Report on Connect 4 Artificial Intelligence Assignment 2

Ishan Nerlekar

2020B3A71515G

In this report, I have analyzed the implementation and performance of our AI system in playing the game Connect 4. The report provides a detailed overview of the assignment, including the objectives, methodology, and results.

We discuss the various algorithms and techniques used to develop the AI system, such as minimax with alpha-beta pruning, evaluation functions, and move selection strategies.

The report also includes a thorough evaluation of the AI system's performance, including measures such as win rates, search depths, and computational efficiency. We compare the performance of our AI system against a Myopic Player and assess its strengths and weaknesses.

Question 1

Part A

I have used a game tree depth of 5 for the following analyses.

Evaluation Heuristic 1:

The computer calculates game state utility in the following manner:

- Calculates the total number of consecutive streaks of 4 of its own color. Since streaks of 4 win the game, we assign a weight of 10 to these.
- Calculates the total number of consecutive streaks of 3 of its own color. We assign a weight of 5 to these, since they are not winning streaks, however the player has come close to winning before being blocked.
- Calculates the total number of consecutive streaks of 2 of its own color. We assign a weight of 2 to these.
- Repeats this process for the opponent player and calculates opponent utility.
- Returns a value equal to the computer’s utility after subtracting opponent utility.

Code:

```
def Evaluate(self, state):
        my_color = 2  # Assuming GameTreePlayer is always Player 2
        opponent_color = 1 if my_color == 2 else 2

        fours = self.checkForStreak(state, my_color, 4)
        threes = self.checkForStreak(state, my_color, 3)
        twos = self.checkForStreak(state, my_color, 2)

        opp_fours = self.checkForStreak(state, opponent_color, 4)
        opp_threes = self.checkForStreak(state, opponent_color, 3)
        opp_twos = self.checkForStreak(state, opponent_color, 2)

        return (fours * 10 + threes * 5 + twos * 2) - (opp_fours * 10 + opp_threes * 5 + opp_twos * 2)
```

Analysis Table:

Game Number	Total Moves	GameTreePlayer Moves	Winner
1	11	6	GameTreePlayer

2	3	2	GameTreePlayer
3	3	2	GameTreePlayer
4	3	2	GameTreePlayer
5	17	9	GameTreePlayer
6	17	9	GameTreePlayer
7	3	2	GameTreePlayer
8	3	2	GameTreePlayer
9	32	16	Myopic Player
10	3	2	GameTreePlayer
11	15	8	GameTreePlayer
12	30	15	Myopic Player
13	3	2	GameTreePlayer
14	20	10	GameTreePlayer
15	3	2	GameTreePlayer
16	14	7	GameTreePlayer
17	3	2	GameTreePlayer
18	5	3	GameTreePlayer
19	25	13	GameTreePlayer
20	21	11	GameTreePlayer
21	32	16	Myopic Player
22	3	2	GameTreePlayer
23	11	6	GameTreePlayer
24	3	2	GameTreePlayer
25	15	8	GameTreePlayer
26	3	2	GameTreePlayer
27	17	9	GameTreePlayer
28	3	2	GameTreePlayer
29	15	8	GameTreePlayer
30	3	2	GameTreePlayer
31	3	2	GameTreePlayer
32	34	17	Myopic Player
33	5	3	GameTreePlayer
34	3	2	GameTreePlayer
35	11	6	GameTreePlayer
36	13	7	GameTreePlayer
37	3	2	GameTreePlayer
38	15	8	GameTreePlayer
39	3	2	GameTreePlayer
40	3	2	GameTreePlayer
41	12	6	Myopic Player
42	3	2	GameTreePlayer
43	17	9	GameTreePlayer
44	5	3	GameTreePlayer
45	3	2	GameTreePlayer

46	3	2	GameTreePlayer
47	11	6	GameTreePlayer
48	13	7	GameTreePlayer
49	3	2	GameTreePlayer
50	3	2	GameTreePlayer

- Average Total Moves = 10.1
- Average GameTreePlayer Moves = 5.48
- Average GameTreePlayer Moves to Win = 4.53
- Win Rate = 90%

Evaluation Heuristic 2:

