AI 531 : PROJECT
CONNECT 4

Sai Bhaskar 934-480-538
Sundareswar Pullela 934-452-706
Harsha Chaitanya N 934-459-905

Abstract

Connect 4 is a two-player game in which the players choose a color out of two (usually red and yellow) and drop that coloured pieces in a 6X7 grid. Each move of a player is considered as dropping a piece in a column. In this assignment, we have assigned a heuristic to each player(both players are assumed by AI agents) and compared them with our heuristic to understand which one works better.

Introduction

Connect 4 is a game between two people. It has two sets of 21 identical pieces. Usually one set is in color red and the other is in color yellow. The players play the game on a vertical board which has 6 rows and 7 columns. A move of any player is considered as dropping a piece in any column that has space for the piece, and the piece occupies the lowest available space. No piece can be dropped on a column that has 6 pieces. Both the players play their moves in turns. There is no specific rule as to which player should start first.

Both the players try to win the game by connecting their four pieces either vertically or horizontally, or diagonally. The first player to connect the pieces wins. If all the 21 pieces of both players are used and no player is able to connect their four pieces then that state is considered to be a draw.

Connect 4 is one of the zero-sum games. Hence, one of the common heuristics to use for this game is minimax. In order to measure its efficiency we have implemented a couple of other heuristics. One of them is the Monte Carlo Tree Search algorithm and the other is a custom heuristic which decides a move based on the density of the pieces.

Zero-sum game

A zero-sum game is defined as a game where the utility to all the players of the game is the same for each instance of the game.

Example: Chess is also a zero-sum game as it has the playoffs as 0,1, or ½.

Minimax Algorithm

The minimax algorithm is one of the frequently used algorithms in two-player games. The players are called MAX and MIN where MAX moves first and then the players take turns until the game is finished. At the end of the game, the player who wins will be awarded with points and the loser is given penalties.

A game can be defined using the following elements:

- S_0: An initial state, that describes the initial set up at the start
- *Player(s)*: Describes which player has to move in the current state
- *Actions(s)*: It is a set of all the legal moves made in the game
- *Terminal-Test(s)*: States whether the game ended or not. All terminal states return true for *Terminal-Test.*
- *Utility(s,p)*: A utility function is the one which describes the terminal state s for a player p in terms of a numeric value.

The minimax algorithm calculates the decision from the current state of a game. It uses a recursive computation function of minimax values of each successor state. This recursion is carried out till the leaves of the tree and the values are backed up using the tree as the recursion unwinds.

The minimax algorithm is a depth-first search of the game tree. For a game tree whose depth is *m* and the total number of legal moves are *b*, the time complexity is $O(b^m)$ and space complexity is $O(m)$.

Alpha-Beta Pruning

We can reduce the complexity of the minimax search by pruning large parts of the tree from consideration and compute the minimax decision. One of the ways to achieve the pruning is by implementing Alpha-Beta Pruning. It eliminates the branches that do not affect the final decision. As the name suggests, there are two parameters: Alpha and beta, that describe the bounds on the backed-up values that appear anywhere in the path.

α = The highest-value choice we have found in the path for MAX.

β = The lowest- value choice we have found in the path for MIN

The search updates the values of α and β, as it traverses along the tree and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the values of α or β with respect to MAX or MIN.

Algorithm:

function ALPHA-BETA-SEARCH(state) **returns** an action

v ←MAX-VALUE(state,−∞,+∞)

return the action in ACTIONS(state) with value v

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        If v ≥ β then return v
        α←MAX(α, v)
    return v
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        If v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Monte Carlo Tree Search Algorithm

MCTS is a method to find optimal decisions for a given domain by considering samples at random in domain space and constructing a search tree based on the results. MCTS builds a search tree iteratively until a constraint based on time, memory or iterations is reached. The search then halts and returns the best performing root action. Each node in the tree describes a state of domain, and the links to the successor nodes describe the actions that lead to that respective state.

MCTS applies four steps at each iteration:

- *Selection*: Right from the root node, a child selection policy is applied to traverse through the tree till the most urgent expandable node is reached. (expandable node is the one which is a non-terminal state and it is not visited yet.)

