Connect 4: Advanced AI Algorithms

Gagana Ananda[1], Shubham Bansal[2], Vansh Gupta[3],
[1]Northeastern University
[2]Khoury College of Computer Science
ananda.g@northeastern.edu, bansal.shu@northeastern.edu, gupta.vansh2@northeastern.edu

Abstract

The main goal of this project is to create a competent AI agent that can participate in Connect 4. Our attention has been on the agent's capacity to adapt to any kind of opponent, including humans and other AIs. We provide a traditional method that guarantees that the AI will behave consistently regardless of how well or poorly the opponent plays. The AI agent consists of advanced AI methodologies along with Expectimax, Minimax, Minimax with Alpha-beta pruning, and Monte Carlo Tree search. Increasing the AI's overall performance and improving decision-making ability requires those algorithms. It equips the AI agent with the optimal route at any given time and ensures that it can make strategic decisions, which is crucial depending on whether the opponent is a human or an AI player. This project demonstrates how an actual time version of and counteraction of human techniques through sophisticated AI algorithms may additionally revolutionize gaming. Future developments in AI may be prompted with the aid of the insights we gain from studying the performance of different algorithms. This is a massive improvement for artificial intelligence and games alike.

Introduction

Artificial Intelligence (AI) in video games serves as a vital tool in each academic and research context, providing a managed yet complicated environment wherein diverse AI strategies can be examined and refined. The implementation of AI in video games no longer best facilitates beautifying the gaming experience. However, additionally contributes considerably to the development of the system gaining knowledge of techniques, decision-making approaches, and strategic making plans.

Problem Statement

Connect 4's primary goal was to see whether AI agents created, using different algorithms, could effectively compete with one another. to observe the AI's flexibility in terms of how it plays strategically versus other AI opponents as opposed to competing against humans.

To begin with, using a variety of approaches, we analyzed and constructed an AI agent that is fully independent and adaptive to its environment. Among these algorithms are Monte Carlo Tree Search (MCTS), Expectimax, Minimax with Alpha-beta pruning, and Minmax. By opting for these techniques, we can ensure that our AI agent has all the qualities needed to contend with other AI players.

This report gives a fundamental understanding of the algorithms we use and the performance of the AI agents in response to the selected algorithms. We have also discussed issues that were observed throughout the development process and provided suggestions for future applications of this research, particularly about the AI gaming environment. The algorithms are thoroughly described, along with the rationale behind their use in the context of the AI's complexity in performing against one another. One of our sophisticated algorithms, MCTS, is regarded as the most effective as it responds effectively and enhances the AI's capacity for decision-making by offering the optimal path against AI opponents.

Algorithm Overview

Brief overview of the Algorithms used in the project:

Expectimax

Expectimax is an excellent algorithm for managing many strategies. It is adept at responding to a variety of unexpected and varied AI strategies because it can assess future game states by averaging all possible states.

Minimax

By reducing the likelihood of failure and increasing the likelihood of success, this approach models every conceivable future situation to assist the AI in choosing the optimal course of action. In particular, whether people or AIs are designed to play at their best, it guarantees that AI performs well.

