of beating beginner human adversaries and not having to compete at a high worldwide level. More on why this approach was not chosen can be read in Section 2.4.

2.4 Emulating Human Play

It is clear that there are distinct methods in letting a computer program play card games. A concise answer to the question of what is known about methods of letting a computer program play a card game is either using algorithms with trees, creating a learning AI or a third one which is emulating how a human plays. This last approach was used for the program, named “*BRIDGE BARON*”, that played bridge [SNT98]. This program successfully won the world championship in contact Bridge which clearly shows its capability. The program uses a number of logic rules and emulation of human play to play the game. This capability lead to the decision of creating the computer program for the game Tysiąc in a similar fashion. More about the design of the program can be read in Chapter 3.

2.5 Rules of Tysiąc

The rules that are followed by the program in this paper and are played in the experiments in Section 4 are taken from the website pagat[1] along with the necessary alternate rules which are also described on the same site. This is because the game Tysiąc is played in different ways, alongside the fact that not many translations of the original Polish version exist. The rules of the game consist of various parts which will be explained in line with the course of the game.

2.5.1 Points

The game of Tysiąc is played with a regular card deck consisting of the cards nine, ten, jack, queen, king and ace. For each suit there is one of each card bringing the total game deck to 24 cards. Each card has its own points which are counted after a game. These points from most to least are as follows:

- Ace: 11 points
- Ten: 10 points
- King: 4 points
- Queen: 3 points
- Jack: 2 points
- Nine: 0 points

One important point to take notice of is that Tysiąc is an ace-ten card game. This means that the ten beats all other cards except the ace which should not go unnoticed.

Further points can be scored by announcing a marriage during the game. Marriages consist of both the king and queen of the same suit and can only be announced during the game if a person holds both the king and queen of that marriage and if either of these cards are played. The points for announcing a marriage are as follows:

♥ Marriage of Hearts: 100 points
♦ Marriage of Diamonds: 80 points
♣ Marriage of Clubs: 60 points
♠ Marriage of Spades: 40 points

2.5.2 Dealing

The cards are dealt starting with the player left of the dealer and going clockwise. There is also a "*prikup*" that contains three cards in the middle of the table. These cards are dealt face down after dealing the second person his card. This is done for the first three rounds of dealing after which the cards are simply distributed clockwise until everyone has 7 cards. After a full round has been played, the cards will be dealt by the player left of the previous dealer.

[1]https://www.pagat.com/marriage/1000.html

2.5.3 Bidding

After the cards have been dealt, the bidding begins. In the bidding phase the players bid the number of points they believe they will get after playing all the cards. The bidding starts at the player left of the dealer and must start at 100. Each bid must be made in increments of 10. A player can choose to either pass or outbid the previous bid. Since 120 is the total value of points in the deck, a bid higher than that may only be made if the players hand contains a marriage. If none of the players make a bid, the dealer becomes the highest bidder with the minimum bid of 100.

2.5.4 Prikup

The highest bidder of the round may expose the three prikup cards for everyone to see and add them to his own hand. After this, he or she must give the other two players one card each of his own hand meaning that every player ends up with 8 cards before starting the game. The highest bidder may lead in to the first "*trick*", which is the word for a round in which everyone plays one card.

After giving away the two cards, the highest bidder may increase his bid to any higher value that is a multiple of 10. A rare occurrence is when a player has all four nines in their hand at which point this player may demand a reshuffle with the current dealer staying the same.

2.5.5 Playing

Having stated previously, the highest bidder will lead the first trick. So long as no marriages have been declared, the first card drawn will decide the suit to follow in that trick. This means that if a player starts the trick with an ace of hearts, every player must follow suit if possible. If a player does not have a card of the leading suit, he or she may play any card but will never win that trick if no marriages have been declared.

If a marriage has been declared, either in the current trick or in a previous trick, the suit of that marriage becomes the trump suit for the rest of the round. This means that if a player cannot follow suit, he or she may play a card of the trump suit and will still win that round. For example: A marriage of hearts has been declared and player one leads into the trick with an ace of diamonds. Player two cannot follow suit and instead plays a nine of hearts. Player three also cannot follow suit and plays a ten of hearts. Player three will win this trick since he has the highest trump suited card.

2.5.6 Scoring

The winner of the previous trick leads the next trick. This is done until all cards are played meaning a total of eight tricks will be played. After this, the players count the amount of points they got from the tricks they won plus any marriages they declared themselves. The point totals are rounded to the nearest ten with a five being round upwards. These point totals are then added to a total point count which carries over between played rounds with the exception of the highest bidder of that round. For this player, the total amount bid must be made before rounding to the nearest ten. This means that 108 is not enough to make a bid of 110. If the player succeeds in making the bid, only the bid amount is added to the total amount for that player. If the player hasn't made his bid, the bid amount is subtracted from that players total. (This may seem unfair however it is to be remembered that this player leads the first trick and can thus decide a large portion of the game if he or she plays well).

The objective of the game is to get a thousand points or more. If a player has a total of 900 points or more but still less than 1000, he or she will be "stuck in the barrel". This means that this player may only score points if he or she is the highest bidder and makes the minimum bid of a hundred thus bringing the players total over a thousand. If the player is the highest bidder but he or she does not make the bid, the bid amount is subtracted bringing the total under 900 and removing that player from being stuck in the "barrel". If a player is stuck in the "barrel" and is not the highest bidder, that player cannot score any points.

2.6 Characteristics of Tysiąc

There are many card games of imperfect information all with different size search spaces. The games that play with a full deck will have larger spaces than the ones with a smaller deck. Since Tysiąc is only played with the cards 9 to ace, the search space will be considerably smaller than that of a game with a full deck. When calculating the total number of different possible hands we have to take into consideration that although the same cards hold the same points, they could be used in different ways in respect to the creation of marriages and winning hands by the trump suit. The order in which cards are held does not matter. Due to this we have to consider all possible hands disregarding order after the prikup which is given by the following calculation:

$$\frac{(24 \times 23 \times 22 \times ... \times 19 \times 18 \times 17)}{8!}$$

24 factorial until 17 equals 24!/16!. This creates the final solution of:

$$\frac{24!}{(16! \times 8!)} = 735471$$

possible hands.
The 24 factorial is because there are 24 cards in the deck with the first card having the chance to be any 24 of those cards, the second card any of the 23 remaining cards etc. This is down to 17 because a person receives 8 cards. This is also why the equation has to be divided by 8 factorial since the order in which the cards are held does not matter. This search space is not too large for computers to search through.

3 Design

Within this chapter the design of the computer program will be explained focusing on certain design choices and the functions used in the logic behind playing the game.

3.1 Logic vs Trees

The design of a program that plays card games can be done in many ways as discussed in Section 3. As stated, this computer program will be created by following logical rules and imitating human play as opposed to using large search trees. As stated in Section 2.4, this decision was made based on the level of play the program would need to be able to play at. An example on how this program decides a card can be made clearly apparent by an example game state. A simple example of this state would be the following:

Consider a current state of the game, with every card being visible for simplicity sake:

Player 1:
$\diamondsuit 9$ $\clubsuit 10$ $\spadesuit A$ $\clubsuit H$ $\diamondsuit A$ $\spadesuit 9$ $\diamondsuit V$ $\diamondsuit B$

Player 2:
$\diamondsuit 10$ $\spadesuit H$ $\spadesuit B$ $\spadesuit V$ $\heartsuit A$ $\clubsuit V$ $\clubsuit 9$ $\heartsuit V$

Player 3:
$\heartsuit 9$ $\heartsuit B$ $\spadesuit 10$ $\heartsuit H$ $\heartsuit 10$ $\clubsuit A$ $\diamondsuit H$ $\clubsuit B$

In this example, the game is starting with Player 1 leading into the first trick. Player 1 will always lead with either one of the aces and win that trick since an ace is highest and no marriages have been declared. Furthermore, if Player 1 opens with the ace of diamonds, Player 2 will have to play the ten of diamonds and Player 3 will have to play the king of diamonds since you need to follow suit. This in turn means that the other diamond cards Player 1 holds, the nine, queen and jack, will always win since the other players have no diamonds left and no marriages have been declared. This would be an ideal case however, bids can be made fairly easy when winning just a few tricks and declaring a marriage. Due to this, the deliberate decision was taken not to use trees but see if this type of logic is enough to beat beginners.