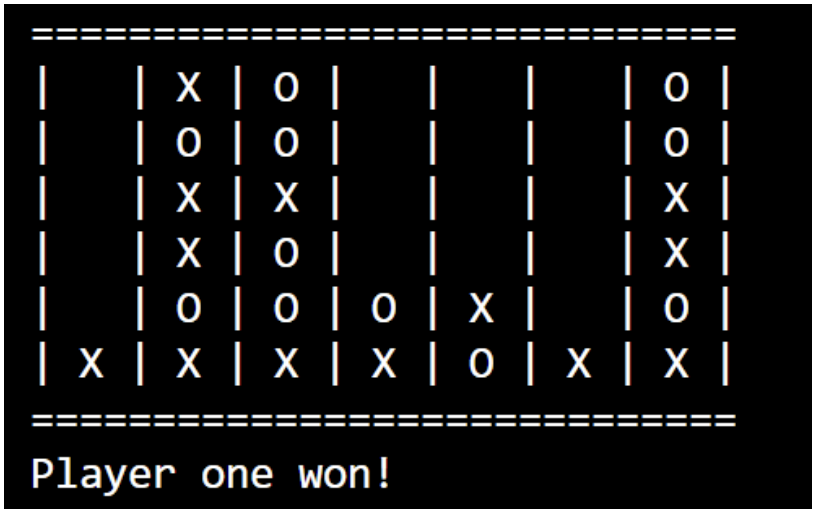
- *Expansion*: Child nodes are added to expand the tree, with respect to the available actions.
- *Simulation*: It is a run from the new node that obeys the default policy to produce an outcome.
- *Backpropagation*: The result simulated, is backed up through the selected nodes.

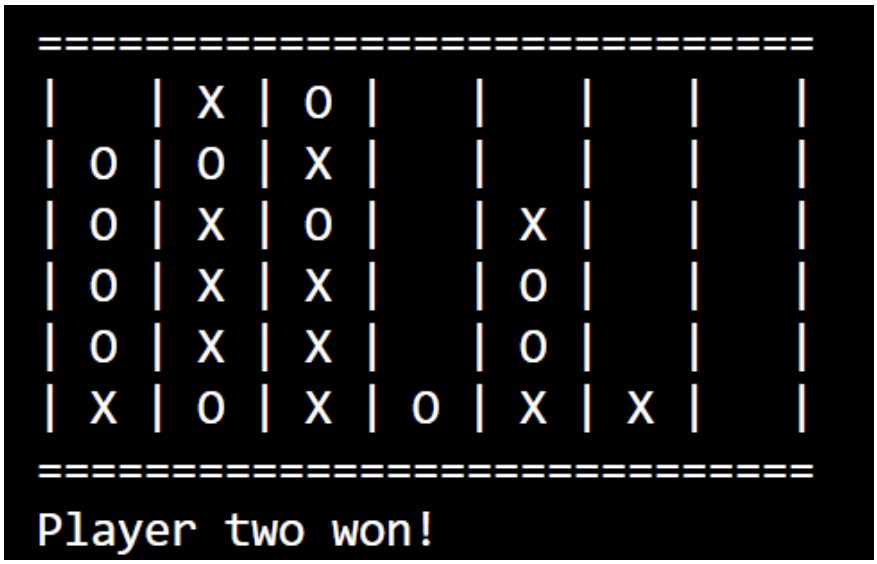
Custom Heuristic

The custom heuristic we developed for this experiment is the one which decides a player's move based on the density of the current player's pieces present in all the 4x4 sub-grids and selects the one that is hugely populated with the respected piece.

Results

In our project, we aimed to explore the effectiveness of three different approaches to playing the game Connect4: a custom heuristic function, Minimax and the Monte Carlo Tree Search (MCTS) algorithm. As mentioned above, The custom heuristic function is based on the density of similar coins on the board, with a strategy of dropping a coin randomly and then attempting to score by placing additional coins close to the initial position.





