Application of Minimax Algorithm with Alpha-Beta Pruning in the Game of Nim

Project report

Ayah Nassar

60101805

DSAI3203 – 1

Instructor

Dr. Somaiyeh MahmoudZadeh

University of Doha for Science and Technology

March 30, 2024

Table of Contents

Abstract

This report examines the Game of Nim using mathematical and computational methods. The game is a two-player, turn-based strategy game with applications in combinatorial game theory (geeksForgeeks, n.d.). A Python program simulating the game uses the Minimax algorithm with Alpha-Beta pruning to improve the AI's decision-making was developed giving it a high chance of winning. The report emphasizes the iterative approach employed to enhance the AI agent, which eventually beats human opponents. The report highlights the iterative approach used to enhance the AI agent, which eventually beats human opponents. The project demonstrates how traditional AI algorithms can contribute to developing strategies for games. In addition to showing the effectiveness of these algorithms, it also discusses the adaptability & flexibility of the AI in responding to the dynamic and energetic nature of human strategies, as well as illustrating the potential for AI in complex decision-making scenarios. Through multiple testing and analysis, the study provides valuable insights into the strengths and limitations of AI in games of strategy, suggesting pathways for future research in AI and game theory. This study not only highlights the technical achievements but also encourages a larger discussion on the implications of AI in understanding human cognition and decision-making processes.

1 Introduction

The Game of Nim, as a classic example of combinatorial strategy, offers a challenge for formulating and testing principles of artificial intelligence. Historically, the game's roots can be traced back to ancient times, yet it stands steadily within modern computational theory and practice. The game consists of a finite number of piles, each containing a certain number of items such as sticks or stones. Players take turns removing items from the piles, the goal is to be the last to perform a valid move under the game's rules (geeksForgeeks, n.d.).

The Game of Nim stands as a testimony to the enduring challenge of combinatorial strategy, encouraging players to engage in a battle of wits through its simple rules. This project embarks on an exploration of artificial intelligence (AI) by developing an AI agent to navigate the complexities of Nim, a game rooted in ancient civilization yet remarkably pertinent to modern computational theory. Utilizing the Minimax algorithm with Alpha-Beta pruning, this investigation not only gets to create an AI capable of competing at Nim but also aims to highlight the game's utility in revealing fundamental AI and game theory principles. Through this project, we underscore the educational significance of applying algorithmic strategies to deepen our understanding of AI behavior and decision-making processes (Julian, n.d.).

2 Background

The Game of Nim is a very old game that people have played for a long time. In this game, you have a few groups of items like sticks or stones. Players take turns taking away these items from the groups. The main goal is to be the smart one who makes the last move under the game's rules. Nim is interesting because it's all about making smart choices. There's no luck involved because everything you need to know is right in front of you. This makes Nim a perfect game for seeing how well different strategies work (Julian, n.d.).

Long ago, mathematicians started looking closely at games like Nim to understand better how to win at them. They discovered some cool math tricks that can help players decide the best moves to make. When we talk about using computers to play Nim, we use special math-based tricks to help the computer make smart moves. One of these tricks is called the Minimax algorithm. This trick helps the computer think about all possible moves it can make and then choose the best one. Another trick, called Alpha-Beta pruning, makes this process faster by ignoring moves that won't help the computer win (Julian, n.d.).

This section explores key artificial intelligence (AI) concepts that underpin the decision-making process in the Game of Nim. Understanding these concepts is crucial for comprehending how the AI developed for this project makes strategic choices.

- **Bayesian AI:** Bayesian AI involves decision-making with incomplete information, using probabilities to make informed guesses. While the Game of Nim is a game of complete information, Bayesian principles can help design AI that adapts to the player's strategies by updating beliefs about the player's behavior (TURING, n.d.).
- **State Space Search:** The Game of Nim can be represented as a state space, where each state represents a possible configuration of piles and objects. The AI uses state space search to navigate through these configurations, evaluating possible moves to make an optimal decision (Singh, 2023).
- **Normative Decision Theory:** This theory focuses on identifying the best decision to make under uncertainty. In the context of Nim, it helps in formulating strategies that maximize the AI's chances of winning, guiding the AI to evaluate the outcomes of its actions (Elliott, 2019).
- **Markov Decision Problems (MDPs):** MDPs are used to model decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. Although outcomes in Nim are deterministic, understanding MDPs aids in

developing algorithms for more complex games that the AI might encounter, enhancing its ability to plan and make decisions (stanford, n.d.).

