a)

Evaluation Function :

I came up with 4 different evaluation functions. The best I could come up with was the evaluation function 4.

Explanation :

An evaluation function for a game like Connect 4 should consider both the immediate tactical positions (like winning, blocking winning moves) and strategic positions that lead to future opportunities (like setting up two potential winning moves). The evaluation function might sometimes take more number of moves to play strategically.

Central Control:

Pieces in the center column are generally more valuable because they participate in more potential winning combinations.

Potential Connects:

Not just the immediate threat but also positions that allow rapid connection of pieces.

Flexibility:

The more places you can play and still have a potential to connect four, the better your position is.

Result : lookahead depth = 3 , ranking evaluation function from best to worst

Evaluation function 4 :

```
On running 50 Games the results are
Average number of times player 2 win = 72.00 %
Average number of moves moves player 2 win = 19.11
Total execution time: 94.04 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 3 :

```
On running 50 Games the results are
Average number of times player 2 win = 62.00 %
Average number of moves moves player 2 win = 21.10
Total execution time: 73.91 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 2 :

```
On running 50 Games the results are
Average number of times player 2 win = 54.00 %
Average number of moves moves player 2 win = 17.48
Total execution time: 53.13 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 1 :

```
On running 50 Games the results are
Average number of times player 2 win = 38.00 %
Average number of moves moves player 2 win = 23.79
Total execution time: 18.00 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

b) Comparing the time taken by all evaluation functions with alpha-beta pruning with depth 3

```
On running 50 Games the results are
Average number of times player 2 win = 68.00 %
```

Evaluation function 4 :

Evaluation function 3 :

```
On running 50 Games the results are
Average number of times player 2 win = 58.00 %
Average number of moves moves player 2 win = 20.97
Total execution time: 28.64 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 2 :

```
On running 50 Games the results are
Average number of times player 2 win = 58.00 %
Average number of moves moves player 2 win = 17.03
Total execution time: 20.07 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 1 :

```
On running 50 Games the results are
Average number of times player 2 win = 38.00 %
Average number of moves moves player 2 win = 23.37
Total execution time: 4.38 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Comparing the time taken by all evaluation functions with alpha-beta pruning with depth 5

Evaluation function 4 :

```
On running 50 Games the results are
Average number of times player 2 win = 88.00 %
Average number of moves moves player 2 win = 22.05
Total execution time: 355.47 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 3 :

```
On running 50 Games the results are
Average number of times player 2 win = 70.00 %
Average number of moves moves player 2 win = 15.09
Total execution time: 270.38 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 2 :

```
On running 50 Games the results are
Average number of times player 2 win = 70.00 %
Average number of moves moves player 2 win = 18.40
Total execution time: 442.28 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Evaluation function 1 :

```
On running 50 Games the results are
Average number of times player 2 win = 50.00 %
Average number of moves moves player 2 win = 23.84
Total execution time: 33.78 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

When the Game Tree-based program initially looks only 3 moves ahead with alpha-beta pruning, here's what can be observed:

Speed: The program will run faster due to the shallower depth, as the number of game states to evaluate grows exponentially with each additional ply (half-move) in depth. Alpha-beta pruning will also help reduce the number of evaluated nodes, further speeding up the search.

Play Quality: The play quality will be decent but not optimal. The program might miss strategies that require more foresight than 3 moves ahead, and it may not be able to defend against or set up more complex traps.

When I increase the depth of the game tree to 5, the following observations are:

Slower Performance: The computation time will increase, potentially significantly, due to the deeper search. However, alpha-beta pruning will continue to mitigate this by eliminating the need to evaluate branches of the search tree that cannot affect the final decision.

Improved Play Quality: The program's ability to identify and execute longer-term strategies will improve. It will be better at both setting up future wins and avoiding pitfalls that weren't visible with a shallower search depth.

Better Evaluation: With a deeper search, the quality of the evaluation function becomes more important. Good evaluation functions will lead to stronger play, as the program can more accurately assess positions that are several moves away.

Increased Wins and Possibly Fewer Moves: With better foresight, the program may win more often because it can foresee and avoid potential losses, or set up wins that are more than 3 moves away. Depending on the play style encouraged by the evaluation function, the program might also win in fewer moves by identifying the quickest path to victory.

c) Depth 5 : Move Order 1 : [6 ,5, 4, 3, 2, 1, 0]

Result :

```
On running 50 Games the results are
Average number of times player 2 win = 76.00 %
Average number of moves moves player 2 win = 19.79
Total execution time: 727.97 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Depth 5 : Move Order 2 : [3, 2, 4, 1, 5, 0, 6]

Result :

```
On running 50 Games the results are
Average number of times player 2 win = 78.00 %
Average number of moves moves player 2 win = 19.33
Total execution time: 435.35 seconds
PS D:\BITS\College\Fourth year\4-1\AI\assignment2>
```

Move Ordering 1 (Reverse Order):

With Move_order_1 = [6, 5, 4, 3, 2, 1, 0], the algorithm starts evaluating moves from the rightmost column to the leftmost. This might not be as efficient as evaluating moves from the center because, in Connect 4, moves in the center columns can often lead to more opportunities for making connections. Therefore, this move ordering may not reduce the number of recursive calls or the average game duration as effectively as center-oriented heuristics.

Move Ordering 2 (Center-Oriented Order):

With Move_order_2 = [3, 2, 4, 1, 5, 0, 6], the moves are evaluated starting from the center column and moving outward. Since Connect 4 is a game where central control is often advantageous (as the

center column participates in the greatest number of possible 4-in-a-rows), this heuristic is likely to be more effective. It will often lead to earlier alpha-beta cutoffs and reduce the total number of recursive function calls and the average time needed to beat the Myopic player.

In summary, while both heuristics can potentially reduce computation compared to no move ordering at all, the center-oriented heuristic (Move_order_2) is more likely to lead to a greater reduction in recursive calls and game duration than the reverse order (Move_order_1). This is because it aligns better with the winning strategies of Connect 4, where central positions are generally more valuable.

d) Increasing the cut-off depth in a minimax algorithm (from 3 to 5) results in a stronger playing agent for a few reasons:

Deeper Lookahead: A greater depth allows the algorithm to see further into the future, which means it can better understand the consequences of its moves. This often leads to more strategic play, as the agent can plan several moves ahead rather than making decisions based solely on the immediate board state.

Better Decision Making: With a deeper search, the agent can evaluate more potential game states, including those that may not be immediately obvious. This can help it identify opportunities for winning and threats from the opponent that it would otherwise miss with a shallower search.

Improved Heuristics: If the evaluation function provides a good heuristic for non-terminal states, a deeper search will make better use of those heuristics, leading to more accurate assessments of the game state.

However, there are also trade-offs to consider:

Diminishing Returns: There's a point at which increasing the depth yields minimal improvements in play quality, especially if the evaluation function isn't sophisticated enough to make fine distinctions between similar board states.

Regarding the frequency of winning and the number of moves before a win:

Frequency of Winning: It is likely to increase because the agent can make better-informed decisions, potentially outmaneuvering an opponent who is using a shallower search or less sophisticated strategies.

Average Number of Moves Before a Win: This might decrease or increase. A more strategic agent could win more efficiently (hence in fewer moves) by setting up traps and forced wins. However, if the agent becomes too conservative or overvalues the board positions that don't lead directly to a win, it might miss quicker winning opportunities that a more aggressive strategy would catch.

Conclusion: In some cases, deeper search depth will dramatically improve performance, while in others, the improvements may be marginal.