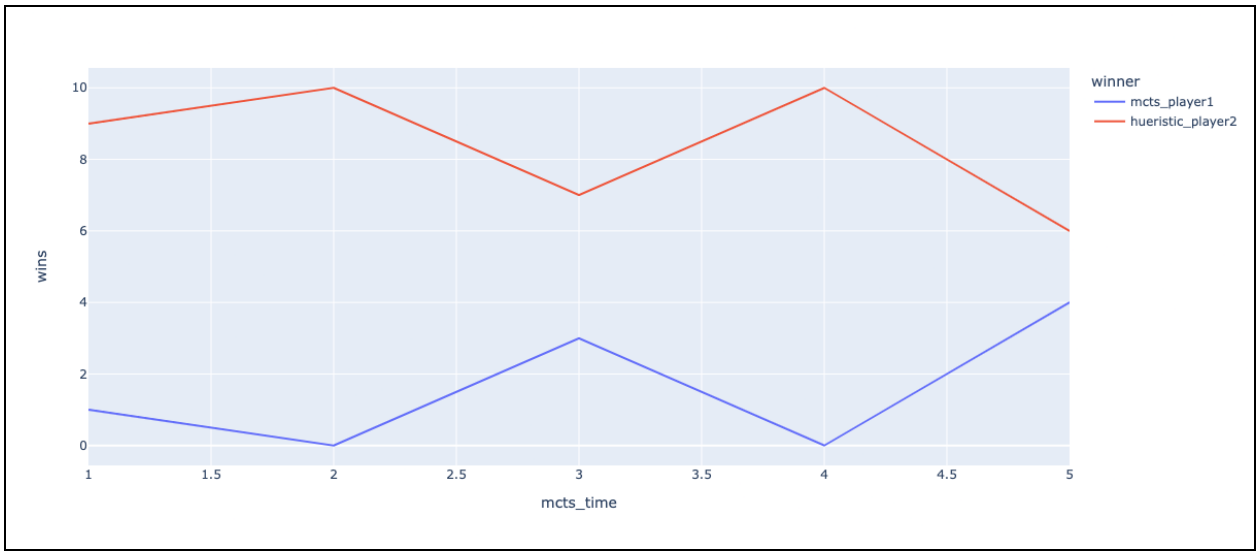
The above pictures represent a scenario where player one and player 2 wins respectively. X represents player 1 while the O represents player 2.

Few of the limitations were considered to reduce computational complexity :

- maximum number of seconds allotted for the MCTS is 5 seconds
- the maximum depth considered for minimax algorithm is 5

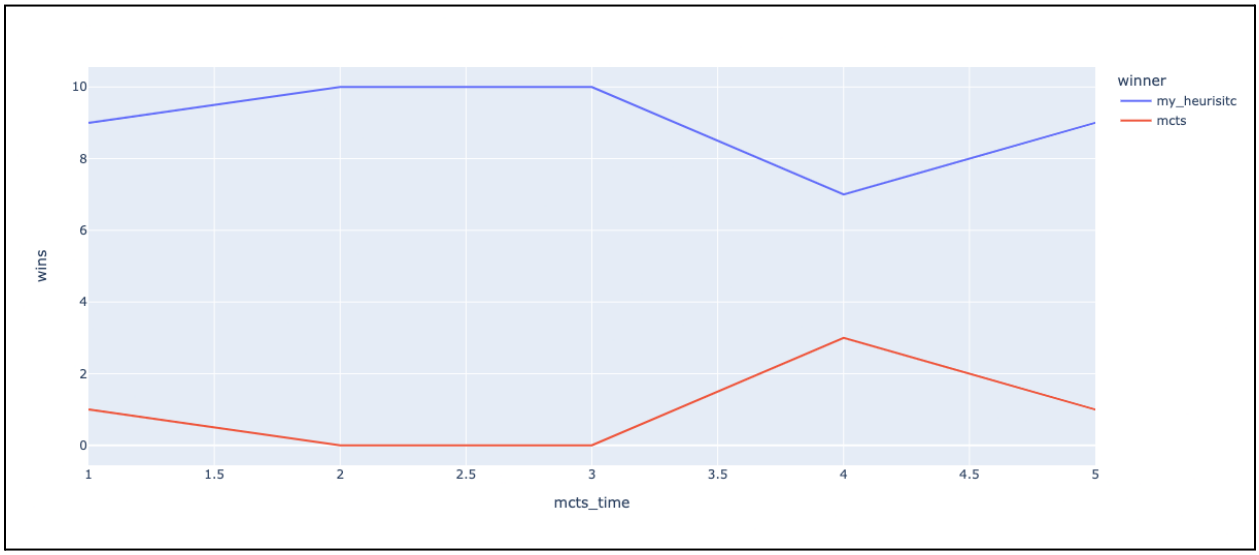
Here is the experimentation and evaluation process

