university of groningen / faculty of science and engineering / artificial intelligence

Playing Rimboe Road Using Minimax and Alpha-Beta Pruning

Bachelor's Project Thesis

Rengert van Dolderen, s4329716, r.j.van.dolderen@student.rug.nl,
Supervisors: Dr. H.A. de Weerd

Abstract: This research explores the application of the minimax algorithm to create AI opponents in the game Rimboe Road. The goal of this study is to investigate whether AI opponents can be created of varying skill levels through the use of this algorithm. An experiment was conducted on a total of 12 participants who played against three AI opponents. There was data collected on the performance of the players against the different AI opponents together with their reported experienced difficulty playing against each opponent. Statistical analysis shows that participants won less often from the minimax agents and considered these agents to be more challenging/difficult. However we find no difference in performance or experience based on the depth of the minimax agent. Overall, this study showcases that the minimax algorithm is a promising algorithm to use to create AI opponents of varying difficulty in the context of the game Rimboe Road.

1 Introduction

Board games have been a key asset in not only the world of leisure and strategic thinking but also a repeating component in the fields of computing science and artificial intelligence (Schaeffer, 2020). Especially the timeless two player games like chess and go have played numerous roles in these fields. The well defined rules in these games make them ideal for computational analysis.

The combination of well defined rules and a complex gameplay create an intriguing challenge for researchers to develop systems that are able to play these games. In 1997, the Deep Blue chess-playing computer managed to beat the former world's chest champion Garry Kasparov (Campbell et al., 2002) showing the potential for algorithms to surpass humans in complex cognitive tasks. Besides highlighting advancements in artificial intelligence and computing science, challenges like these have also shown to be a key component in shaping algorithmic game theory, which is often applied in fields like economics and logistics (Lin et al., 2018). The encountered challenges and solutions from these challenges also often serve as an educational tool within these fields.

This research focuses on a board game created around 30 to 40 years ago by Bart Munting called Rimboe Road. Like board games for which other AI programs have been developed, Rimboe Road consists of well defined rules with a complex game structure. The game contains similar aspects which can be found in chess and noughts and crosses.

In the context of games, there are many algorithms which can be utilized to create AI players. There are both algorithms which require preprocessing and training like using a neural network, while there are also algorithms which only use on-the-fly computation like for example the minimax algorithm. This research requires an AI opponent which is able to play at different skill levels. An algorithm is therefore required which provides control over it's computational strength to the developer. An example of such an algorithm is the minimax algorithm which is an on-the-fly computational algorithm and will therefore be utilized within this research.
This study seeks to address the following research question: Can the application of the minimax algorithm be effectively utilized to create bots of varying difficulty levels in the game Rimboe Road? Answering this question will shed light on the transferable skills from techniques developed in other domains in a new one and will also provide the game Rimboe Road with an Artificial Intelligent player generating more immersive gameplay. This research is guided with the following hypothesis: Utilizing

the minimax algorithm in Rimboe Road will create AI bots of varying difficulty levels, offering a scalable and competitive solution for diverse player skill levels.
This paper will cover the implementation of the game together with the algorithm behind the AI player to create an experimental platform to test the AI's ability to play the game against human players. It also covers the results of the experiments together with comments on the research and possible future research.

2 Method

Rimboe Road is played on a grid which can vary in size creating different playing experiences. The possible board sizes for this game are 4x4, 6x6, 8x8, 10x10 and 12x12. Not only the size of the board can vary but also the number of players. This game is a free-for-all game which can be played with up to 8 players. Within this study, humans will have to play multiple games against AI opponents. This study therefore only focuses on 2 player games on a 6x6 board to keep the games short while still maintaining strategic gameplay.
The game starts of with an empty grid on which each of the players choose a starting position. From here on, the players take turns moving around the playing field. They can move 1 to 3 spaces up/down/right/left. When they leave their position, it becomes unavailable and they then occupy the new location. They are able to jump over unavailable spaces or other players. The playing field used in Rimboe Road does not warp around, this means that you do not end up on the left side of the board if you cross the right border. An example of a game being played between two players on a 6x6 board can be seen in Figure 2.1. In the case of this example, it is blue's turn. The possible positions the blue player can move towards are displayed in light blue.
The premise of the game is to prevent yourself from getting stuck while trying to force the other players to get stuck. Players often accomplish this by either picking a route which allows them to make more moves compared to the other player or try to block the path of their opponent. An example of a blocking move can also be seen in Figure 2.1 where the player in red moved towards the top left