- **Dynamic Programming:** Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It's particularly useful in optimizing the Minimax algorithm for Nim, allowing the AI to efficiently calculate the best move by reusing results from previously solved subproblems (BasuMallick, 2022).

Together, these concepts form the theoretical foundation for AI decision-making in the Game of Nim. By leveraging these ideas, the project not only demonstrates the AI's capability to strategize but also showcases the application of advanced computational techniques in understanding and developing intelligent behaviors. Moreover, studying Nim and using these computer tricks, we learn a lot about how computers can make decisions. This helps us understand more about artificial intelligence, which is all about making computers that can think and make decisions like humans (Julian, n.d.).

3.0 Methodology

This section outlines the process of developing an artificial intelligence (AI) to play the Game of Nim effectively against a human opponent. The AI's decision-making is powered by the Minimax algorithm with Alpha-Beta pruning, a choice motivated by the game's combinatorial nature and the need for efficiency in evaluating potential moves. The approach can be summarized in several key steps, as follows:

3.1 Game Initialization

The game begins by establishing a variable number of piles (n_piles), each randomly assigned a quantity of objects within a specified range (1 to 7 for this project). This randomness in the initial setup ensures variability in gameplay, requiring the AI to adapt its strategy to different starting conditions (Julian, n.d.).

3.2 Game State Display

A simple console-based interface displays the current game state to the player, indicating the number of objects in each pile. This visibility allows both the AI and the human player to make informed decisions based on the same information, reflecting the game's "perfect information" characteristic.

3.3 Decision Making

The core of the AI's strategy lies in the Minimax algorithm, enhanced by Alpha-Beta pruning:

- **Minimax Algorithm:** This recursive function evaluates all possible moves and their outcomes, aiming to minimize the possible loss for a worst-case scenario which is assuming the opponent plays optimally. The algorithm alternates perspectives between maximizing and minimizing players, reflecting each player's aim to win (Brennan, 2023).

- **Alpha-Beta Pruning:** This technique improves the efficiency of the Minimax algorithm by eliminating the need to explore fewer moves. By setting thresholds (alpha for the maximum lower bound and beta for the minimum upper bound), the algorithm prunes branches of the game tree that won't affect the final decision, speeding up the decision-making process (Brennan, 2023).

3.4 AI and Player Moves

- **AI Move Execution:** Based on the Minimax algorithm's evaluation, the AI selects and executes the optimal move by removing objects from one of the piles. This move is dynamically calculated to adapt to the current game state, showcasing the AI's ability to strategize.
- **Player Input Handling:** The human player interacts with the game through console inputs, choosing a pile and the number of objects to remove. This input is processed and reflected in the game state, maintaining the turn-based nature of the game.

3.5 Game Loop

The game proceeds in a loop where the current state is displayed, and players alternate turns until a terminal state is reached (all piles are empty). The loop's structure facilitates continuous gameplay and decision-making, highlighting the iterative nature of turn-based strategy games.

3.6 Conclusion of Game

The game concludes when one player removes the last object, resulting in a loss for the other player. This outcome is immediately displayed, signaling the game's end and demonstrating the AI's competency or the human player's skill.

4 Experiments and Outcomes

Our AI, created to excel in Nim, faced human opponents in multiple contests. The AI displayed impressive strategic abilities due to the Minimax algorithm with Alpha-Beta pruning. While the AI didn't strive for perfection, its victories highlighted the practical value of these algorithms. The AI consistently beaten the human opponent, demonstrating the strength of its decision-making process. This testing phase showcased AI's effectiveness in challenging human players.

5 Analysis

When tested in multiple games, the AI's Minimax algorithm with Alpha-Beta pruning helped it make smart strategic choices. However, human players can be unpredictable, making the AI's moves less effective. When the AI faced unique or unexpected strategies from humans, these games showed how well the AI could adjust its decision-making. This variability highlights the key fact in combative games: the outcome depends on the strategies used by both sides and how they interact. Despite the AI's advanced strategy, its reliance on logic and prediction sometimes proved insufficient against the unexpected and inventive tactics of human opponents. This underscores the inherent unpredictability in AI vs. human interactions in strategic games. While this analysis shows the AI's strengths, it also identifies areas for improvement, especially in handling unconventional and creative strategies.