- We plotted on a line graph, with the depth of the algorithm in the case of minimax algorithm and number of seconds in the case of MCTS algorithm on the x-axis while the number of wins are shown on the y-axis.
- We get four different line graphs
    - Player 1 MCTS vs Player 2 custom heuristic
    - Player 1 custom heuristic vs Player 2 MCTS
    - Player 1 Minimax vs Player 2 custom heuristic
    - Player 1 custom heuristic vs Player 2 Minimax



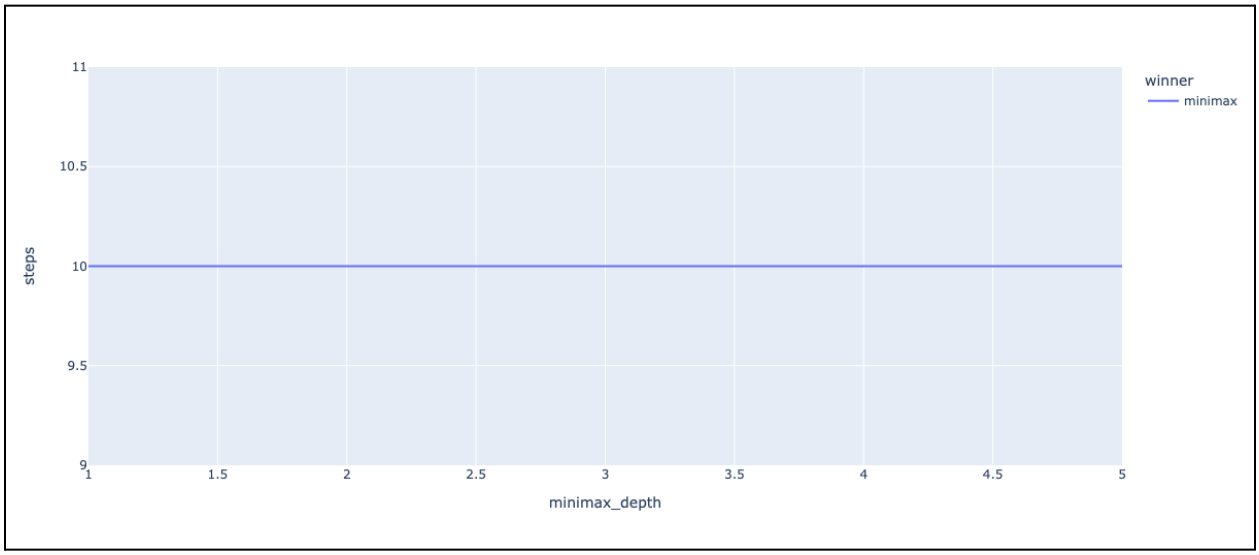
MCTS vs Custom Heuristic

In this scenario, Player1 is MCTS and Player 2 is the custom heuristic. For every value of mcts_time(in secs), 10 rounds of testing was performed. We can observe that the custom heuristic working on the basis of density performs better than the MCTS approach in every phase i.e MCTS with 1,2,3,4 and 5 seconds limit. We can also observe that MCTS with 5 second limit tries to catch up with the heuristic with the maximum wins i.e 4 out of 10.



Custom Heuristic vs MCTS

In this case, Player1 is Custom heuristic and Player 2 is the MCTS. We can observe that the custom heuristic still overperforms the MCTS approach in every phase. Considering number of wins as the metric for evaluation, we can observe that MCTS as player 1 has more than MCTS as player 2, which proves the strategy of starting first in connect4 is more beneficial.