3.2 Class design

In this Section the design of the C++ class and the functions of this class will be explained

3.2.1 Functions

The computer program was made in C++ using a class structure. This allowed for the creation of simple functions which are used during the game. In Figure 3 a call graph of the functions can be found with subsequent explanation for the functions. It has to be noted that the classes for the experiments against agents versus the program that plays against humans differs with three functions outlined in red. This is because the program that plays against humans requires human input for the cards and the experiments against agents are autonomous.

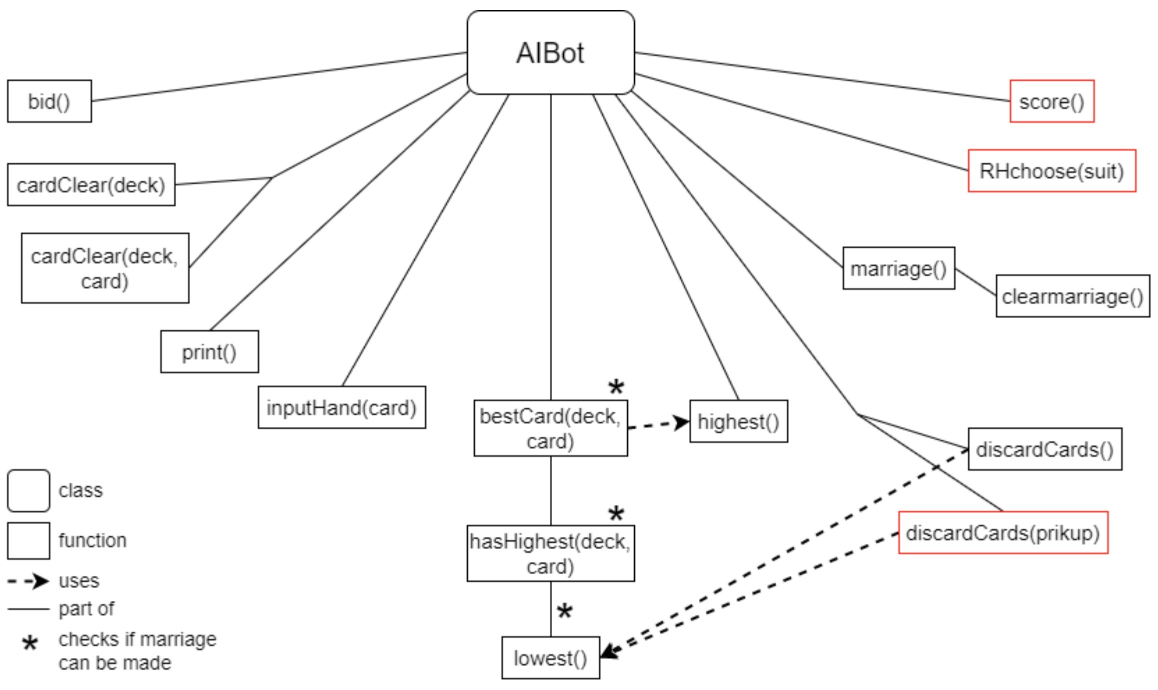

Figure 3: Function call graph.

In order to understand the logic implementation the different functions displayed in Figure 3 will be briefly explained here:

- `bid()`: The bid functions creates a maximum bid that the computer program may bid in order to become the highest bidder. This is based of the number of marriages currently held plus an average score of 30. This average score is further elaborated upon in Chapter 4
- `cardClear(deck)`: Along with a card deck, the computer program also keeps an internal card deck in which the program tracks the unplayed cards. By doing this, the program can essentially count cards and always knows whether a card is the highest of the unplayed cards. This function removes the cards currently held by the program from the unplayed deck.
- `cardClear(deck, card)`: This function does the same as the function with which it shares it's name apart from clearing the specified card from the unplayed deck instead of a whole hand.
- `print()`: This function outputs the entire held hand and the internal tracking of the number of marriages.
- `inputHand(card)`: This function is mainly used in the version that plays against humans and is needed to manually input cards into the programs held hand since there is no way for the computer to "see" the cards on the table.

- `highest()`: This function returns the highest card from the held hand sorted by value.

- `bestCard(deck, card)`: The best card function is one of the main functions that checks whether a certain card is the best card of the unplayed deck. It does this by comparing the card to the internal deck of the program which then returns whether the card is the highest from that suit. This function also checks whether any marriages should be declared. The earlier stated highest function is used to initially get the card to be checked.

- `hasHighest(deck, card)`: If and only if the best card function fails this function is called. This function checks whether a card in the held hand is still the highest of that suit. This is possible when all higher cards of that suit have already been played. The card may not be the highest card in the held hand which is why the best card function can fail but will still win when played. This function also checks if a marriage can be created.

- `lowest()`: Lowest is the opposite of the highest function and returns the lowest card from the held hand. This function is called only when the has highest function has also failed. This means that currently no held card can guarantee a win of the trick thus playing the lowest card making the lost points minimal. It has to be noted that between the has highest function and the lowest function another check for a possible marriage is done. If a marriage can be created at that point, the trick will be lost but there will be made points due to marriage creation which is more desirable than playing the lowest card.

- `discardCards()`: This function is used to discard cards after the prikup and is used in the version against humans.

- `discardCards(prikup)`: This is the automated version with digital deck of the similarly named function described earlier.

- `marriage()`: The marriage function is used to calculate and save how many marriages or part of marriages are currently held. These statistics are used in making a bid, playing a hand and choosing cards from the prikup.

- `clearmarriage()`: This function clears certain marriage statistics.

- `score()`: Score is responsible for calculating the final score and is only used in the autonomous experiments.

- `RHchoose(suit)`: This function is used by the agents to choose a random card from their hand given a certain suit.

These functions make up the logic used by the program to play the card game.

3.2.2 Agents

In order to test the basic effectiveness of the program, two kinds of agents are created to play large amount of games against. The first type of agent is a “random” agent. The random agents play completely random cards from their hands with the only exception that they have to follow suit if possible since that is one of the rules. The second agents are decent agents. These agents play the highest card of their hand using the highest function and will play a random suited card when not starting a trick. These agents in theory play better than the random agents since there is a bit of logic used. More on this will be discussed in chapter 4

Note that these agents have not been created by an expert in the game of Tysiąc and many more types of agents can be created. These agents have been chosen to create a baseline in order to test the functionality of the computer program. Since human players of beginner level generally do not play incredibly well; the decision to leave it at these two agents has been made.

3.2.3 Testing

The program will be tested by performing numerous experiments using different setups. Multiple rounds will be played against the two different kinds of agents as well as against humans. Since an entire round can take up a significant amount of time in real-life play, the number of rounds played against humans will be significantly less compared to the number of games played against the agents.

The reason this way of testing was chosen was because it gives clear insight in how well the program performs against different adversaries. The expected results will be a decrease in outcomes in favor of our program when playing against better opponents. The expertise of agents, whether this be humans or agents, is difficult to measure for this card game because there is no global ranking. However, the performance will be based solely on the number of games won and the duration of a round. This last element is key since it gives insight in to the duration of a game which in turn shows whether the adversaries of the program are really bad or just unlucky. The difference between bad and unlucky will be made apparent during the experiments in Chapter 4

4 Experiments

In this Section the experiments are explained and analysed. Three different experiments are performed to be able to measure the performance of the program.