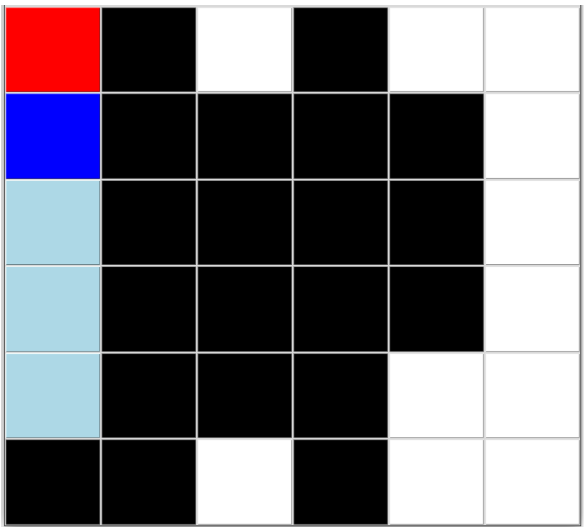


Figure 2.1: Example Match

position causing its opponent player in blue to get trapped on the left side of the board. At this point, the blue player is limited to a total of 3 positions to move towards while the red player can still reach a total of 11 positions. This move by the player in red assures that player the win.
The state of the game is only evaluated when all players completed the turn. In the case of a two player game, this means that if the starting player is unable to move, the other player still has to make a move to complete the turn. If the other player also ends up getting stuck, the game will end in a draw. If only one of the players gets stuck and the turn has come to an end, the game also ends and the stuck player loses. This turn-based rule can be found in other turn-based games to decrease the advantage of the starting player. To further prevent unfair gameplay within this research, we consider a match to consists of two games being played where the players alternate between being the starting player.

2.1 Game Design

For the design of this experiment, a version of Rimboe Road was created in Python to function as a building platform for the rest of the experiment. This reconstructed version of the game consists mainly of two modules: the board and game module. The board module contains the playing board object and processes all operations made on the board, for example updating positions to occupied/unavailable. This module also holds the information on the player positions and available locations on the board for both players to move towards. The game module handles the course of the game. This means handling the turn-based system of the game, operating the AI player's move and handling the user input for the human player. This module inherits the board module. It communicates the player's actions to it and requests information from the board.

Furthermore, a user interface was created for the experiment with the use of the python package tkinter (2022). The tkinter package allowed easier interaction between the experiment and the participant compared to a terminal-based experiments as the participants can click on the screen instead of type out commands. In addition, the use of a user interface allowed the use of colours which both helped visualise the positions to which the participant could move and also distinguish their own position from the one of the opponent. These observations are all based on feedback received from people trying terminal-based pilot versions of the game. A demonstration of the final version of the user interface can be seen in Figure 2.2. The colour coding of the game is as follows:

- White: Available positions
- Black: Unavailable positions
- Red: AI player position
- Dark blue: Human player position
- Light blue: Positions human can move towards

To collect data of the experimental trials and also allow examination of the trials afterwards, the program maintained a textual record of the game board and progress through all matches. When the experiment has been completed, the program generates two text files, one containing the full match history together with the scores and the other containing the content submitted by the participant.