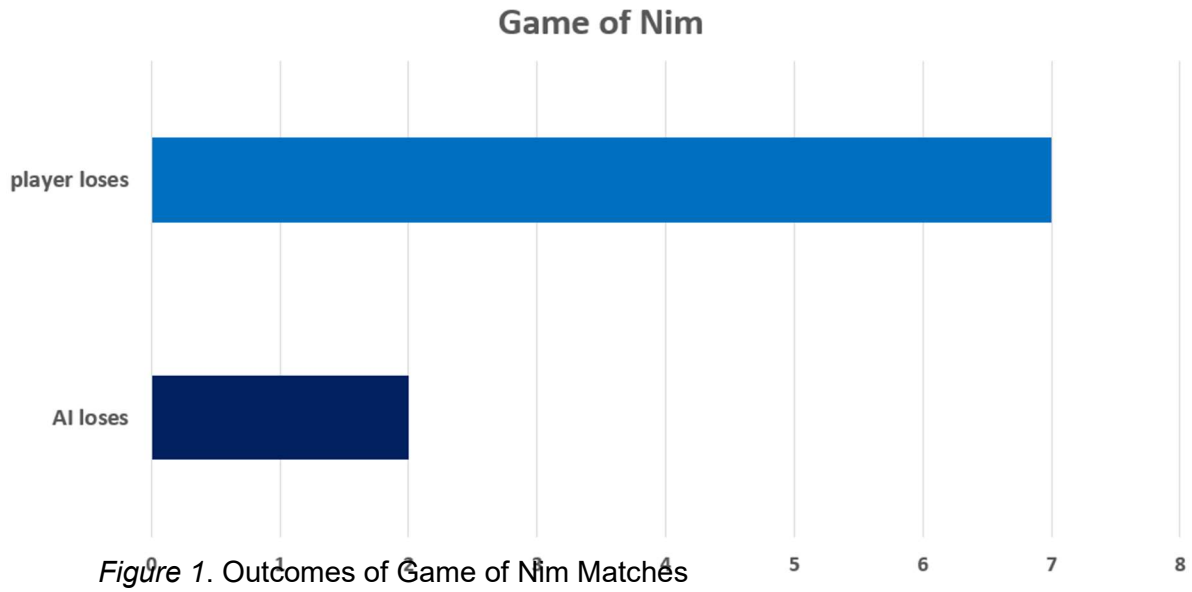

Figure 1. Outcomes of Game of Nim Matches

Figure 1 presents a bar chart that compares the number of games won by the AI against the number of games won by human players in several matches. It is marked that human player tend to lose more frequently, as indicated by the longer blue bar indicating player losses. In the other hand, the shorter navy bar indicates the AI's losses, indicating that the AI designed for this project generally outperformed the human participants. This visual representation emphasizes the effectiveness of the Minimax algorithm with Alpha-Beta pruning in leading the AI to victory in most of the games played.

6 Conclusion

This project was about making a computer program, or "AI," that could play the Game of Nim really well. We used some smart techniques called the Minimax algorithm and Alpha-Beta pruning. These techniques helped the computer think ahead and make really good moves. Even though the game is pretty old, it was a cool way to show how these computer tricks can do some amazing things. The computer did a great job at playing the game. It could beat people most of the time, which was pretty impressive. But playing against people was also hard because people can be very unpredictable. They don't always play the game the way the computer expects, which makes it a tough challenge for the AI.

We learned a lot from this project. It showed us that the ideas we use to make computers smart are really powerful. But it also showed us that there's still a lot to learn, especially when it comes to guessing what a person might do next. This project wasn't just about winning a game. It was a way to learn more about AI and how we can teach computers to think and make decisions. It was a fun challenge, and it gave us some good ideas about how to make AI even better in the future.

In short, we made a computer program that's really good at the Game of Nim. It showed us how smart computer techniques can be used in games. And it made us think about how we can keep improving AI to deal with unexpected moves, especially when playing against humans. This project was a great way to see how far we've come in making smart AI, and it gives us lots of ideas for what to do next.