4.1 Setup

Firstly, an experiment was conducted in which three very basic agents competer against each other in order to get a baseline bid as discussed in Section 3.2. After this, an experiment is performed in which the type of agents are pitted against each other. This is done in order to confirm whether the "decent" agents indeed play better than the random agents. If this would not be the case, the experiment with the program against the decent agents would be useless in terms of performance measurement. A total of 10.000 games is played against each agent type with each game consisting of any amount of rounds until a player is victorious with more than a thousand points. The games against humans cannot be played as fast and will be fewer as a result of this. It has to be noted that not every human player plays at the same level meaning a conclusion about human level of play will be proportional to the level of the players that are tested against. Another important note is that this research focuses on humans that are at beginner level. The moment a player shows a more intricate understanding of the rules and a better strategy, he or she could be considered a non beginner.

4.2 Expectancy

Based on the design of the program the expected results will vary based on the opponents. For the random agents the expectation is that the program will always win. For the decent agents the expectancy is also a high win percentage and may even also be flawless. Furthermore, the expectation for the agents versus agents experiments is that the decent agents will always win against the random agents. Against human adversaries the expectation is that the computer will have a considerable harder time gaining points. The outcome of a game will most likely depend on the amount of experience a player already has with this game. This is because more experienced players can think further ahead and play with better strategies. Furthermore, the game of Tysiąc remains a game with a lot of randomness in the cards meaning that players could be unlucky and get dealt bad hands resulting in a loss. We expect this to come in to play with human players since these players can actively choose to go against a player using strategy.

4.3 Outcomes

The outcomes of the different experiments will be given and explained in this section.

4.3.1 Bid baseline

The programs played 10.000 games between the three basic agents in which every score of one of these agents is kept. After these games the average score per round is calculated which turned out to be a score of 37. Because this score serves as a bid baseline, it is rounded down to the nearest ten since bids can only be made in increments of ten. This gave the final baseline bid of 30 points.

4.3.2 Agents versus Agents

As discussed, the agent versus agent experiment was ran in order to determine that the decent agents are indeed better. Within this experiment 10.000 games were played with the setup in which Agent 1 is a random agent, Agent 2 is the decent agent and Agent 3 is another random agent. The reason only one agent of the three is a decent agent, is because if two or more were to compete against each other, they would be evenly matched and would never make the minimum bid of 100 thus resulting in no one ever gaining enough points for winning the game. The outcomes of the 10.000 played games can be found in Table 1.

	Wins
Random Agent 1	0
Decent Agent	10.000
Random Agent 2	0

Table 1: Agent versus agent outcomes. Decent agent won all games and is better than random agents.

It is clear that the decent agents indeed play better than the random agents and thus can be used to asses our computer program.

The variation in game sizes is remarkable with the total round count being able to go up into the thousands. The distribution of the 10.000 games in round sizes can be found in Figure 4.

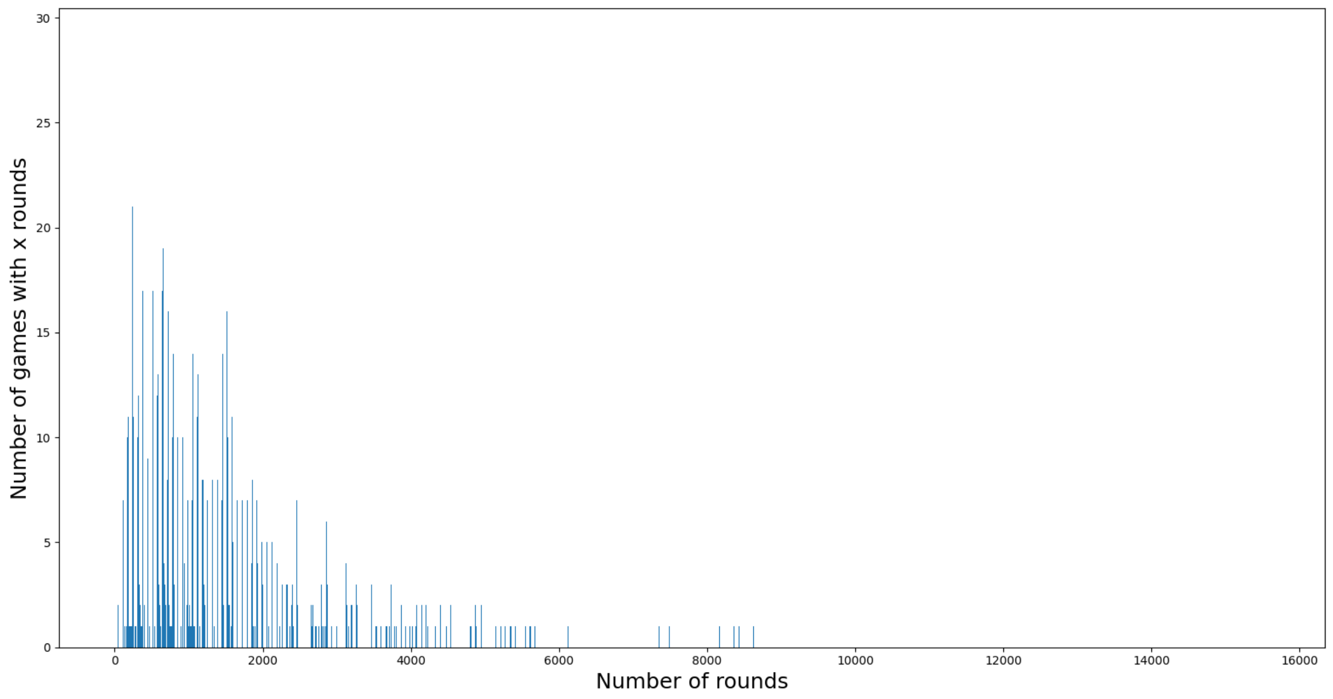

Figure 4: Agent versus agent round distribution. We see that most of the games have a relatively short number of rounds.

The majority of games were finished in under a couple hundred rounds which is still less than desirable in a real match however these agents weren't made to be experts. In order to get a better understanding of the performance, each game is written to a file. This allows us to visualize what rounds the games ended most often in and then calculate a minimum, maximum and mean of those games. The most ideal situation would be that the game ends as quickly as possible. This means that the higher the amount of games ending in low amount of rounds, the better the performance. The most of the 10.000 games finished in 681 rounds which is the reason why this number is chosen to graph the minimum, mean and maximum scores of the agents over the rounds. This choice was made deliberately because graphing all the games in one would not be readable and choosing a round number in which few games end would not be presentable of the performance. The results of the 681 round games can be found in Figure 5.

