Table of Contents

Introduction

Connect Four is a two player classic strategy game. It is played on a 6*7 board between two players. Each player drops a checker or a piece down any of the available slots. The goal of the game is each player targets to build a row of four checkers preventing their opponent from doing the same.

Game Playing Rules:

1. The player to go first is decided.
2. Each player puts one piece in one of the columns which falls down under the force of gravity to the lowest unoccupied space in the column.
3. On each of their turn, the player drops one of their checkers down any of the slots from the top of the grid.
4. The first player to get four of their checkers in a row wins the game. The four in a row can be horizontal, vertical or diagonal.
5. As soon as the a column contains 6 pieces, no other piece can be put in the column
6. If the board fills up with all 42 checkers without four in a row, the game is tied.

Objectives:

The objective of the game is to be the first player to get four of your colored checkers in a row horizontally, vertically or diagonally.

Agent Description:

The bot is a goal based agent which takes decision based on how far they are currently from their goal. Each of its action is intended to reduce its distance from the goal. This allows it to choose among the multiple possibilities, selecting the one which reaches a goal state.

Performance Measure:

i. Put four checkers adjacently in a row, column or diagonal to each other.
ii. Prevent the opponent from doing the same.

Environment:

i. Game Board
ii. AI's and Player's Checker

Actuators:

Game Playing Function, Minimax Function

Sensors:

Knowing the state of the board

Agent Environment:

The following table shows the different agent environments with their property.

Environment Property	Description
Fully Observable vs Partially Observable	Connect4 has a **fully observable** environment consisting of the board, checkers and the grids. The agent has full access of the information of the environment so it has the complete knowledge of the board. It knows the state of the system at each step to make the optimal decision.
Deterministic vs Stochastic	The environment is **deterministic** as there is no uncertainty in the game. There are no random elements at work. The agent knows with certainty the next state of the environment and the resulting state doesn't change given the same state and action at all times.
Episodic vs Sequential	The environment is **sequential** as the past history of actions in the game affects the current decision which further affects all the future decisions.
Static vs Dynamic	It has a **fully static** environment as time is not a factor in making the decision of where to place the checker. Once it is agent's turn, the state doesn't change until it makes its move. Since time is not a factor for decision making, there is no penalty to the agent.
Discrete vs Continuous	The environment is **discrete** as there are a finite number of actions to be performed. At every step, the agent and its opponent have 7 possible actions of which it makes to choose the optimal one. There is also a finite set of possibilities to win among the large set of actions.
Single vs Multi-Agent	The environment is **multi-agent** containing two agents, an AI agent and a Human Player agent. Both agents target to maximize their own performance measure and minimize their opponent's.

Problem Specification

Problem:

Given an empty grid of size 6*7, get four of the colored checkers in a row.

Goal:

Goal is reached when one of the two competing players build a row of four checkers horizontally, vertically or diagonally first.

Constraints:

No checker can be placed below the existing checker.

State Space Representation

Any board game has a well-defined state or position at all times. The state refers to the all possible arrangement of different checkers or pieces together to describe the state of the game. When a move is made, the current state changes to another. The starting position of a game is initial state (Checkers=0 in the table) which is an empty board for Connect-4. The figure below shows the possible states for first checker at the center column.

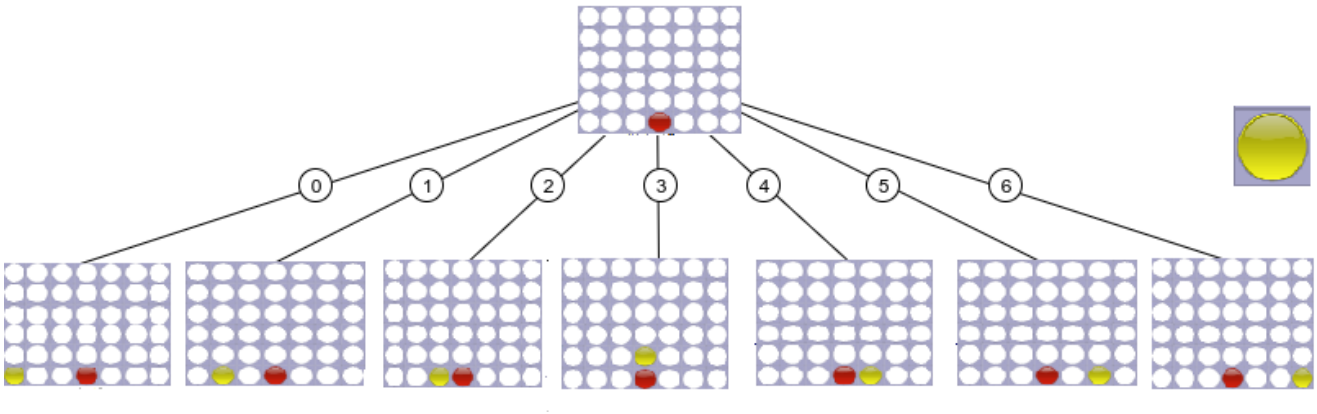

Fig: First state (Initial move at center column) and its possible states

The following table consists of the number of leaf nodes and the state space complexity for a given number of checkers.