Alpha-Beta Pruning

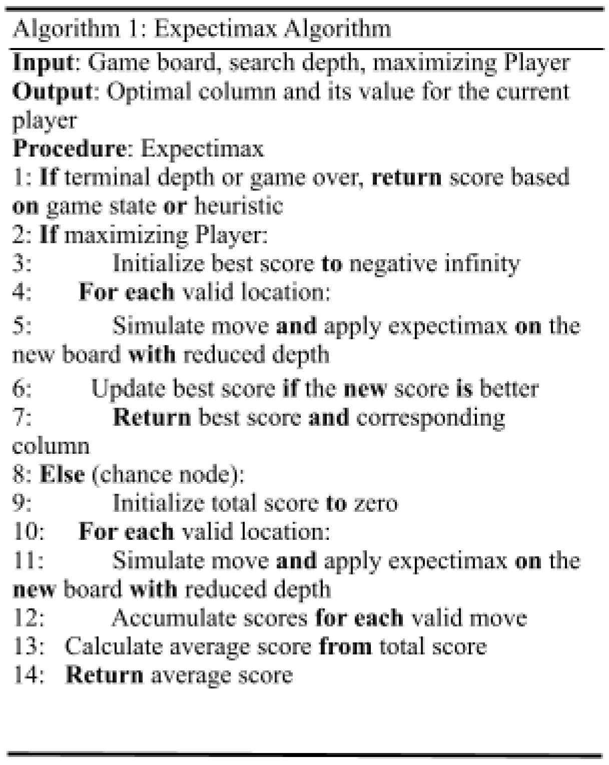
Building at the Minimax method, this method complements efficiency by cutting out hunting tree additives that don't impact the final decision. This enables the acceleration of the choice-making technique for the AI, permitting it to reply more quickly without sacrificing strategic intensity.

Monte Carlo Tree Search (MCTS)

Using random simulations, this technique explores and evaluates capability moves from the modern activity country. It's extremely beneficial for managing complex endeavor conditions and excels at helping the AI adapt to and counter the tactics of sophisticated AI opponents.

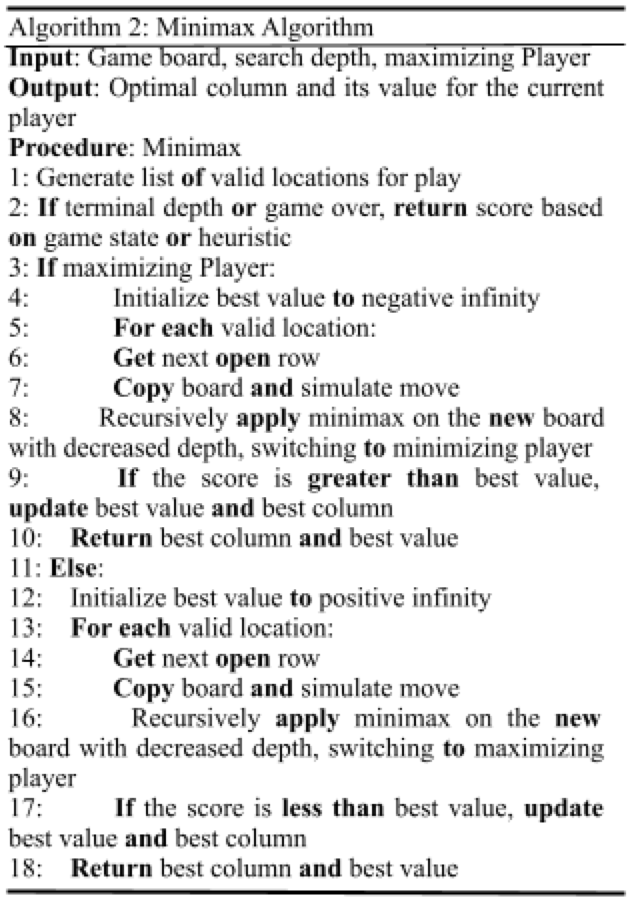
Algorithms

Expectimax



```
Algorithm 1: Expectimax Algorithm
Input: Game board, search depth, maximizing Player
Output: Optimal column and its value for the current player
Procedure: Expectimax
1: If terminal depth or game over, return score based on game state or heuristic
2: If maximizing Player:
3:         Initialize best score to negative infinity
4:     For each valid location:
5:         Simulate move and apply expectimax on the new board with reduced depth
6:       Update best score if the new score is better
7:         Return best score and corresponding column
8: Else (chance node):
9:         Initialize total score to zero
10:    For each valid location:
11:        Simulate move and apply expectimax on the new board with reduced depth
12:        Accumulate scores for each valid move
13:  Calculate average score from total score
14:  Return average score
```