2.2 AI Design

This newly designed Rimboe Road game served as a production hook to allow easy implementation of

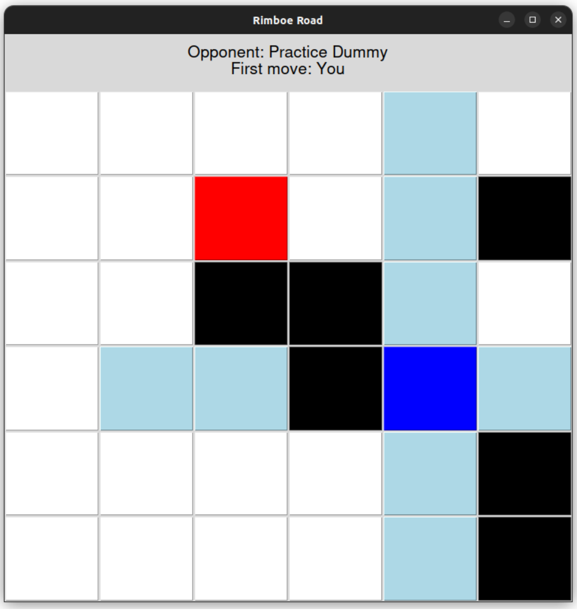


Figure 2.2: Rimboe Road GUI

the artificial players required for this experiment. In the case of this experiment these are the human player class and the AI player class. The human player class was designed to handle user input and translate it onto the board. The AI player class was designed to operate autonomous. It operates with the use of the minimax algorithm (Russell & Norvig, 2010).

The minimax algorithm is a decision-making algorithm used in games with two players. The algorithm's strength can be modified through a single variable which facilitates the creation of AI players of varying skill levels. The algorithm's goal is to choose the most optimal move to make assuming that the opponent also plays optimally. It achieves this goal by creating a tree containing every possible combination of moves which both players can make. Working from top to bottom recursively, the algorithm will decide on what the best move is for the player at each level of the tree. It does this by evaluating the state of the game at the end of each branch by assigning a value to it. A high value indicates that the outcome is favourable towards the current player and a low value towards the opponent. At every node, the maximizing player will choose the move with the highest value and the minimizing player will choose the move with the

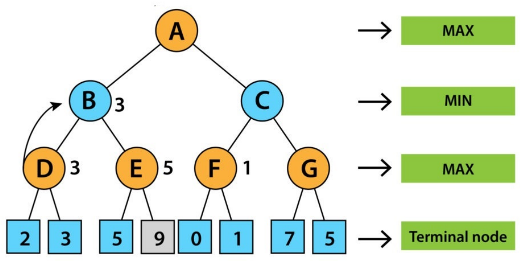


Figure 2.3: Minimax Algorithm Visualised

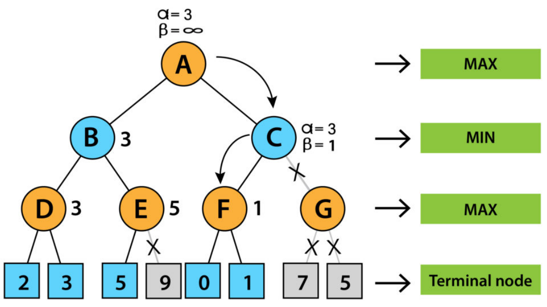


Figure 2.4: Minimax Algorithm with Alpha-Beta Pruning Visualised

lowest value. Each node in the branch will make a choice and receive an evaluated score. The tree's nodes will be filled with values until it reaches the starting node (which is the current player's decision). The algorithm then decides on which move has the highest score and settle on making that move. Figure 2.3 visualises this process. In the case of this project, the evaluation value ranges between -1 and 1, where 1 indicates a win for the minimax player and -1 represents a win for its opponent.