Checkers	Leaf Nodes	State-Space Complexity
0	1	1
1	7	8
2	49	57
3	238	295
4	1,120	1,415
5	4,268	5,678
6	16,422	22,100
7	53,955	76,055
8	181,597	257,652
9	534,085	791,737
10	1,602,480	2,394,217
. . .	. . .	. . .
42	140,728,569,039	4,531,985,219,092

Connect-4 has a medium state space complexity calculated to be 4,531,985,219,092. The logarithmic (log 10) state-space complexity is 14 and game-tree complexity of approximately 10^{21}. Since the state-space complexity for this game is too high, it is very difficult to be solved by simply enumerating all the legal positions starting from the initial space. The figure below shows the moves made by Min and Max players.

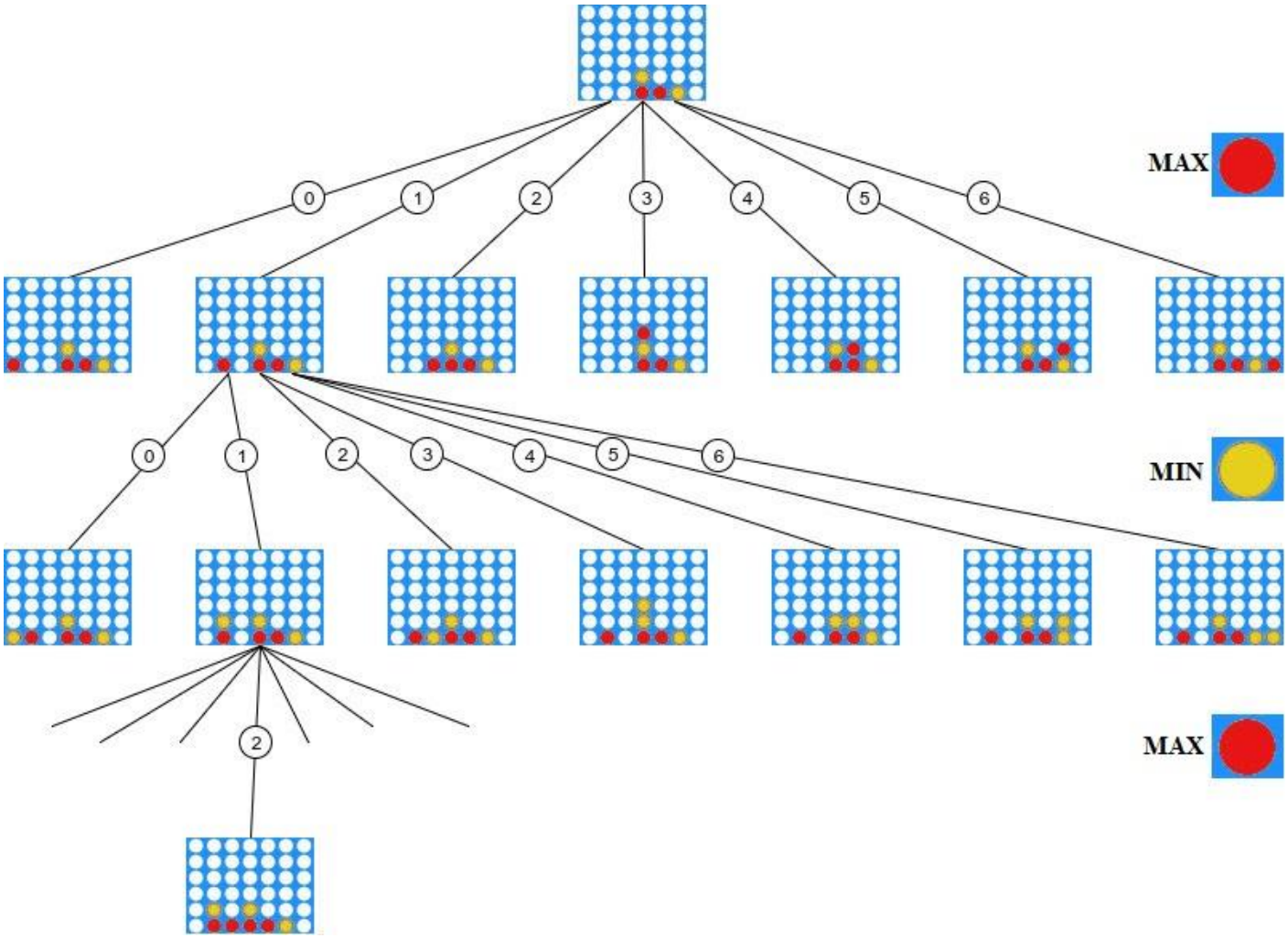

Fig: A small portion of the state space tree.

Algorithm Used/ Technical Description

The mini-max with alpha beta pruning is used for the solution of this game. *Mini-Max Algorithm* falls in the category of backtracking algorithm. It is used in decision making and game theory to find the optimal move for a player, assuming that opponent also plays optimally.