- This heuristic scores the players based on favourable positions captured on the board.
- For example, points are rewarded for having your pieces occupying the centre columns of the board:

```
# Add points for controlling the center column
        if col == 3:
            score += 2
```

- Points are further added for having consecutive pieces in a row or column:

```
# Add points for having consecutive pieces in a row
        row_streak = self.count_streak(row, col, state, color, (0, 1), 0)
        if row_streak >= 3:
            score += row_streak * 2

        # Add points for having consecutive pieces in a column
        col_streak = self.count_streak(row, col, state, color, (1, 0), 0)
        if col_streak >= 3:
            score += col_streak * 2
```

Analysis Table:

Game Number	Total Moves	GameTreePlayer Moves	Winner
1	15	8	GameTreePlayer
2	3	2	GameTreePlayer
3	15	8	GameTreePlayer
4	3	2	GameTreePlayer
5	14	7	Myopic Player
6	12	6	Myopic Player
7	17	9	GameTreePlayer
8	21	11	GameTreePlayer
9	12	6	Myopic Player
10	3	2	GameTreePlayer
11	16	8	Myopic Player
12	13	7	GameTreePlayer
13	12	6	Myopic Player
14	11	6	GameTreePlayer

15	17	9	GameTreePlayer
16	29	15	GameTreePlayer
17	30	15	Myopic Player
18	20	10	Myopic Player
19	17	9	GameTreePlayer
20	13	7	GameTreePlayer
21	8	4	Myopic Player
22	18	9	Myopic Player
23	11	6	GameTreePlayer
24	3	2	GameTreePlayer
25	17	9	GameTreePlayer
26	15	8	GameTreePlayer
27	3	2	GameTreePlayer
28	3	2	GameTreePlayer
29	18	9	Myopic Player
30	27	14	GameTreePlayer
31	21	11	GameTreePlayer
32	15	8	GameTreePlayer
33	12	6	Myopic Player
34	6	3	Myopic Player
35	3	2	GameTreePlayer
36	15	8	GameTreePlayer
37	17	9	GameTreePlayer
38	3	2	GameTreePlayer
39	34	17	Myopic Player
40	7	4	GameTreePlayer
41	9	5	GameTreePlayer
42	13	7	GameTreePlayer
43	33	17	GameTreePlayer
44	6	3	Myopic Player
45	24	12	Myopic Player
46	3	2	GameTreePlayer
47	20	10	Myopic Player
48	13	7	GameTreePlayer
49	12	6	Myopic Player
50	3	2	GameTreePlayer

- Average Total Moves = 13.7
- Average GameTreePlayer Moves = 7.18
- Average GameTreePlayer Moves to win = 6.73
- Win Rate = 66%

Evaluation Heuristic 3:

This is a simple evaluation function that only considers the number of pieces on the board without taking into account their positions or any strategic considerations. This was by far the least efficient evaluation heuristic.

Code:

```
def Evaluate(self, state):
        my_color = 2  # Assuming Game Tree player is always Player 2
        opponent_color = 1 if my_color == 2 else 2

        my_count = sum(row.count(my_color) for row in state)
        opponent_count = sum(row.count(opponent_color) for row in state)

        return my_count - opponent_count
```

Analysis Table:

Game Number	Total Moves	GameTreePlayer Moves	Winner
1	26	13	Myopic Player
2	23	12	GameTreePlayer
3	16	8	Myopic Player
4	23	12	GameTreePlayer
5	27	14	GameTreePlayer
6	28	14	Myopic Player
7	22	11	GameTreePlayer
8	18	9	Myopic Player
9	29	15	GameTreePlayer
10	24	12	Myopic Player
11	33	17	GameTreePlayer
12	15	8	GameTreePlayer
13	27	14	GameTreePlayer
14	19	10	GameTreePlayer
15	28	14	Myopic Player
16	32	16	Myopic Player
17	36	18	Myopic Player
18	25	13	GameTreePlayer
19	18	9	Myopic Player
20	20	10	Myopic Player
21	22	11	Myopic Player
22	16	8	Myopic Player
23	25	13	GameTreePlayer
24	23	12	GameTreePlayer
25	21	11	GameTreePlayer
26	28	14	Myopic Player
27	21	11	GameTreePlayer
28	15	8	GameTreePlayer
29	27	14	GameTreePlayer
30	18	9	Myopic Player
31	36	18	Myopic Player