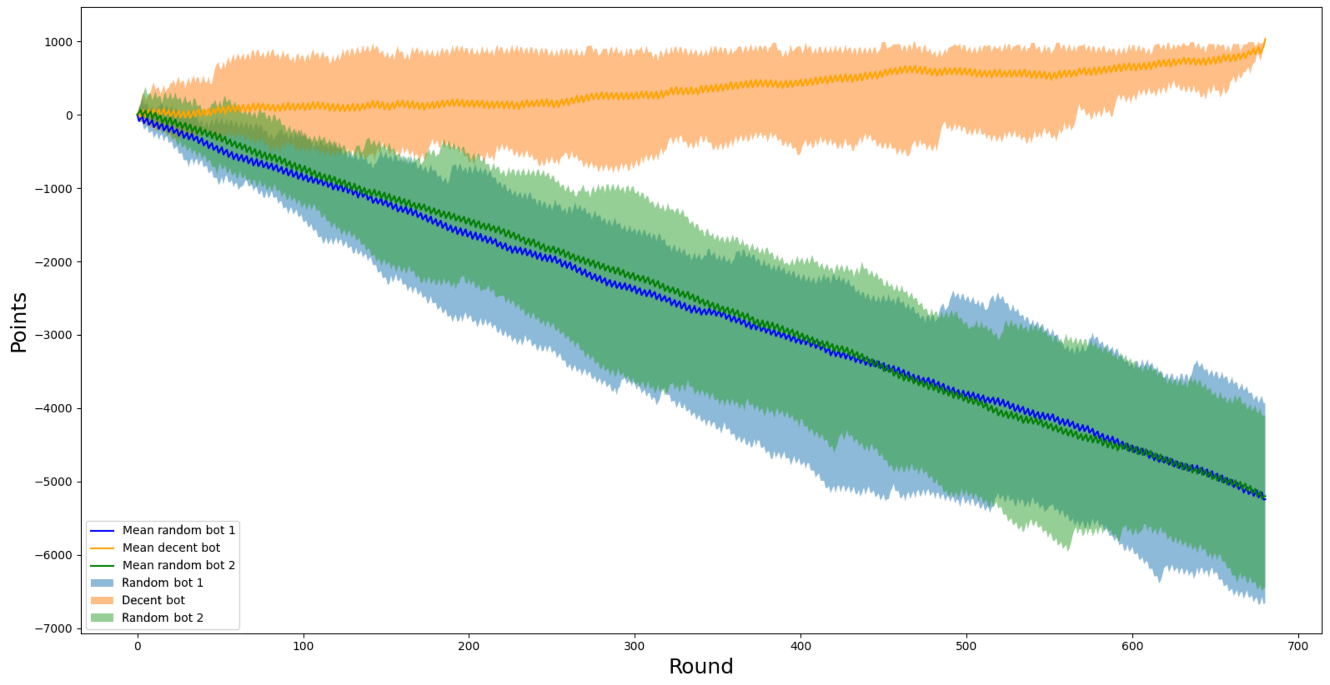

Figure 5: Agent versus agent, 681 rounds. We see that the orange of the decent agent slowly rises to the win while the other two agents lose more over the rounds.

From Figure 5 it becomes clear that the random agents perform increasingly bad over the rounds and the decent agent slowly but surely comes closer to the 10.000 points. It has to be noted that even if most games against agents were finished in such high round numbers, the decent agent is capable of finishing earlier. This mainly has to do with the cards the agents get dealt. In Figure 6 the lowest round number can be found in which the decent agent won in a mere 26 rounds.

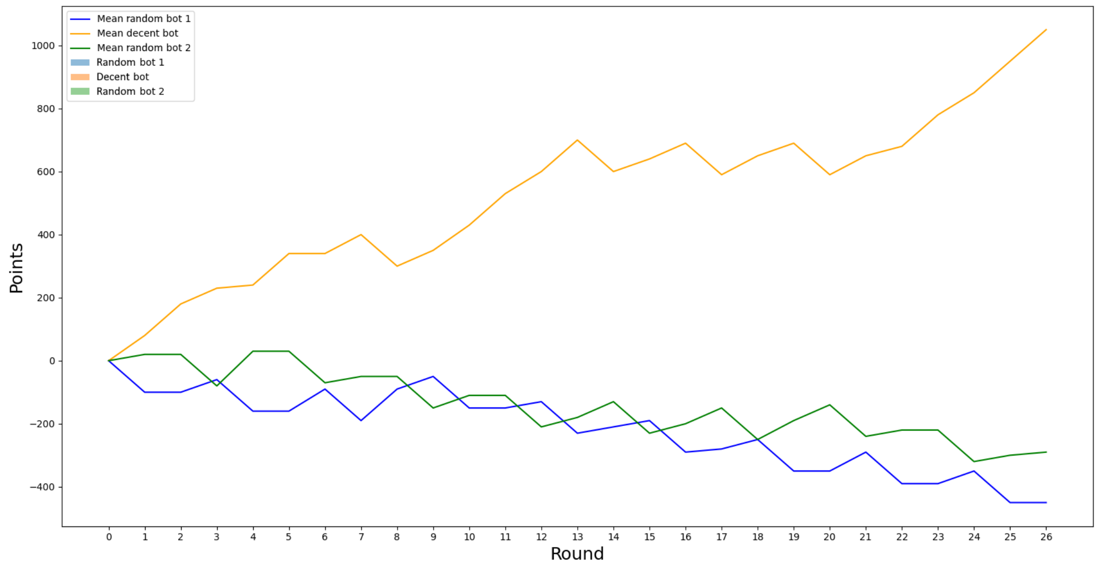

Figure 6: Agent versus agent. The decent agent won after 26 rounds while also constantly leading.

From these experiments we can conclusively say that the decent agents are in fact better than the random agents.

4.3.3 Random agents

The outcomes of the games against the two random agents can be found in Table 2

	Wins
Computer program	10.000
Random agent 1	0
Random agent 2	0

Table 2: Random agent outcomes. Computer won all games and performs better than random agents.

As expected the computer program won all of the played games against the random agents. In Figure 7 the distribution of the games is shown with 16 being the most finished round number.

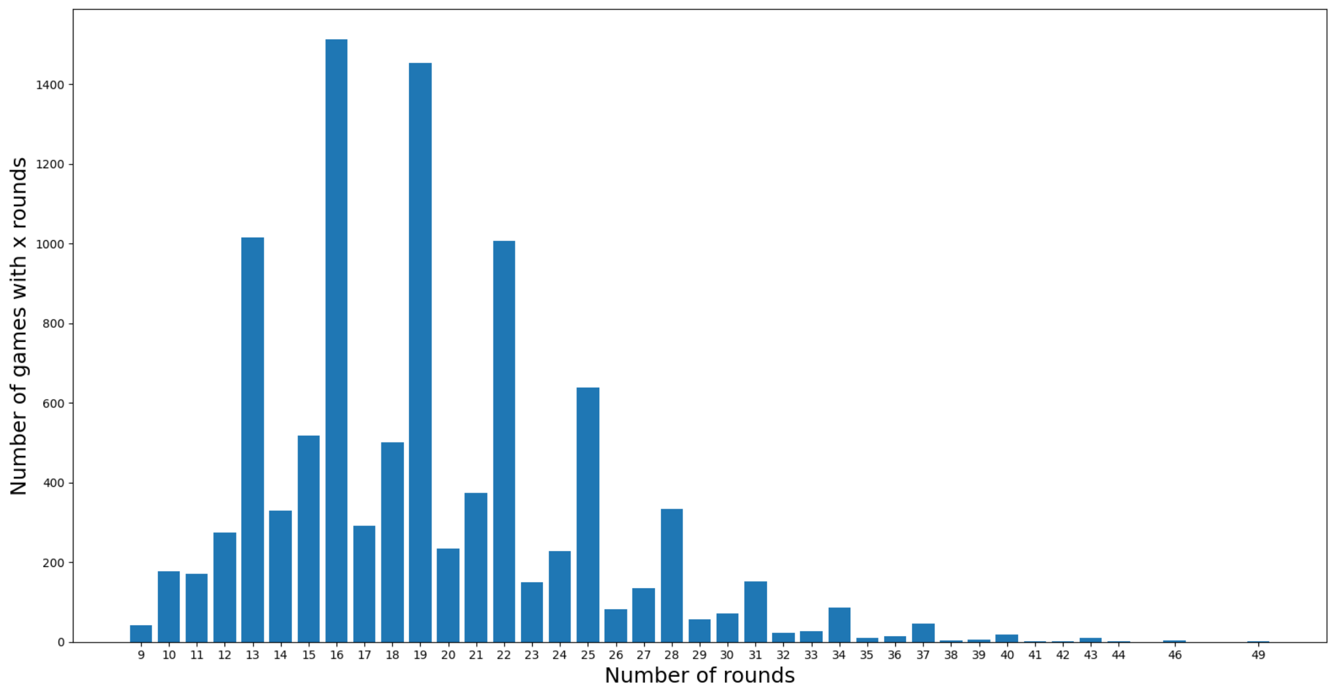

Figure 7: Random agent round distribution. We see that most the games ended in 16 rounds and that games mostly ended under 30 rounds.

Once again, due to readability, this round number will be used to graph the minimum, maximum and mean of the computer program and the agents. This was a conscious decision as will be demonstrated by graphing the highest games. These games ended in 49 rounds and are not a good representation of the minimum, maximum and mean. This will be elaborated upon further following said graph.

An interesting fact to notice is the distinctive- distribution of number of rounds corresponding with how many games ended in those number of rounds. The bulk of the games end in rounds that are 3 rounds apart starting from 11 all the way up to the end when looking at the previous 2 round numbers. For example: More rounds were finished in 14 rounds than in 13 and 12. And more rounds finished in 23 round than in 22 and 21. This can be explained due to the fact the computer, who was the sole winner of all games, needs to be the highest bidder in order to actually get more than 1.000 points and win the game. In the rounds where the computer is not the highest bidder either by not bidding or not being the dealer, the computer won't be able to get out of the barrel and finish the game.