The minimax algorithm, especially in cases where there are still a lot of moves to make or if there are a lot of possible moves per turn, is very computationally intensive. It is therefore possible to limit the algorithm to a certain depth. This depth restricts how far the minimax algorithm is allowed to branch out. Setting the depth to 4 would restrict the algorithm to compute a maximum of 4 ahead of its current state. As an example, the depth of the minimax algorithm visualised in Figure 2.3 has a depth of 3. Together with these two characteristics we arrive at a time complexity of $\mathcal{O}(b^d)$ with `b` for the number of possible moves per position and `d` for the set depth of the algorithm (Russell & Norvig, 2010). As the game progresses and more spaces become unavailable, the algorithm will require less computational time as the number of possible moves per position decreases. Furthermore, more actions will lead to a terminal state and cause the algorithm to stop exploring. This depth value is the variable earlier mentioned with which the strength of the algorithm can be set.

When determining the value to set for the depth limit in a minimax algorithm, various factors must be considered, balancing the trade-off between computational time and the strength of the AI player. When increasing the depth value, the strength of the AI increases but so does the the time for it to compute the best move.

The minimax algorithm in its current state does not allow us to set a high depth value without creating a visual delay. The use of the alpha-beta pruning technique helps to tackle this issue (Russell & Norvig, 2010). This technique is used to reduce the number of states that the minimax algorithm has to evaluate and therefore reduces the computational weight of the algorithm. The technique adds two new values to the algorithm, alpha and beta. These values hold information over the lowest possible score that the maximizing player can achieve and vice versa. These values allow the algorithm to avoid processing nodes that cannot possibly improve the decision of a parent node. Take a look at Figure 2.4 to create a better understanding of this process.

You can see the values of α and β at the C node: 3 for alpha and 1 for beta. The beta value is the value retrieved from the child node (F) and the alpha value is currently the best value for the parent node (A). As this is a minimizing player, it's computed best move will have a rating of 1 or lower. But as the maximizing parent (A) node already found a move with a value of 3, you can already conclude that the C node is not going to affect the parent node's (A) choice. Therefore you can prune the G node together with its child nodes and skip processing that part of the tree.

On average, the use of the alpha-beta pruning technique allows the minimax algorithm to skip processing half of the nodes (Russell & Norvig, 2010). This reduces the time complexity of the minimax algorithm from $\mathcal{O}(b^d)$ down to $\mathcal{O}(b^{d/2})$. With this new time complexity it is possible to set a higher value for the depth without creating a visual delay. With the use of some testing, we arrive at a maximum depth of 6 without creating a visual delay.

The current minimax algorithm together with the alpha-beta pruning technique has another flaw. It currently assesses the game state as 1 (AI win), 0 (Draw / Game not finished yet) and -1 (Human win). Unfortunately, it is impossible to find a terminal state for either players within 6 moves from the starting positions. That is, neither the player can guarantee a win for themselves within 6 turns. Therefore, the algorithm barely affects the move choices at the start of the game as every final node in the tree contains the value 0 because no combination of moves results in a terminal state. To make sure the agents are able to assess whether a move is favourable from the start of the game as well, a heuristic function is introduced. This heuristic function computes a score of the current state of the game even when no terminal state has been reached yet. The minimax tree will, accordingly, be filled with evaluated values instead of zeros and will allow the agent to assess how favourable a situation is. The heuristic function computes a score between -1 and 1 based on the number of possible moves both players have at that current state. This heuristic function is `(AIm - Hm) / (AIm + Hm)`. With `AIm` for the number of possible moves which can be made by the AI player and `Hm` for the number of possible moves which can be made by the Human player. If the computed value is close to 1 it means that the AI player has a lot of possible moves compared to the human player. This works the other way around for a value close to -1. The number of available moves a player has, serves as a predictor for their following turn potential, this function will therefore provide an evaluation which represents how favourable a game state is for the AI player. The addition of the heuristic will not only prevent the AI from making random moves at the start of the game, but it also allows the alpha-beta pruning technique to be useful. If all nodes reached a score of 0, the algorithm would not be able to prune any branches.