Minimax



```
Algorithm 2: Minimax Algorithm
Input: Game board, search depth, maximizing Player
Output: Optimal column and its value for the current player
Procedure: Minimax
1: Generate list of valid locations for play
2: If terminal depth or game over, return score based on game state or heuristic
3: If maximizing Player:
4:         Initialize best value to negative infinity
5:         For each valid location:
6:         Get next open row
7:         Copy board and simulate move
8:       Recursively apply minimax on the new board with decreased depth, switching to minimizing player
9:         If the score is greater than best value, update best value and best column
10:    Return best column and best value
11: Else:
12:    Initialize best value to positive infinity
13:    For each valid location:
14:        Get next open row
15:        Copy board and simulate move
16:          Recursively apply minimax on the new board with decreased depth, switching to maximizing player
17:        If the score is less than best value, update best value and best column
18:    Return best column and best value
```

Alpha-Beta Pruning

```
Algorithm 3: Alpha-Beta Pruning Algorithm
Input: Game state, depth, player ID
Output: Best action for the current player
Procedure: Alpha-Beta
1: Initialize alpha to negative infinity and beta to positive infinity
2: Define recursive function to handle both min and max computations
3: If terminal state or depth is 0, return utility or heuristic evaluation
4: For maximizing player:
5:         Initialize best value to negative infinity
6:         For each action, compute value with recursive call and update alpha
7:         If value is better, update best value
8:         If best value exceeds beta, break loop (beta cutoff)
9: For minimizing player:
10:    Initialize best value to positive infinity
11:    For each action, compute value with recursive call and update beta
12:    If value is better, update best value
13:    If best value is less than alpha, break loop (alpha cutoff)
14: Return best value and action for initial call
```

Monte Carlo Tree Search (MCTS)

```
Algorithm 4: Monte Carlo Tree Search
Input: Initial game state, player ID, time limit
Output: Best move
Procedure: UCB1
1: If node visits are zero, return infinity
2: Else, calculate and return UCB1 score
Procedure: Rollout
1: Simulate game until terminal state
2: Return utility value for player ID
Procedure: Backpropagation
1: Increment node visits, add value
2: Propagate value up to parent nodes
Procedure: Expansion
1: Create children for all possible actions
2: Return first child
Procedure: Selection
1: Return child with highest UCB1 score
Procedure: Best Move Calculation
1: Return action with highest average value from root node children
Procedure: MCTS
1: Initialize MCTS with root node
2: While within time limit
3:         Traverse tree using Selection and Expansion
4:         Simulate game using Rollout
5:         Update nodes using Backpropagation
6: Return Best Move Calculation
```

Data Representation

The game consists of

- State board: illustrated in the form of a matrix with six rows and seven columns of spaces.
- State spaces: Each space is marked with a number to determine its state.
  - ‘0’ for a space with no coins.
  - ‘1’ First player’s coin in the space
  - ‘2’ Second player’s coin in the space

The Agent may analyze the state board and develop methods with the aid of numbers or symbols, which is important for the AI’s optimal performance and results.

Performance Metrics

The list of metrics we have used to analyze each algorithm's performance is as follows:

- Win Rate: Determines the number of games won by the total number of games played by the AI.
- Algorithmic performance: Utilizing the win rate metric, we evaluate each model's performance when pitted against each other in a simulated environment.
- Player-specific performance: Analyzed each algorithm’s win rate when playing first and when playing first as playing first has a bias.

These metrics helps us to capture the AI’s efficiency and growth according to the game and learn the adaptability of its decision-making ability.

Project Repository

The complete source code and implementation details for the Connect 4 AI algorithms covered in this paper can be found on GitHub at:

https://github.com/Gagana1312/CS5100_FAIProject_Connect4

Results

The results are represented in various formats as shown below.

Adaptability and Performance

Algorithms	Wins	Losses	Draws
Expectimax	17	19	0
Minimax	27	9	0
Alpha Beta Pruning	11	25	0
Monte Carlo Tree Search	17	19	0

Figure 1: Performance summary of each Algorithm.

