Optimizing Play with Minimax in Connect Four

Garrison Creek, Zachary Nelson, Matthew Higdon

Introduction

Connect Four is a solved game. Player one, can guarantee a win every game with perfect play. Connect Four's game board has around 4.5 trillion different possible positions. Because of this, perfect play is hard to achieve. The only perfect move that can be guaranteed on every game is the first move of the game. This move is for player one to drop their token in the middle column as their first move. This is mathematically the only way for player one to force a win, but perfect play still has to continue for the rest of the game. If player one does not play its first piece in the middle column, and instead places it in one of the two adjacent columns to the center, the second player could then force a draw with perfect play. Finally if player one plays their first move in any other column then the center most three, then player one could be forced into a loss.

The first goal of our project is to create an AI capable of perfectly playing connect four. Measuring perfect play is difficult for Connect Four though. For instance, if both players play perfectly, player one will always win. So how can player two be said to be playing perfectly? The best that player two can hope to achieve is to draw the game out until it is close to a draw before losing. The metrics we can use to judge our AI on is that as player one, it should start the game by playing the middle column every time, and it should always win. If the AI is player two, we can judge it on that the AI should lose on or near to the 41'st or 42'nd moves. We can also judge it on if it is capable of beating human players as well. Measuring the perfect playing for the AI as player two, requires player one to be playing perfectly though. Rather than having the same AI play itself we decided to make the project more interesting by creating a second AI. One would use the Minimax algorithm, the other would use Monte Carlo Tree Search. These AI's will play against each other and the outcome will be recorded. To better judge performance, the positions of player one and two will be switched and tested as well. We hope to be able to determine which of these algorithms can perform more optimally in the game of Connect Four, within a time constraining environment. Due to Minimax's deterministic nature, as well as MCTS requiring prior simulations before choosing, we hypothesize that Minimax will outperform MCTS in an environment with tight time constraints.

Background

The two algorithms that we are testing are Minimax with alpha-beta pruning, and a Monte Carlo Tree Search (MCTS). Minimax works by attempting to minimize loss, and maximizing gain. It is best applied to environments of two opposing players where one player's loss is another's gain, and vice versa. In Connect Four, each move would be considered as a branch on a tree, and future moves represent layers of a tree. The bottom of the tree would be the ending conditions. The minimax algorithm takes turns choosing the branch of the tree that maximises its score, while the opponent chooses the branch that minimizes it. It then works down the tree alternating the best possible move, and worst possible opposing move until it finds what initial move leads to the best outcome.

Minimax is often optimized with alpha-beta pruning. While not changing the final decision made by the algorithm, it "prunes" the branches of the tree by removing branches in which there exist better choices for itself to take, or worse choices for the opponent to take. This allows quicker computation as the program only traverses necessary branches in the tree, while avoiding branches that neither it, nor its opponent will make.

Monte Carlo Tree Search is an algorithm that, like Minimax, explores every possible scenario before choosing the optimal path. Unlike Minimax, MCTS runs random simulations of how games might play out depending on its choices. After running many simulations, MCTS builds up statistical knowledge of optimal choices and winning moves. These moves are then revisited more in future simulations, and statistical knowledge grows. The more simulations that are ran, the better estimates of what the best choices are. This allows for optimal paths and choices without evaluating choices or functions.

While exploring research in Minimax and MCTS, we found several published works comparing the two in different environments. Last June, researchers from the KTH Royal Institute of Technology in Sweden conducted a case study into Minimax and Monte Carlo to determine which was more optimised at chess. They determined that, if given adequate resources, MCTS outperforms Minimax. Minimax, however, outperformed MCTS in less-complex environments. (Wilkens and Khoury, 2024). We also found that knowledge of the performance differences between MCTS and Minimax can open the door for work on future algorithms. Researchers at Maastricht University in the Netherlands proposed a hybrid model which combines Minimax and MCTS. This would combine the advanced statistics of MCTS, with the quick deductive reasoning of Minimax to create a more optimized, encompassing algorithm (Baier and Winands).

Methodology

Our Minimax algorithm is initialized with its player number (1 or 2), and a time limit of five seconds. On its turn it gets passed a copy of the board to start recursively evaluating potential moves. Our algorithm uses custom heuristics for move prioritization and scoring board states. One of the biggest challenges in using Minimax is that it is only as good as its heuristics, and that can be difficult for some games without definitive scoring. Creating the heuristics we use in the score_board function, required trial and error in discovering what is effective in Connect Four. The function takes in a board state and returns a score rating for it. The biggest factor in scoring, is the detection of a winning pattern, which adds a positive or negative value of 10,000 depending on if it is maximising or minimizing. This is exponentially larger than other factors of scoring, which is to ensure that a move that wins or leads to an instant loss is not overshadowed by anything else. Then the function checks for each instance of three and two in a row, and adds or subtracts exponentially increasing values of 100 and 50 respectively to each instance. Finally it adds 3 points for each of its players tokens in the center column. These points are small, so that they only really weigh in on placement decisions when all options would be valued at equal points. This causes the AI to choose the center column every time on equal point decisions like the first turn. It also allows the AI to tie-break in equal point decisions later in the game by playing the center if it's available, this is good because the center will always have more opportunities to connect four than other columns. The AI uses these scores to evaluate the different possible moves and uses alpha-beta pruning to reduce the tree size for efficiency.