As stated previously, we also want an insight on the actual course of a game. This is because a player might not be as bad as may seem. If this were the case, the player in question could play really good until a certain point after which it stops gaining points thus never winning. On the outside this player would seem rather bad since he or she never wins. However, the argument could be made that the starting rounds of said player are above average since he or she managed to gain points making the player average or above average at best instead of bad. The graph plotting all the games which ended in 16 rounds can be found in Figure 8.

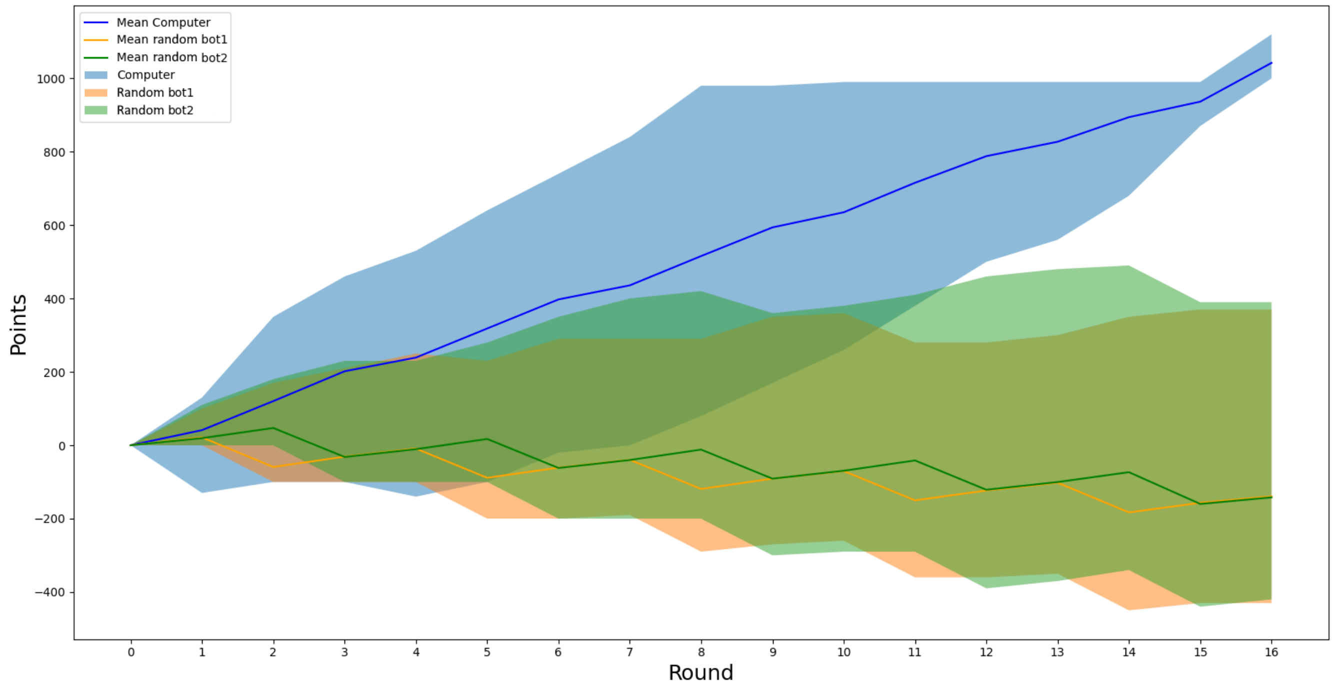

Figure 8: Random agents versus computer program. The computer won in 16 rounds with the agents never gaining more than 400 points.

In Figure 8 it is clear that even the computer does not always play the best. If we look at the (translucent) blue background of the computer mean, we can see that the minimum (the under border of the blue area) reaches around -180 at some point. This means that there was a game in which the computer lost quite a few points in the beginning but eventually managed to get back up. The staircase-like pattern of the agents can be contributed to the bidding system. As can be read in Section 2.5, the rules state that if no bid is made the dealer must take a minimum bid of 100. Since the dealer changes clockwise each round, the agents also have to take bids of 100. Since the agents are rather bad, the 100 is almost never made. This, along with the spare points made when an agent is not the dealer and thus not the highest bidder with no negative points awarded as result, creates the staircase-like pattern.

The overall trend of our computer program is to go further upwards which is most desirable since getting further into the negative points also means taking more time to get out of such a position. As described earlier, the highest games will be plotted to demonstrate the need for choosing a round number with many games.

In Figure 9 the graph for the highest finished game is given. Two of the played 10.000 games were finished in 49 rounds with the computer program taking the win in both. If we look at the minimum and maximum areas it becomes clear why such a game is not presentable for a good analysis. Since only two games were finished in 49 rounds, the maximum and minimum areas correspond to one of those games respectively. This means the blue area of the computer is really pointed and does not represent a nice distribution. It is interesting to see why such a game would take a long time as the maximum of one of the two games managed to get quite a high score at first. After this the computer was most likely dealt bad hands and gained quite a bid of minus points.

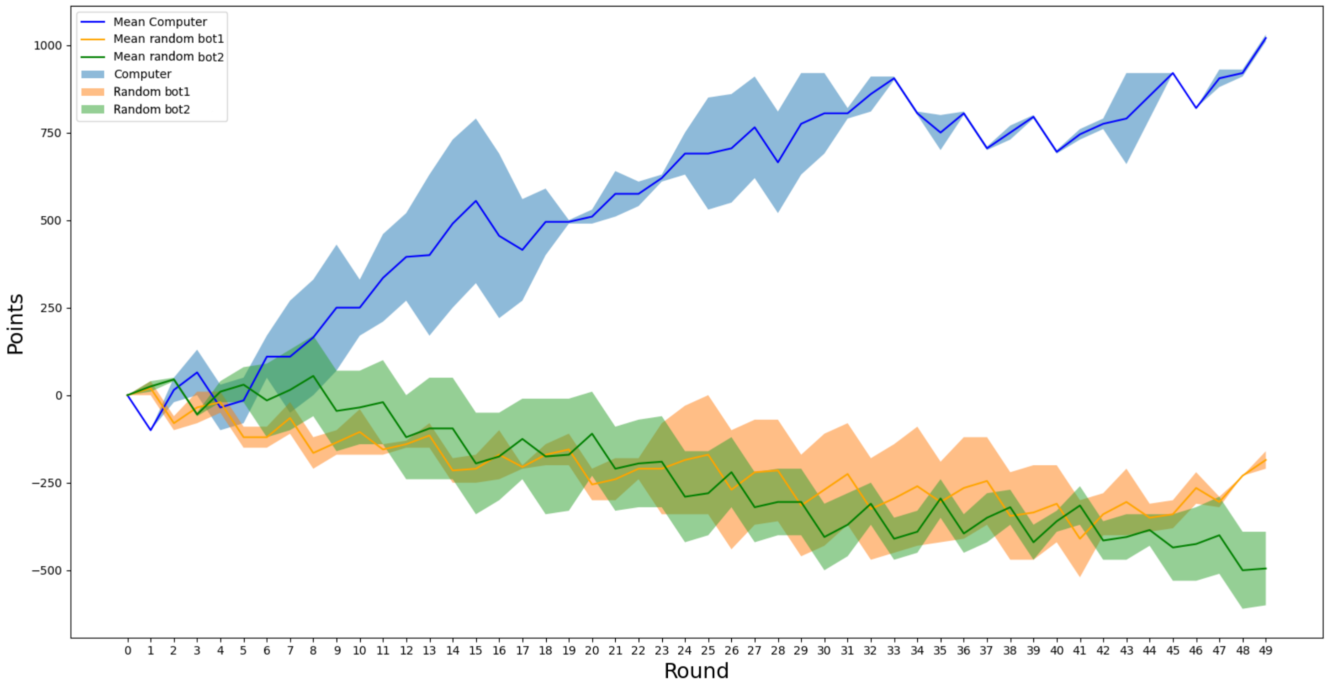

Figure 9: Random agents versus computer program, 49 rounds. We can see that the graph is very spikey and not a good representation of the mean over time.