Appendix

Pseudo code of the python code:

```
import random library

Function initialize_game with parameter n_piles defaulting to 3:
  Initialize an empty list called 'state'
  Repeat n_piles times:
    Append a random number between 1 and 7 to 'state'
  Return 'state'

Function display_state with parameter state:
  Print "Current game state:"
  For each pile in state with index i:
    Print "Pile", i, ":", visual representation of pile, "(", number of objects in pile, ")"

Function minimax with parameters state, alpha, beta, maximizingPlayer:
  If sum of state is 0 (game over):
    Return 1 if maximizingPlayer else -1

  If maximizingPlayer is True:
    Set maxEval to negative infinity
    For each pile in state with index i:
      If pile is not empty:
        For remove from 1 to pile size:
          Copy 'state' to 'newState' and remove 'remove' objects from pile 'i'
          Calculate 'eval' by calling minimax on 'newState', alpha, beta, and False
          Set maxEval to maximum of maxEval and eval
          Set alpha to maximum of alpha and eval
          If beta is less than or equal to alpha, break
```

```
    Return maxEval
  Else:
    Set minEval to positive infinity
    For each pile in state with index i:
      If pile is not empty:
        For remove from 1 to pile size:
          Copy 'state' to 'newState' and remove 'remove' objects from pile 'i'
          Calculate 'eval' by calling minimax on 'newState', alpha, beta, and True
          Set minEval to minimum of minEval and eval
          Set beta to minimum of beta and eval
          If beta is less than or equal to alpha, break
    Return minEval
Function ai_move with parameter state:
  Set bestEval to negative infinity
  Set bestMove to None
  For each pile in state with index i:
    If pile is not empty:
      For remove from 1 to pile size:
        Copy 'state' to 'newState' and remove 'remove' objects from pile 'i'
        Calculate 'eval' by calling minimax on 'newState', negative infinity, positive infinity, and False
        If 'eval' is greater than bestEval:
          Set bestEval to 'eval'
          Set bestMove to tuple (i, remove)
  Remove 'bestMove[1]' objects from pile 'bestMove[0]' in state
  Print AI's action
Function player_move with parameter state:
  Ask player to choose a pile and store it in 'pile'
  Ask player to choose number of objects to remove from the pile and store it in 'remove'
  Remove 'remove' objects from 'pile' in state
```

```
Function game_loop:
  Set 'state' by calling initialize_game()
  While sum of state is greater than 0:
    Call display_state with 'state'
    Call player_move with 'state'
    If sum of state is 0:
      Print "Player loses!"
      Break
    Call ai_move with 'state'
    If sum of state is 0:
      Print "AI loses!"
      Break
Main Program:
  Call game_loop()
  Print "Game over!"
```

References

BasuMallick, C. (2022, 10 19). *What is Dynamic Programming?* Retrieved from spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/

Brennan, A. (2023, 1 20). *Minimax algorithm and alpha-beta pruning*. Retrieved from Medium: https://medium.com/@aaronbrennan.brennan/minimax-algorithm-and-alpha-beta-pruning-646beb01566c#:~:text=Alpha%2Dbeta%20pruning%20is%20a,be%20worse%20than%20other%20branches.

Elliott, E. (2019, 9 16). *Normative Decision Theory*. Retrieved from ANALYSIS: https://academic.oup.com/analysis/article-abstract/79/4/755/5570297?redirectedFrom=fulltext

geeksForgeeks. (n.d.). *Combinatorial Game Theory | Set 2 (Game of Nim)*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/

Julian, R. (n.d.). *The Game of Nim*. Retrieved from wikimath: https://wiki.math.wisc.edu/images/Nim_sol.pdf

Singh, N. (2023, may 8). *State Space Search in Artificial Intelligence*. Retrieved from scaler: https://www.scaler.com/topics/artificial-intelligence-tutorial/state-space-search-in-artificial-intelligence/

stanford. (n.d.). *Markov Decisions*. Retrieved from stanford: https://stanford.edu/~cpiech/cs221/handouts/markovDecisions.html#:~:text=Markov%20Decision%20Problems%20(MDPs)%20extend,much%20richer%20set%20of%20problems.

TURING. (n.d.). *An Overview of Bayesian Networks in AI*. Retrieved from TURING: https://www.turing.com/kb/an-overview-of-bayesian-networks-in-ai