The Monte Carlo Tree Search algorithm simulates all the possible outcomes of different decisions. It chooses randomly which action to simulate. Our implementation of MCTS uses the exploration and exploitation of different game states to decide on moves with the upper confidence bound formula. The MCTS is initialized with its player number and its time limit of 5 seconds. On its turn it takes in a copy of the current board state, and begins evaluating moves. It gets all of the possible moves it could play, which is a max of 7 and begins simulating games from there, keeping track of visits, wins, and losses of expanded child nodes. When MCTS is done searching it returns its best UCB scored move. The UCB score is exploitation + exploration. Exploitation is the win rate of the child node or move (total wins / total visits). Exploration ensures that less visited nodes get visited (sqrt(total parent visits / total visits)).

To test the performance of Minimax and MCTS we simulated 125 games, then switched their positions and simulated 125 more games. To accelerate testing and allow for practical applications, we imposed a 5-second restriction on the decision making process for both AI's.

Results

After implementing the Minimax and MCTS algorithm, we pitted the AI agents against each other; with player 1 using MCTS and player 2 using Minimax. During data collection, we noticed that both algorithms were making obvious mistakes that would lead to a loss. After some troubleshooting, we discovered that both algorithms were making moves that were optimal but had ignored potential losses. We revised both the Minimax and MCTS algorithms for a second round of tests.

In the first trial, we set player one to use Minimax and player two to use MCTS. This procedure would more accurately test the hypothesis, as the objective is to demonstrate perfect play from player 1. In 125 games, player 1 achieved a win rate of 96.65 % including four losses to player 2. There were no draws in testing.

In the second trial we set player one to use MCTS and player two to use Minimax. In 125 games player 1 achieved a win rate of 27.2 % with 91 losses. There were no draws.

Trial 1

	Games Won	Games Lost	Moves taken (Average)	First move middle Column (Percent)	Win Rate
Player 1: MiniMax	121	4	34	100%	96.8 %
Player 2: MCTS	4	121	34	100%	3.2 %

Trial 2

	Games Won	Games Lost	Moves taken (Average)	First move middle Column (Percent)	Win Rate
Player 1: MCTS	34		26	68 %	27.2 %
Player 2: Minimax	91		26	68 %	72.8 %

Conclusions and Future Work

The results of the experiment clearly show that the minimax algorithm greatly outperforms the MCTS. In all of the metrics that we outlined for perfect play. The results also show that neither algorithm reaches absolute perfect play. Minimax did consistently start with the middle column, and won almost nearly every game as player one. Games also lasted much longer with Minimax as player one, with a higher average move count. With MCTS as player one in trial two, the switch in positions shows clearly that minimax performs better. MCTS's win rate does jump from 3.2% to 27.2%. Ideally it would have a much higher increase. MCTS also did not always start with the center column. The average round length decreased as well. These results do support the facts of Connect Four, like the middle column being the best starter move, and that player one starts in a much stronger position.

Our program allows users to play against the AI's. Playing both of the AI models had very interesting results. Both of the AI's, even MCTS with 5 seconds of time, are more than capable of beating human opponents at least on occasion. An observation we had in the difference between humans playing connect four and robots playing it is that, when playing people, it's important to be subtle in how you connect dots. If you just place three dots in a rot turn after turn. It's going to get blocked. Most games are won against people by a surprise connect four they didn't see coming. When the AI's play they don't play with subtlety, they don't need to when battling other AI because they do not get surprised. When the Monte Carlo plays it tries to connect 4 in a really obvious attempt then gets blocked repeatedly. The Minimax does the same, but usually stops just short of its goal, before putting itself in a position where the other player would simply block it. It instead then starts creating threats in other places, blocking the opponent, or just fills up the board until the other player is forced to play a move that lets it win. This behavior is close to our goal of a perfect AI because as player one it should be forcing the wins from not just trying its best to win. One of the biggest flaws with the MCTS is that it does not simulate moves with knowledge that the opponent is going to play perfectly, it plays randomly until it gets to terminal states, then adds these up. Both the AI's were capped at 5 seconds each in the trials. In player vs AI testing, the MCTS algorithm in particular performed much better with an increased time limit. We chose five seconds for each algorithm in an attempt to level the playing field. Though this is not exactly fair as the MCTS requires a lot more time than the Minimax. Future testing could be done at different time allotments for the algorithms to see if given enough time, MCTS could outperform Minimax.

In conclusion our findings show that the Minimax algorithm performs much better and beats the Monte Carlo Tree Search algorithm in the game of Connect Four nearly every time. We found that the Minimax algorithm gets close to performing perfectly under the criteria that it always plays the middle column first as player one, always wins as player one, and is capable of beating human opponents. We also found that the Monte Carlo algorithm does not perform as close to perfect in tight time constraints in five seconds. But its performance does improve with more time allowed to it. These results highlight the strengths of deterministic approaches in structured environments like Connect Four.

Works Cited

- Wilkens, Isak, and Jean El Khoury. *Minimax vs. Monte Carlo: A Comparative Case Study on the Use of Minimax with Alpha-Beta Pruning versus Monte Carlo Tree Search as Decision-Making Algorithms in an Android Chess Application.* Undergraduate degree project. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, 2024. Supervisor: Fredrik Kilander, Examiner: Niharika Gauraha, Host company: Apex Consulting AB.
- Baier, Hendrik, and Mark Winands. *IEEE TRANSACTIONS on COMPUTATIONAL INTELLIGENCE and AI in GAMES 1 MCTS-Minimax Hybrids*.
- asun.net (2013). Four in a Row Solver [computer software]. Microsoft Corporation.