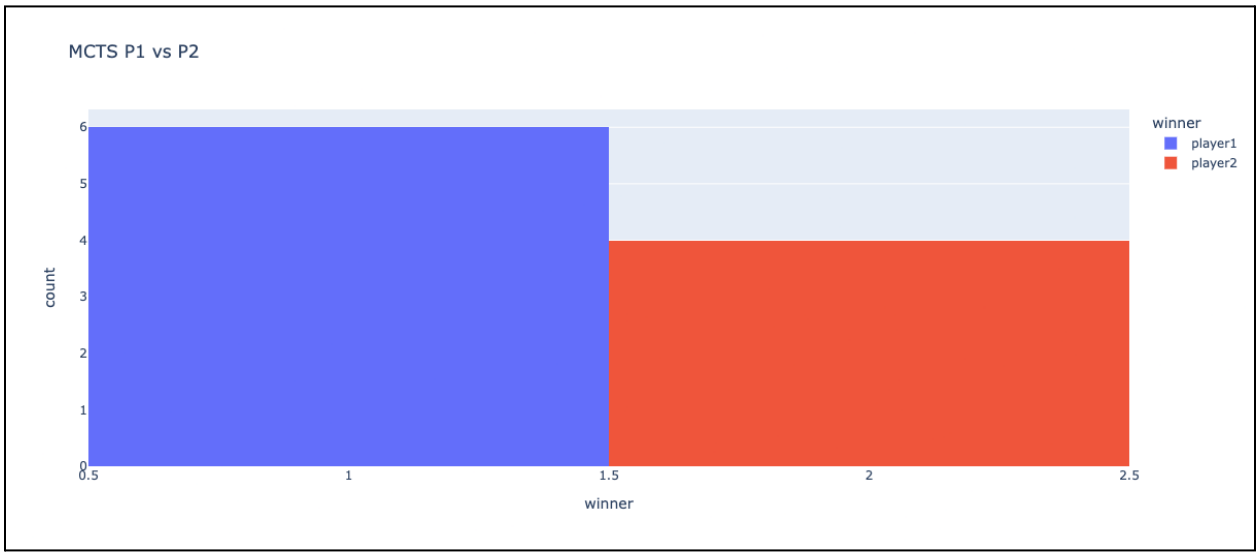
Custom Heuristic vs Minimax (vice-versa)

The above graph depicts that Minimax overperforms over the custom heuristic in every phase both as player 1 and player 2(reason for adding viceversa in the graph).

Lets look at the major observations and few possible reasons for the results we have observed so far. The five second time limit is a constraint that can adversely affect the Monte Carlo Tree Search (MCTS) algorithm. When the time limit is short, the algorithm may not be able to explore the decision tree fully, and as a result, it may miss out on potentially good moves or paths. The effectiveness of the MCTS algorithm is directly proportional to the amount of time it is allowed to search and explore the decision tree. Therefore, a shorter time limit can have a negative impact on the algorithm's performance.

In contrast, a custom heuristic can be improved upon by adding more rules to it. A heuristic is a problem-solving technique that uses a practical approach, rather than a

theoretical one, to solve a problem. In the case of a game AI, a custom heuristic can be designed to follow specific beneficial strategies, such as starting from the center and blocking opponents from stacking. By adding more rules to the heuristic, it becomes more intelligent and can make better decisions. However, when compared to the minimax algorithm with a depth of five, the custom heuristic and the MCTS algorithm with a time limit of five seconds are less powerful. By exploring the decision tree to a depth of five, the minimax algorithm can identify the best possible move or path to take, making it more effective than the custom heuristic and MCTS algorithms in this scenario.



One more key observation can be made from the above graph. Player 1 (player starting the game) has an upper hand when compared to player2. The above graph is MCTS vs MCTS and we can observe that player1 has won 6 times while player2 has won 4 times. In the case of Minimax vs Minimax, we observed that 10 out of 10 times, player 1 has won the game.

In conclusion, while a custom heuristic can be improved upon by adding more rules, and the MCTS algorithm can be effective with more time, the minimax algorithm with a depth of five is the most powerful option in this context.