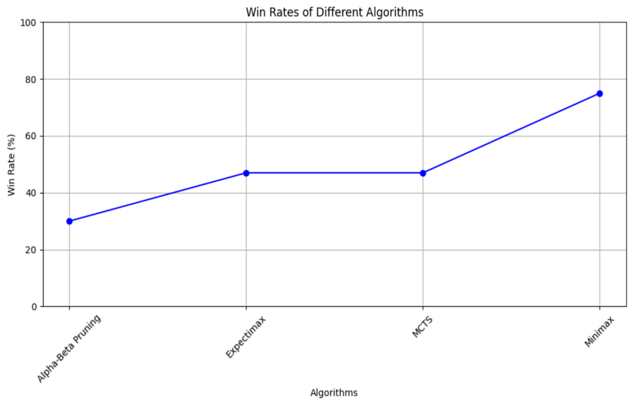
Player 1

Algorithms	Wins	Losses
Minimax	15	3
Expectimax	12	6
Alpha Beta	9	9
MCTS	5	13

Player 2

Algorithms	Wins	Losses
Minimax	12	6
Expectimax	5	13
Alpha Beta	2	16
MCTS	12	6

Fig: Algorithm-Specific Performance Metrics



Discussion

Analysis of Results:

Level of Performance	Algorithm	Win Rate
Poor	Alpha-Beta Pruning	30%
Average	Expectimax	47%
Average	Monte Carlo Tree Search (MCTS)	47%
Best	Minimax	75%

Contrary to our initial assumption that MCTS will outperform every algorithm, we observe above that Minimax is outperforming all the other algorithms. There are several reasons for that -

1. Game Complexity and Horizon Effect: Connect 4 has a relatively manageable game complexity compared to more complex games like Go or Chess. The horizon effect refers to the limited depth of the game tree that can be effectively explored within a given computational budget. Minimax, with its ability to look ahead to a fixed depth and evaluate positions using a static evaluation function, might perform well within the horizon of Connect 4.

2. Depth of Search: Minimax performs a systematic depth-first search up to a specified depth in the game tree, whereas MCTS incrementally builds a search tree based on statistical sampling. In games like Connect 4, where the game state space is well-defined and not overly large, a deeper and more exhaustive search by minimax within its limit will lead to more accurate evaluations and decisions compared to the more sampling-based approach of MCTS.

3. Evaluation Function and Heuristics: The effectiveness of minimax depends heavily on the quality of the evaluation function used to assess game positions at the leaf nodes of the search tree. For Connect 4, a well-tuned evaluation function that accurately predicts winning chances based on board configurations (e.g., considering open rows, columns, diagonals) can lead to strong performance by minimax. In contrast, MCTS relies on simulation and statistical estimates of winning probabilities, which can be more variable and less precise in games with relatively simple dynamics like Connect 4.

We also notice that MCTS performs equally well compared to Minimax when playing as player 2. As Player 2 in Connect 4, MCTS leverages its adaptive and exploratory nature to navigate through the potential game states created by Player 1's moves. By sampling and evaluating different strategies through simulation and tree expansion, MCTS can identify effective responses even when the opponent's actions are not fully predictable.

Theoretical Implications

Comparing Minimax, Monte Carlo Tree Search (MCTS), Expectimax, and Alpha-Beta Pruning in Connect 4 helps us see which methods work best for different games and situations. By studying how these methods do in Connect 4, we can learn important things about what they're good at and what they're not so good at. This research tells us how much time and computer power each method needs and shows us the best ways to play the game. It also helps us understand how to make smart choices in games and other areas. Additionally, it makes us think about using these methods fairly and ethically, so everyone can have fun playing games without worrying about cheating or unfairness.

Looking at Minimax, MCTS, Expectimax, and Alpha-Beta Pruning in Connect 4 isn't just about games—it's about understanding how computers make choices in different situations. This study gives us insights into how to design smart computer systems and solve problems better. By comparing these methods, we learn about the challenges of making smart systems and how to use them in a fair and responsible way. This research lays the groundwork for improving artificial intelligence and making sure it's used in a fair and ethical manner in games and beyond.