From these experiment we can conclusively say that the computer program is indeed better than the random agents.

4.3.4 Decent agents

For the decent agents the same approach as the random agents was used. Again, a total of 10.000 games was played with two decent agents and the computer program. The outcome of these games can be found in Table 3.

	Wins
Computer program	10.000
Decent agent 1	0
Decent agent 2	0

Table 3: Decent agent outcomes. Computer program won all games and is better than the decent agents

The computer program also won all 10.000 games in this case. This is a slight step up from the baseline of the random agents as these agents were better than the random agents as demonstrated in Section 4.3.2.

An interesting fact to observe is the number of games ending in a higher round number. The distribution of games ending in a certain number of rounds can be found in Figure 10 with 19 being the most frequent ending round.

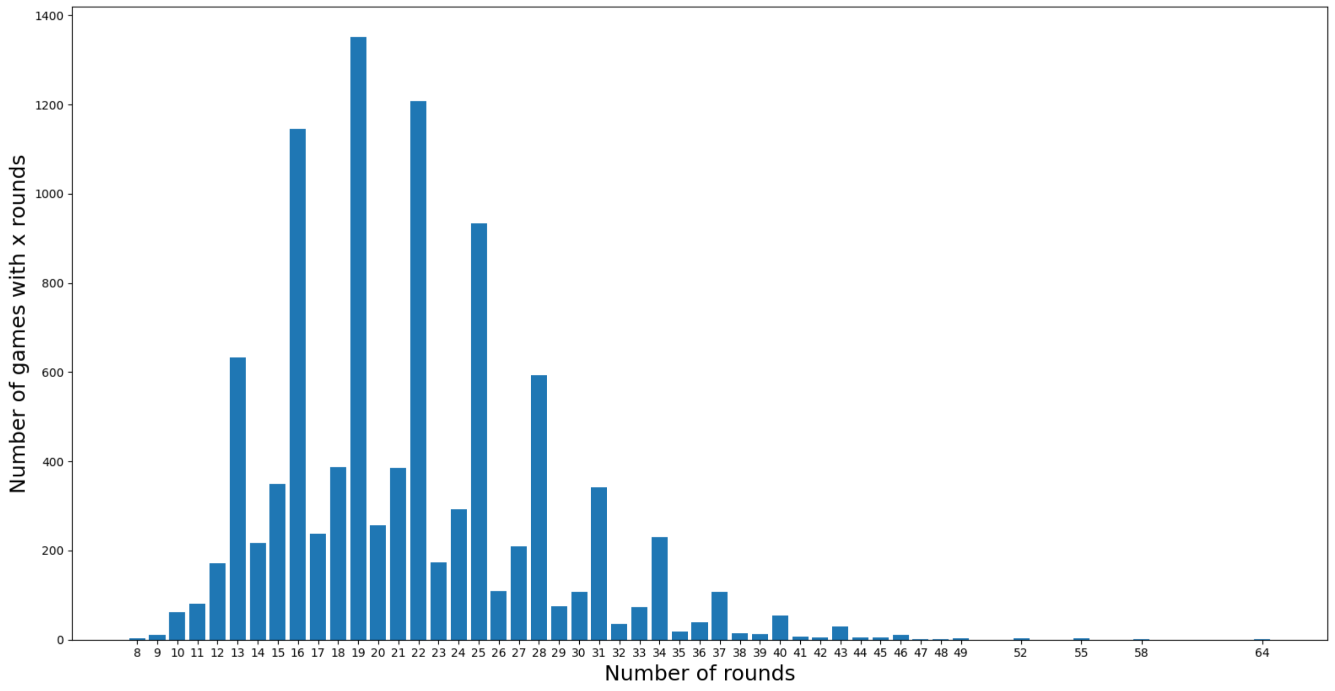

Figure 10: Decent agent round distribution. We see that most the games ended in 19 rounds and that games mostly ended under 37 rounds.

As can be seen in Figure 7, the most frequent ending round was 16 with the highest being 49. In this case 19 is the most frequent with the highest round being 64. This shows that the computer struggled a bit more against the decent agents and beat them in an average longer time than the random agents. This further proves that the decent agents are better than the random agents.

Since 19 rounds was the most frequent this round will again be used to plot the maximum, mean and minimum over all rounds that ended in 19 of the 10.000 played games. This graph can be found in Figure 11.

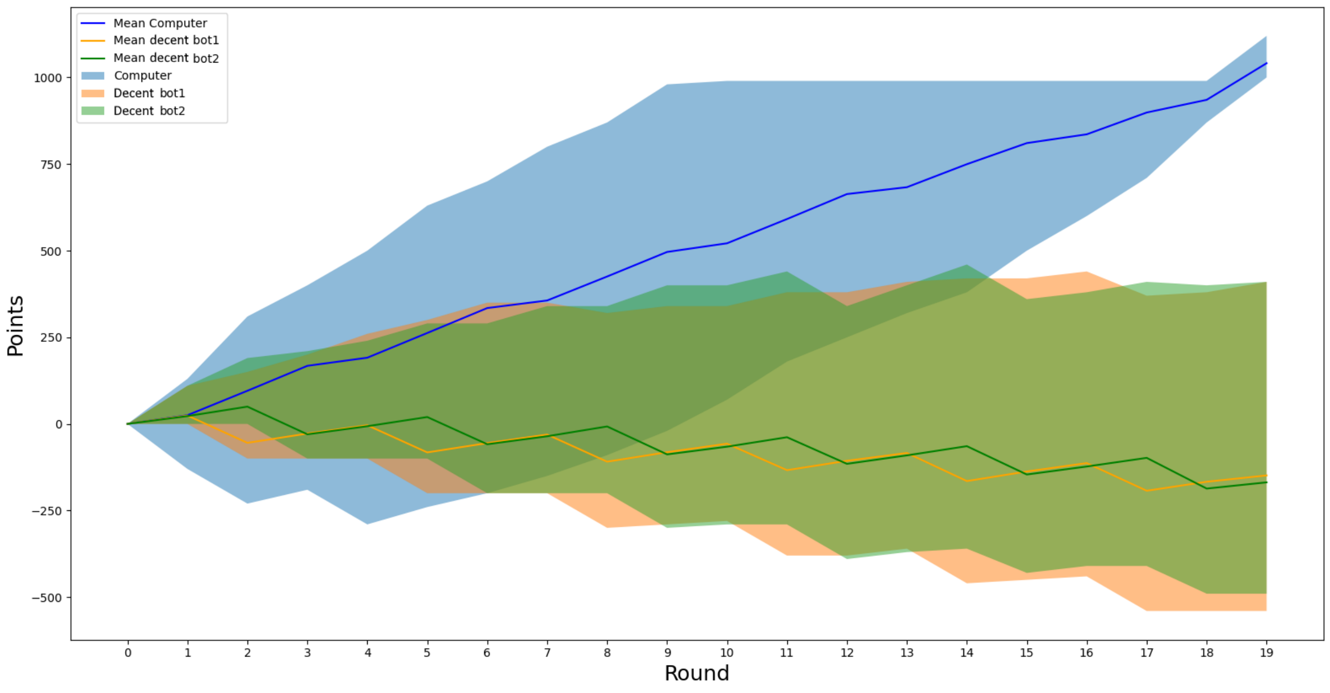

Figure 11: Decent agents versus computer program. The computer won in 19 rounds with the agents not gaining more than 450 points.

An interesting point to notice is the visualisation of the "barrel" as discussed in Section 2.5. If a player is stuck in the "barrel" he or she can only score points if they are the highest bidder. Because the computer plays quite conservative and will only bid if the chance of making such a bid is high, the computer can be stuck within this "barrel" for a long time. This is illustrated by the maximum blue area being stuck at a score just below 1.000 which is the flat area. Only when the computer is sure it can win it bids.