The way the algorithm works is that there are two players, max and min. We typically view the game from max's perspective which goes first. The max tries to maximize its move while the min tries to minimize the moves made by max. There is a value associated with every board state. For a given state if the max has upper hand then the score of the board will tend to be some positive value whereas, if the min has upper hand in that board state then it will tend to be some negative value. Heuristic function is used to calculate the values of the board.

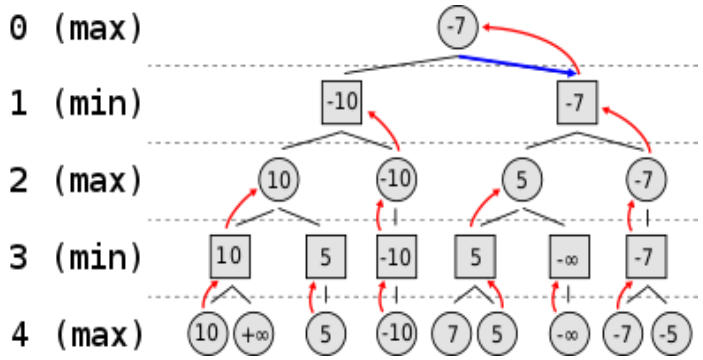

Fig: Min-Max Algorithm

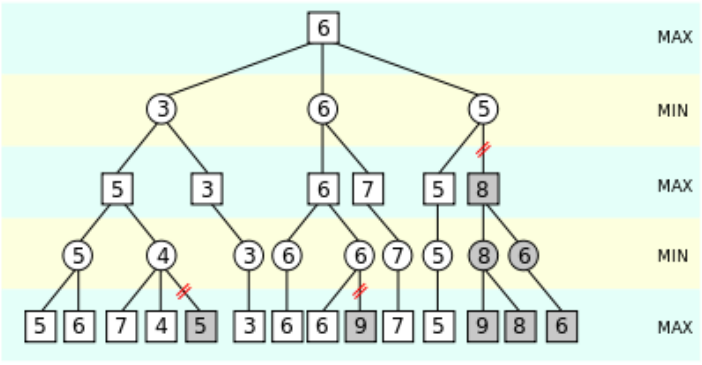

Fig: Min-Max Algorithm with Alpha Beta Pruning

Alpha-Beta Pruning is used to optimize the mini-max algorithm. When mini-max with alpha beta pruning is used instead of simple mini-max algorithm then less number of nodes is evaluated in the game tree. It optimizes tshe mini-max algorithm by bringing down the computation time by performing less iteration. In min-max algorithm, two extra parameters are passed, named alpha and beta which will make searching the game tree much faster and to much more depth. Game tree has many branches leading to different game states. The benefit of using alpha-beta pruning is that if there already exists a better move, then other branches need not be searched and can be pruned saving computation time. At any node in the game tree the maximizing player is assured of a maximum score which is stored by alpha and similarity the minimizing player is assured of a minimum value which is stored by beta. These two values namely alpha and beta are updated and maintained by the algorithm. Initially both players begin with worst possible values they can score. Therefore, alpha is allotted a value of minus infinity and beta is allotted a value of plus infinity.

Since minimax is exhaustive (brute-force) and the search space for Connect-4 is very large, we limit the depth to which max can look ahead. We can speed up the algorithm with alpha-beta

pruning. User plays against the computer (AI) in which mini-max algorithm is used to generate the game states and find the optimal solution.

Heuristics

Heuristic function is used in Minimax with alphabeta pruning for evaluation of the current situation of the game. Designing a reasonable heuristics function is very important. The heuristic function implemented in this project does the following task.

i. First, it evaluates the current situation of the game.
ii. One of the two players wins the game. The winner has absolute advantage over the game. Therefore, infinite value is given to indicate the winner's absolute win.
iii. The occasion when a player has three pieces in a row, 100 points is provided because there is a possibility to win.
iv. The occasion when a player has two pieces in a row, 5 points is provided (lesser than three pieces in a row).
v. The occasion when a player has no piece adjacently connected to any other piece.
vi. If the piece is close to middle column and row, it has a bigger value as there are more possibilities to win the game.

Heuristic Function is implemented by AI to exhaustively evaluate the tree to a certain depth or number of moves to think ahead, and calculate the value of the board depending on the placement of pieces on the board. The score_position function runs on all the columns that a move can be made and the pick_best_move function picks the highest score that is returned.

Outcome Analysis

It is observed that when the user plays the game in single player mode and chooses minimax, AI takes 2799 iterations to generate the game state and computation time of 33 millisecond for a depth 4 while it takes only 477 iterations to generate the game state and computation of 6 millisecond with alpha-beta pruning.

Algorithm Used	MiniMax		Alpha-Beta Pruning	
Depth	No of iterations performed	Computation Time(ms)	No of iterations performed	Computation Time(ms)
1	7	0.03	7	0.008
4	2799	33	477	6
8	5847005	55441	71773	1009

Conclusion

Connect-4 has been implemented using two algorithms i.e. minimax and minimax with alpha-beta pruning. Then, comparison is made based on computation time with depth. It has been found that for the same depth, the two algorithms behave very differently. In terms of number of iterations performed and time taken, alpha-beta pruning takes much less time and performs very few iterations than mini-max.