2.3 Experiment Design

2.3.1 Participants

The participants in this study were individuals known to the researcher. There were 12 participants in total, 6 of them conducted the experiment in a controlled environment and the other 6 conducted the experiment in an uncontrolled environment. The participation of the experiment was done completely anonymous as the program did not ask the participants to provide any personal information. Not only would this data not be used during the research but excluding the data collection also lowered the barrier for people to participate.

2.3.2 Procedure

The experiment starts with a user consent form, the user is informed that they are at all times allowed to stop the experiment and that participation is voluntary. They are also told that their game data will be kept confidential. The participants hereby make the choice to either participate or not by clicking on the ‘accept’ button. Both the consent form and the game explanation can be seen in Appendix A.1 and Appendix A.2. From here on, the experiment will start.

The first part of the experiment contains the explanation of Rimboe Road. This consists of a written explanation that contains the rules of the game. In case the explanation was not clear enough, the participants get the opportunity to practice against the “practice dummy”, a Rimboe Road bot which makes random moves. They play two practice rounds and are then asked whether they are confident enough to start the rest of the experiment. If they do not feel sufficiently confident, they have the option to play additional practice rounds. The explanation also states that the participant will play against three AI players and play two rounds against each AI player. They are also told that they will be asked to rank the skill level of each AI player afterwards.

When the participant moves on to the next phase, they start their matches against the three AI players. Each match consists of two games, the human makes the first move in the first game and the AI makes the first move in the second game. The first match is always played against the medium opponent which has a depth of 2, the second match is played against the easy AI which makes random moves and finally they play against the hard AI which has a depth of 6. As the group of participants consisted of only 12 participants, it was decided to have a set order for the AI opponents to keep consistency in the testing.

The participants themselves are not shown which difficulty they are playing against, only the names that were assigned to the different AI players. This allowed the AI players to still be distinguishable in the data while keeping the difficulty hidden from the participants. The names of the AI players were: Hidde (Easy), Max (Medium), Gabriel (Hard). Furthermore, we do not want the participants to see a visual difference in computational time when playing against the different AI players. This would allow the participant to get an idea of which AI might be stronger than the other. With that in mind, we set the depth of Gabriel to 6, the highest depth value which does not create a visual delay. The depth of Max is set to 2 and the depth of Hidde is set to 0. Hidde therefore makes random moves. After playing each AI player and submitting experienced difficulty level of each player the experiment has been completed.

3 Results

An ANOVA test was performed to compare win rate across multiple difficulty levels of AI players in the game. The AI player type significantly influenced the performance of the participants: (F(2, 69) = 11.28, p = 5.76e-05). The TukeyHSD test was also performed on the data to further test the the specific performance differences among the AI bot difficulty levels. This test revealed a significant difference in win rate between Hidde and Gabriel (p = 0.0000327) and also between Hidde and Max (p = 0.0277597). However, there was no significant difference found between Max and Gabriel (p = 0.0958165). The performance data can be seen in Figure 3.1. This figure shows that when the depth of algorithm increases, the agent wins more often and loses less often.

The participants were also asked to rate the difficulty of each AI player which had to be reported as a value from 1 to 10 (see Figure 3.2). Also on this data, the ANOVA test was used to compare mean difficulty rating across multiple difficulty levels of AI players. In this case, the AI player type also significantly influence the experienced difficulty by the participants while playing: (F(2, 33) = 9.489, p = 0.000555). The TukeyHSD test was also performed on this data and again displayed a significant difference between Hidde and Gabriel (p = 0.0004135) and between Hidde and Max (p = 0.0240371) while no significant difference was detected between Max and Gabriel (p = 0.2927292). The reported ratings plot can be seen in Figure 3.2.

