**Mehak Gupta**

**301311972**

**ASSIGNMENT 5 REPORT (CONNECT 4 USING AI)**

**Q1**. Estimate of how many playouts per second your program can do as a measure of its performance.

A . The program can do around 6000-7000 playouts for every legal move so about 7\*6000 = **42000** **playouts** under a second.

**Q2**. Explain your implementation.

The program is divided into 2 classes namely Board and Connect4. The board class consists of the methods that are used to identify legal moves, play or undo a move and display board after every change. The Connect4 is the main class which takes board as an argument. Following is the role of some main functionality methods in class Connect4 :-

**gameResult**(Board b) - The method takes the board and checks if there is a winner at the present stage of the game. It returns 1 if Heuristic Monte Carlo wins, 2 if Pure Monte Carlo wins ,0 if draw and -1 otherwise.

**pMCTS**() – The method represents the pure monte carlo tree search and records score for the random playouts for every move.

* random\_playout() - helper function used by pMCTS() to simulate random playouts for every legal move.

**hMCTS**() – The method represents Heuristic Monte Carlo and implements Minimax method and returns the next chosen move

**minimax**(int depth, int turn, int alpha, int beta) – The method allows the program to look ahead at the future positions before deciding what move it wants to make in the current position. The method continues to expand the moves until either we reach the end of the game or we decide to stop by setting a max depth because going deeper would take too much time. In the end of the tree the program performs a static evaluation on the final positions to estimate how good the position is for one side without actually making any moves. The Heuristic player is always trying to maximize the evaluation and the other player is trying to minimize the evaluation. The method also uses alpha beta pruning which essentially gives the same result as Minimax but saves some time by not considering positions when they can’t affect the outcome.

* evaluateBoard(Board b) is a helper function used for the static evaluation to decide for the better moves.
* CalculateScore(int aiScore, int moreMoves) is a helper function that calculates score based on how far is the player from the win. For example :- if the player already has 3 in a row, the score will be quite high as player is very close to the win.

**Q3.** How well are both version of the game playing

After a lot of manual testing, what I have noted is the pure monte carlo tree search ALWAYS WINS against a human player and never loses or draws the game even with 50 playouts per every legal move, so it was really smart to play against humans. But the Heuristic monte carlo tree search is definitely better than pure monte carlo as it makes more intelligent moves. To confirm the behavior, I performed 40 games between pure and heuristic monte carlo search.

**RESULTS**

Heuristic version plays first – Heuristic: 17/20 Pure: 3/20

Pure version plays first - Heuristic: 16/20 Pure: 4/20

**Q4**. The extra techniques used by the second version of the code

The second version of the code is using Minimax alpha beta pruning

**Q5**. A comparison of which program is stronger

Version 2 is definitely stronger than the random mcts irrespective of who makes the first move

**Graphs and Tables :-**

|  |  |  |  |
| --- | --- | --- | --- |
| Total Games Played | 100 |  |  |
|  |  |  |  |
|  | Win | Draw | Loss |
| Pure MCTS | 10 | 8 | 82 |
| Heuristic MCTS | 82 | 8 | 10 |
|  |  |  |  |