The decent agents also did not pose much of a threat to the computer with the computer winning all of the games. From this it because evident that the computer is also better than the decent agents.

4.3.5 Humans

The experiments against humans are somewhat different than the ones against the agents. Because multiple humans have been played against, one cannot simply take an average of all of them. This would create a situation in which all humans are treated as if they played the same with the same strategy which is not the case. A total of six games until completion have been played against six different persons. These persons shall be referred to as Player 1 to 6. The outcomes along with how many games each player has played can be found in Table 4.

	Games won	Games lost	Games played
Computer	5	1	6
Player 1	0	1	1
Player 2	0	1	1
Player 3	1	2	3
Player 4	0	4	4
Player 5	0	2	2
Player 6	0	1	1

Table 4: Computer versus humans outcomes. The computer won 5 games and lost 1 game.

Of the six games that have been played five have been won. This is a positive outcome but has some interesting cases. The graph of all the games against human has been plotted in Figure 12.

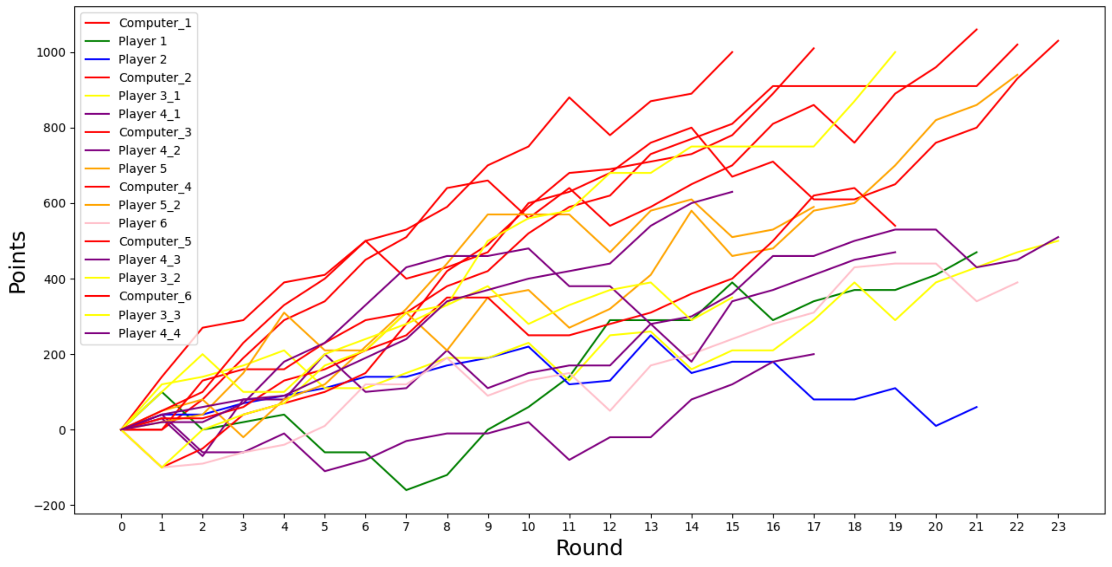

Figure 12: All played games of computer versus humans. We can see that the results of the humans vary greatly with many ending in different rounds and having different ending scores ranging from 100 to over 1.000.

In Figure 12 each color represents a player. If a player has played multiple games this person is represented by the name Player followed by the player number and the game number they played. This means that Player 4_2 is the second game of Player 4. Note that two does not refer to the order in which the games were played but instead simply a second of all the games this player has played. The overall trend of the computer (the red line) is little to no minus points and a steady incline. This is where the strength of the program comes through. Beginner players often play hesitant and cannot quite estimate how much they should bid. This results in many minus points. The computer may not win the fastest but an almost constant incline means it will finish first most of the times.

The one game where the computer lost is an interesting one. The graph of this game has been plotted in Figure 13. The first point to take notice of is the very slow incline in the points of the computer. This is by far the slowest growing line the program has shown over the games and can be explained by the fact the computer simply got rather unlucky. The conservative nature of the bidding of the program often resulted in not winning the prikup which left the computer with the bad cards.

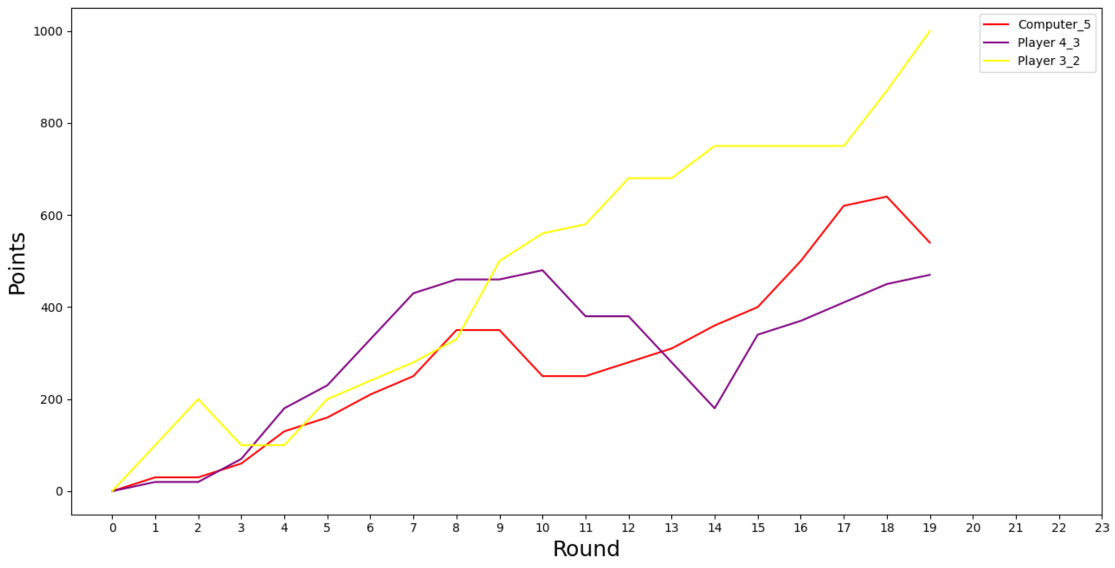

Figure 13: Computer versus Player 3 and 4. The computer had a slow increase of points and Player 3 a very fast increase showing their understanding of the game.

Another noticeable fact is the very aggressive play of Player 3 which resulted in the win. This aggressive play is apparent by the steep incline of the yellow line. This player bid the absolute most times and showed an in depth understanding of the strategies of the game by, for example, counting the cards. This is further backed up by the improvement Player 3 showed over the different games this person played. All the games this player played can be found in Figure 14.

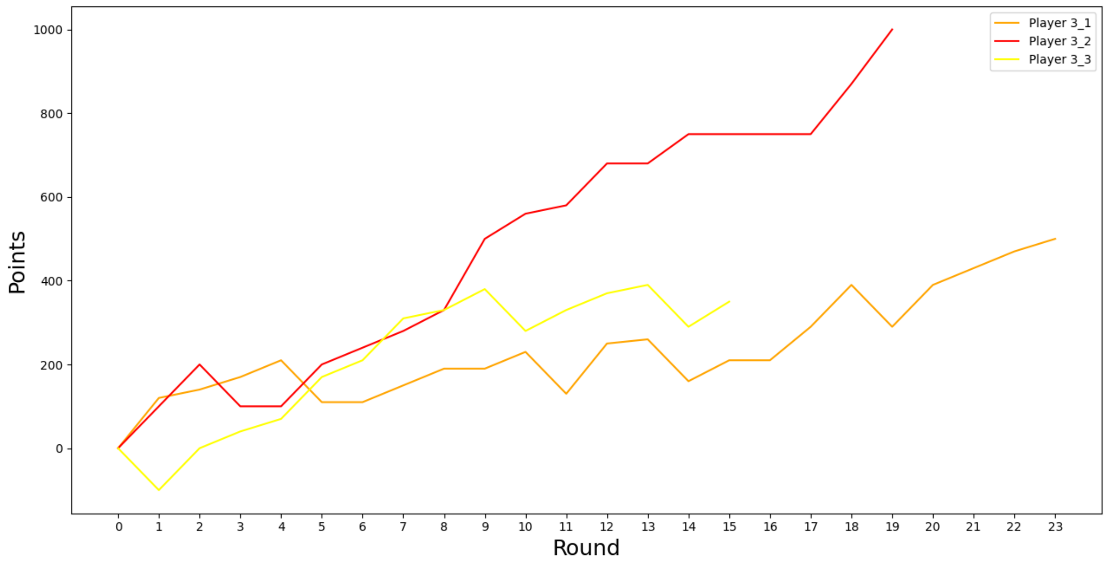

Figure 14: Improvement of Player 3 over games. We can see that the scores go up faster each time showing the improvement over different games.