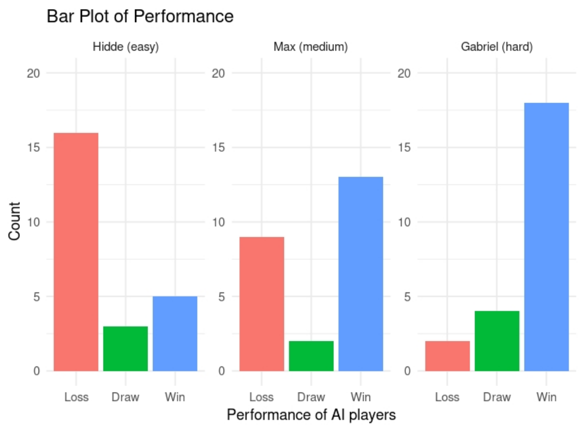


Figure 3.1: Performance of AI players playing against 12 human opponents. Each agent played against each participant twice.

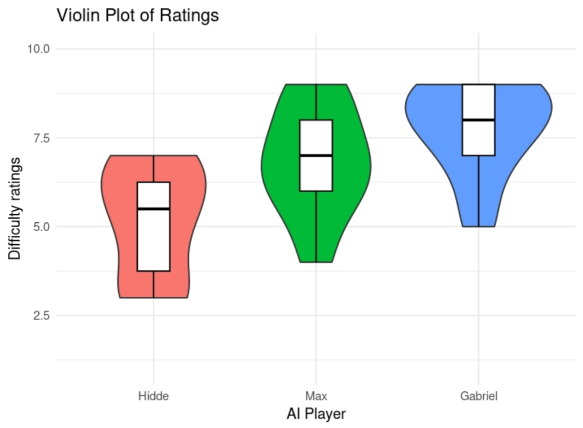


Figure 3.2: Distribution of difficulty ratings participants assigned to the three AI players.

The hypothesis at the start of this experiment stated: Utilizing the minimax algorithm in Rimboe Road will create AI bots of varying difficulty levels, offering a scalable and competitive solution for diverse player skill levels. This test helped determine whether there was a significant performance difference among the levels. The results from the ANOVA tests yielded a significant effect from the AI player type on both the performance and the reported difficulty experienced by the participants. The TukeyHSD tests yielded a significant difference in means for both the performance and the reported difficulty rating between Hidde and Gabriel and also between Hidde and Max. There was however no significant difference found between Max and Gabriel. These current results suggest that the minimax algorithm is a promising algorithm to use to create AI opponents of varying difficulty in the context of the game Rimboe Road.

4 Discussion

The aim of this study was to investigate the efficacy of applying the minimax algorithm to develop AI bots of varying difficulty levels in the game Rimboe Road. To address this objective, the research question guiding this investigation was: Can the application of the minimax algorithm be effectively utilized to create bots of varying difficulty levels in the game Rimboe Road? Hypothesised was that utilizing the minimax algorithm in Rimboe Road will create AI bots of varying difficulty levels, offering a scalable and competitive solution for diverse player skill levels. This was tested through the use of a newly created game platform in Python. The experiment was conducted on 12 participants who played against three different bots. Data was collected on the performance and experienced difficulty of the players. This dataset revealed a significant correlation between the specific AI bot in play and both performance metrics and perceived difficulty levels. Together with the test performed between the groups it was observed that the minimax algorithm resulted in more challenging players than random agents. While there was no significant effect of depth, trends indicate high depths may be more challenging, but this should be researched more extensively. Overall, the algorithm appears to be a promising technique which can be utilized in the context of the game Rimboe Road to create AI opponents of varying difficulty. The experiment together with its results come with notable details to discuss.

Firstly, the explanation at the start of the experiment did not include a clear description over the color coding in the user interface. To some, especially younger participants, the color coding might have been obvious as it is commonly used. However, the absence of a legend or explanation of the color coding might have caused confusion to some participants. One of participant reported that they experienced difficulty understanding which player they were playing. Participants who lack an understanding of how the game works will experience more difficulty observing a difference in strength of their opponents and also play without the use of strategic thinking. The absence of a legend for the colour coding might have caused faulty data and affected the results of the experiment. Future research should take this into account and avoid any possible ambiguity.