Practical Applications

These algorithms are intended to generate tactics for games such as Go, Chess, and Connect 4. These algorithms take similar techniques; modeling, designing, and simulation are important, as is tactically addressing the move predictions and understanding the competition's conduct.

From an academic angle, these algorithms are crucial to realize seeing that they teach us how AI ideas are characteristic both theoretically and logically and offer us with some instances of ways AI might be tested to healthy distinct necessities.

Limitations and Challenges

Reducing computing overhead was one of the challenges encountered throughout project development. The fact that we have to compel out-of-borders to fetch in algorithms such as MCTS and alpha-beta pruning might potentially impair the real-time or resource-constrained applications' performance.

Another problem was adjusting the MCTS with the configurations to make the algorithm function more effectively in order to strike a balance between research and overuse, or improving the heuristic estimates for Minimax and Alpha-beta pruning to deal with the opponent's conduct. These constraints were significant because they allowed us to construct our AI to adapt to its own set of settings without human involvement.

Conclusion

Summary of Findings

Our study, which used Minimax, Expectimax, Alpha-beta Pruning, and Monte Carlo Tree Search to the notion of a game named Connect 4, gave us several insights. Minimax and MCTS have shown to be significantly better strategic decision-makers and competitive players that are concerned with their victory rate. Alpha-beta pruning is a safer strategy, as it assumes that the opponent will likewise play optimally. Expectimax took care of any questionable movements, resulting in a higher victory percentage.

Future Work

The present day observation lays the groundwork for in addition studies, in particular in improving the efficiency and strategic depth of those algorithms. Potential sectors of future work include mainly two categories:-

AlphaZero, developed by DeepMind, merges deep neural networks with reinforcement learning to excel in intricate games without prior knowledge. In Connect 4, AlphaZero would grasp optimal strategies via self-play, constantly refining its gameplay by assessing the board state and making informed moves.

Bit boards condense the game state using binary numbers, offering a highly efficient representation. Each bit signifies a specific game element (e.g., whether a square holds a player's token). Leveraging bit boards can notably fasten operations like move generation and assessment, boosting the effectiveness of algorithms such as minimax, Monte Carlo Tree Search (MCTS), and alpha-beta pruning. This way there are more possible number of iterations performed in the same amount of time and thereby upgrading each model’s performance.

However, it wasn’t possible to include these two approaches in the current implementation as Deep Reinforcement learning involved complex and heavy training which wasn’t feasible. As for Bit Boards, it involved changing the entire game architecture along with changing calculating winning moves, possible actions and other operations pertaining to the Connect 4 game board.

Final Thoughts

This project has highlighted the challenge of balancing complexity and reliability in AI gaming algorithms. It reaffirms the anticipated behaviors of these algorithms in controlled environments like Connect 4 with the only discrepancy being the performance of MCTS, showcasing their strengths and limitations. Minimax looks to be the promising algorithm for this scenario as it explores the entire state space till the specified depth which is good enough for simplistic games like Connect4 but might not perform as well for more complex environments. Beyond enhancing our understanding of algorithmic techniques in gaming scenarios, this project has also improved our practical skills in executing and evaluating AI solutions. We learned how different deterministic and heuristic algorithms perform under the same simulated environment.

Additional Observations

Interestingly, the performance dynamics observed in a couple of the algorithms correspond to theoretical expectations for Connect 4 games. Minimax negotiated aggressive circumstances with ease, whereas the strategic usage of Alpha-Beta Pruning and the probabilistic nature of Expectimax showed unique approaches to coping with game uncertainty. The complex MCTS was limited in it's performance due to the limit on the number of iterations it could perform and hence, wasn't able to find the optimal solutions from the generated limited set of random possible outcomes. These investigations and findings contribute to a much broader knowledge of AI packages in technique video games and lay the groundwork for destiny advances inside the discipline.

References

-https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f

-https://arxiv.org/pdf/1802.05944.pdf

-https://www.academia.edu/46786117/Reinforcement_learning_in_Connect_4_Game