From Figure 14 it is clear that apart from the start, this player gained increasingly more points each game. This can be found by looking how many rounds it took to get to a certain amount of points. Due to these reasons, the argument can be made that Player 3 understood the game so well that this person should no longer be considered a beginner player. Only looking at point graphs is not enough to declare a player "non-beginner". For this the mentioned real life observation of the intricate knowledge of strategy should also be made.

This "learning" of the game rules and strategies is further backed up by the improvement Player 4 made over the four different games. A similar graph to Figure 14 has been plotted in Figure 15. Other than the stroke of bad luck in cards with the mandatory bid of 100, the games steadily improve the more games this player has played. Again, the time in rounds until a certain score has been reached goes down.

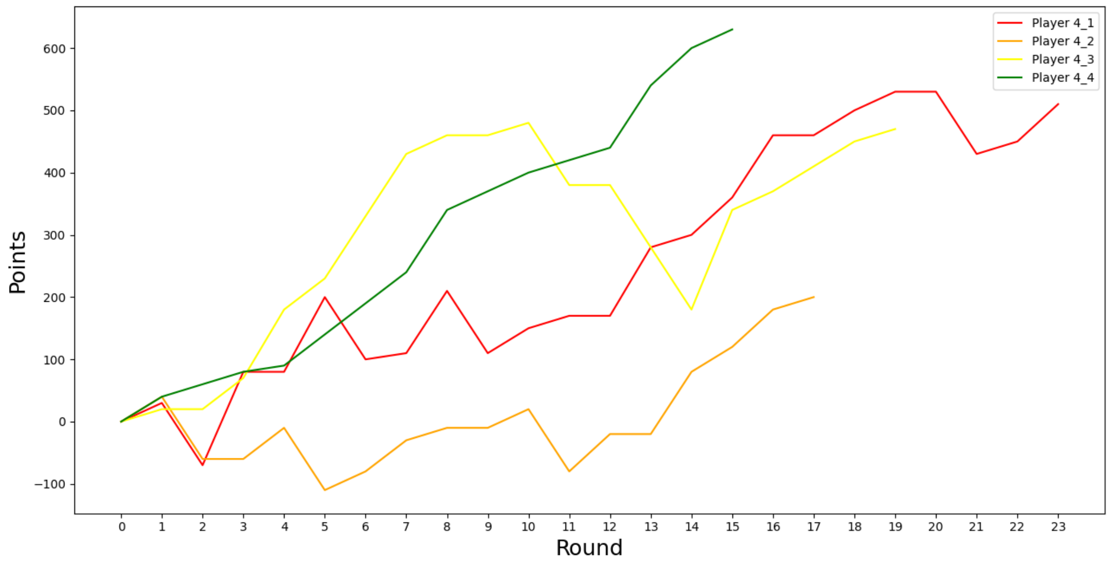

Figure 15: Improvement of Player 4 over games. We can see that the scores go up faster each time showing the improvement over different games.

From these outcomes we can state that this knowledge-based program is good enough to beat beginner players. When a player gains more intricate knowledge of Tysiąc and goes to non beginner, this program will fail more often and would need some adjustment.

4.4 Retrospect

In retrospect, my expectations of the program were fairly accurate. The program was able to beat the agents and most of the humans. The expectancy of the computer having a harder time gaining points was not necessarily correct. The hunch that human players will actively seek to oppose a player that is far ahead was also something that came in to play with humans. One unexpected outcome was the ability of some players to quickly adapt to the play style of Tysiąc and the skill level rising quickly.

5 Conclusion

The conclusion of the thesis is stated here along with the limitations of the program and possible future work.

5.1 Conclusion

Within this thesis the main research question of "How can you construct a knowledge-based program for the card game Tysiąc that is able to beat beginner human players?" was answered by creating a program that is able to beat beginner players. In order to answer the main question three sub-questions were proposed and answered. These sub-questions were:

1. What is known about methods of letting a computer program play a card game?

2. Based on the known methods, how can we create a program that plays the card game Tysiąc?

3. How well does the program perform against different opponents?

There are distinct methods of letting a computer program play a card game. Three of the main methods discussed are by creating decision trees, making a neural network with a learning AI and emulating human play. This third one is the one used to construct the program. A small glimpse into the history of computers playing games is given along with an explanation of what these methods do. The decision to create a program that is based on rules and human play emulation was based off previous programs using similar techniques and gaining high skill levels. This clearly shows the capability of such programs.

Furthermore, it is rather important to know how to design such a program. There are numerous rules and logistics this program follows which need to be understood in order to judge the agent on its effectiveness. The computer uses an internal deck which creates the ability to perfectly count cards. This along with some rules created the design for the computer program and has proven to be effective against different adversaries.

In order to test how well the program holds up against different opponents, multiple experiments were conducted. Different type of agents were created to test the baseline of the program and essentially establish whether the program behaves in the right way. The outcomes of the random agents were successful with the computer winning all of the played games by a considerable margin. The decent agents were also no match with the computer again winning all matches. During the latter the computer program did show a bit more effort with the matches usually taking longer. The two agents that competed against each other also truly showed that one agent was indeed better than the other with the decent agents beating the random agents each time.

The human experiments were also successful with the computer winning all but one game. This loss was shown to be mainly due to bad luck, withheld bidding and the aggressive play of a non-beginner player. During these experiments it became apparent that human players also quickly gain more extensive knowledge in how to play the game and what good and bad strategies are. The overall performance outcome of the computer program was positive and could be a viable option in beating beginner human players.

5.2 Limitations

Due to time constraints, some aspects could not be further expanded upon. One of these aspects is the bidding process. The computer program has a base bid of 30 and adds the highest marriage it can make. While this results in almost always making the bid when the program is the highest bidder, most of the time the program won't be the highest bidder. This means the "prikup" cannot be picked up and bad cards cannot be disposed of. This in turn creates the situation where the program will rarely lose points but won't gain points fast. This is a problem against aggressive players that bid a lot and make those bids most of the time. The bidding process also lacks the context of the held hand for example some hands being really good but with no marriages which in turn makes it so the program won't bid. This causes the program to miss out on points.

5.3 Future work

The limitation of the bidding process can be mitigated by creating a search algorithm. Such an algorithm could use a tree in order to calculate possible outcomes and create a minimum bid based on the worst outcome. Such a program could prove useful when competing against human players based on how far the program could look forwards.

Another option is creating the other method for playing a card game: a learning neural network. This program could play many games against itself thus learning good moves. These programs could then be pitted against each other to determine the best one after which more experiments against humans can be done.

References

[BML93] Jean R.S. Blair, David Mutchler, and Cheng Liu. Games with Imperfect Information. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning, AAAI Press Technical Report FS93-02, Menlo Park CA*, pages 59–67, 1993.

[CHJH02] Murray Campbell, A. Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1–2):57–83, 2002.

[Sam59] Arthur L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[SNT98] Stephen J. Smith, Dana Nau, and Tom Throop. A Big Win for AI Planning. *AI Magazine*, 19(2):93, Jun. 1998.

[vUv02] H.Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1):277–311, 2002.

[Wik22] Wikipedia. Russian Schnapsen — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Russian_Schnapsen`, 2022. [Online; accessed 29 May 2022].

[YT18] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*, volume 2. Springer, 2018.