Furthermore, half of the participants engaged in this experiment in an uncontrolled environment which can influence the validity of the results. One notable observation was how much time this half of the participants spent on the experiment. The data of one of these participants was sent back within 5 minutes of receiving the actual experiment itself. The participants who completed the experiment in a controlled environment spent at least 10 minutes to complete the whole experiment. It is possible that the participants in the uncontrolled environment rushed the experiment and did not spend time to strategically think about their moves. Another notable observation is that the only cases where a human beat the most difficult AI opponent was from participants which conducted the experiment in a controlled setting, further supporting the claim that the participants in the controlled setting were more focused and performed better. Even if participants were playing without strategic reasoning because of this experimental flaw, the validity of the data would not be affected.

Moving on to the implementation of the algorithm, it is worth taking a look at the validity of the implemented heuristic function. The current heuristic function assumes that the state of the board can be purely be evaluated based on the number of moves both players have. The issue with this assumption is the fact that some moves can at first seem disadvantageous towards the player while instead they allow a better position later on. This is similar to sacrificial tactics in chess where players willingly give up a piece to gain an advantage later on. Such moves will be labeled as 'bad' by the heuristic function while they could function as a bridge towards a new position which provides the player with a lot of new possible moves. There is a chance that the heuristic function blocks the AI from making moves which could allow it to win in specific situations. This problem is called the Horizon Effect and happens when the algorithm only searches a fixed number of nodes, causing it to possibly make a detrimental move (Berliner, 1973).

Something remarkable found in the data is the fact that not a single participant has rated an opponent a value lower than 3 or higher than 9. This can be caused due to something called central tendency bias (Baron & Hershey) which is a phenomenon where respondents choose options closer to the middle of the scale. This could have influenced the distribution of responses.

Finally, delving in the results of the experiment. The TukeyHSD test displayed both for the performance and the experienced difficulty a non significant difference between Max and Gabriel. A few notable details from this research can showcase the underlying factors which contribute to this performance variability.

One possible factor of influence for the insignificant results might be the experience of the participants. None of the participants played this game prior to partaking in the experiment. This might have caused the participants to exhibit the same performance across the different opponents which blocks them from experiencing difference in game play when playing against the different bots. Their inexperience causes them to explore the game while already doing the experiment and this might hinder their strategic planning.

Technical limitations might have further influenced this result as the experiment was constructed such that the participants would not be able to get an idea of which opponent might be stronger compared to the other. This limited the maximum strength which could have been set to the minimax algorithm as AI opponents with a depth value higher than 6 create a visual delay, allowing the participant to know which opponent spends more time computing their move. This caused the AI opponents to be of similar strength and to have similar performance. A possible solution to this issue in the future is to implement the strength of the algorithm in a different manner. Currently, from the start of the game, the depth (strength) of the AI opponent is set to a static value. The issue with this is that it has to perform the whole match with this set depth. While this value is extremely useful from the middle or end of the game, it makes the AI computational heavy from the start of the game as it has many possible moves to choose from. This should not be necessary as the first few moves of a match are not as crucial as the moves later on. The depth of the algorithm can instead be set dynamically throughout the match with the use of a function. An example of such a function: $depth = 20 - (A * f)$. Where A is the amount of available spaces on the board and f some factor with which you set the strength of the algorithm. This causes the depth to increase over the course of the game and it prevents the AI opponent from processing an unnecessary high amount of moves at the start of the game. The lower the value of f, the stronger the AI. Besides the algorithm's implementation, it is also possible to boost the speed of the program by using a different programming language. Changing from python, an interpreted language, to a compiled language, can result in performance improvements due to the inherent differences in how these languages are executed. In the case of another experiment, this would allow the developers to set a higher value for the depth without creating a visual delay, creating a performance gap.

References

Baron, J., & Hershey, J. C. (1988). Outcome bias in decision evaluation. *Journal of personality and social psychology*, *54*(4), 569.

Berliner, H. J. (1973). Some necessary conditions for a master chess program. In *Ijcai* (Vol. 3, pp. 77–85).

Campbell, M., Hoane, A., & hsiung Hsu, F. (2002). Deep blue. *Artificial Intelligence*, *134*(1), 57-83. doi: https://doi.org/10.1016/S0004-3702(01)00129-1

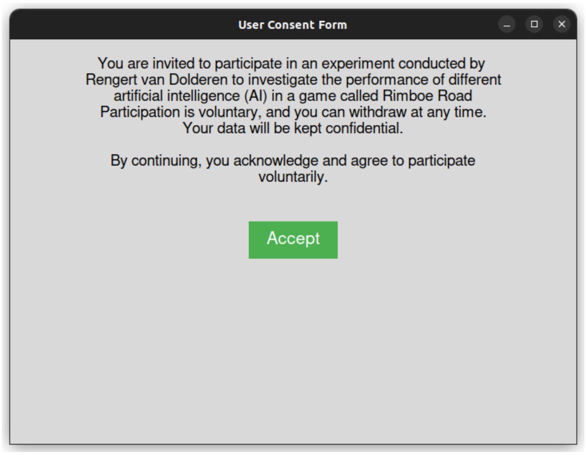
Lin, D., Wang, Q., & Yang, P. (2018). The game theory: applications in the wireless networks. *Game Theory—Applications in Logistics and Economy*.

Python Software Foundation. (2022). *Tkinter: Standard Python interface to the Tk GUI toolkit.*

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Schaeffer, J. (2020). The 1970 united states computer chess championship: The start of the longest-running experiment in computer science history. *ICGA Journal*, *42*(2-3), 72–85.

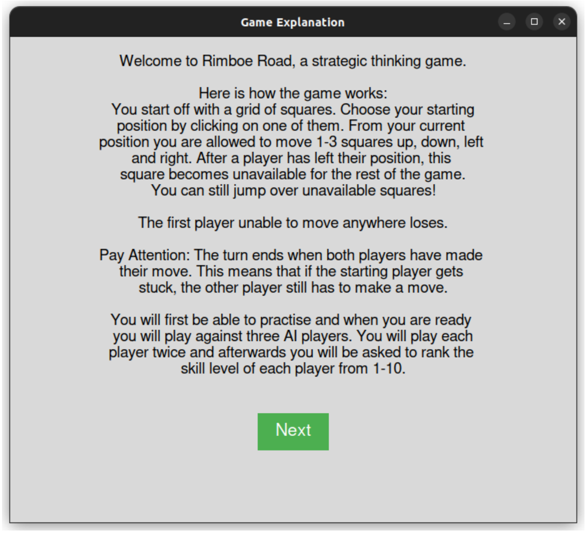
A Appendix


User Consent Form

You are invited to participate in an experiment conducted by Rengert van Dolderen to investigate the performance of different artificial intelligence (AI) in a game called Rimboe Road Participation is voluntary, and you can withdraw at any time. Your data will be kept confidential.

By continuing, you acknowledge and agree to participate voluntarily.

Figure A.1: Informed Consent Form


Game Explanation

Welcome to Rimboe Road, a strategic thinking game.

Here is how the game works:
You start off with a grid of squares. Choose your starting position by clicking on one of them. From your current position you are allowed to move 1-3 squares up, down, left and right. After a player has left their position, this square becomes unavailable for the rest of the game. You can still jump over unavailable squares!

The first player unable to move anywhere loses.

Pay Attention: The turn ends when both players have made their move. This means that if the starting player gets stuck, the other player still has to make a move.

You will first be able to practise and when you are ready you will play against three AI players. You will play each player twice and afterwards you will be asked to rank the skill level of each player from 1-10.

Figure A.2: Game Explanation