32	12	6	Myopic Player
33	22	11	Myopic Player
34	27	14	GameTreePlayer
35	15	8	GameTreePlayer
36	32	16	Myopic Player
37	29	15	GameTreePlayer
38	23	12	GameTreePlayer
39	33	17	GameTreePlayer
40	30	15	Myopic Player
41	21	11	GameTreePlayer
42	29	15	GameTreePlayer
43	12	6	Myopic Player
44	20	10	Myopic Player
45	28	14	Myopic Player
46	25	13	GameTreePlayer
47	21	11	GameTreePlayer
48	33	17	GameTreePlayer
49	22	11	Myopic Player
50	25	13	GameTreePlayer

- Average Total Moves = 24.00
- Average GameTreePlayer Moves = 12.26
- Average GameTreePlayer Moves to win = 12.63
- Win rate = 54%

It is apparent that **Evaluation Heuristic 1** is the most efficient of the three.

Part B

The provided code implements the Minimax algorithm with alpha-beta pruning. It considers the evaluation function to determine the value of each game state and selects the best move based on the maximizing or minimizing player. The depth parameter limits the number of moves ahead the program analyzes.

To compare different evaluation functions, the code uses the `Evaluate` function to assign a value to each game state. The evaluation function can be modified to prioritize different factors, such as streaks of pieces or board positions.

```
def Minimax(self, state, depth, alpha, beta, maximizing_player):
        if depth == 0 or self.IsTerminal(state):
            return self.Evaluate(state), None

        legal_actions = self.GetLegalActions(state)

        if maximizing_player:
            value = float("-inf")
            best_action = None
            for action in legal_actions:
                new_state = self.GetResult(state, action, 2)
                if new_state is not None:
                    evaluation, _ = self.Minimax(
                        new_state, depth - 1, alpha, beta, False
                    )
                    if evaluation is not None and evaluation > value:
                        value = evaluation
                        best_action = action
                    alpha = max(alpha, value)
                    if alpha >= beta:
                        break
            return value, best_action
        else:
            value = float("inf")
            best_action = None
            for action in legal_actions:
                new_state = self.GetResult(state, action, 1)
                if new_state is not None:
                    evaluation, _ = self.Minimax(
                        new_state, depth - 1, alpha, beta, True
                    )
                    if evaluation is not None and evaluation < value:
                        value = evaluation
                        best_action = action
                    beta = min(beta, value)
                    if beta <= alpha:
                        break
            return value, best_action
```

In Part A I have compared the three different evaluation functions I have attempted to use, using the above Minimax algorithm. Evaluation heuristic 1 is the evaluation function I have chosen to ahead with.

Part C

In order to implement move ordering, we have to add the following code snippet to our Minimax function:

```
sorted_actions = sorted(
    legal_actions,
    key=lambda action: self.Evaluate(self.GetResult(state, action, 2)),
    reverse=maximizing_player  # For maximizing player, sort in descending order
)
```

How it works:

- `legal_actions`: The list of legal actions that the Game Tree player can take is about to be sorted by our custom lambda function.
- `key=lambda action: self.Evaluate(self.GetResult(state, action, 2))`: The `key` parameter specifies a custom lambda function to determine the sorting key for each element in `legal_actions`. For each action, it evaluates the resulting state after taking that action using the `Evaluate` function.
- `reverse=maximizing_player`: The `reverse` parameter is a boolean flag that, when set to `True`, sorts the elements in descending order. It is set based on whether the player is maximizing (`maximizing_player == True`) or minimizing (`maximizing_player == False`).

So, if the Game Tree player is maximizing, the actions will be sorted in descending order based on the evaluation of the resulting states, and if the player is minimizing, the actions will be sorted in ascending order.

Using Python's TQDM library, I compared the runtime of Minimax with and without using the move ordering heuristic. The game tree depth shall be 5.

Runtime without Move Ordering:

```
0%|                                                                                                                                        |
Moves: 21
GameTreePlayer Moves: 11
2%|███                                                                                                                             | 1/50 [0
Moves: 11
```

```
GameTreePlayer Moves: 6
4%|█████                                                                                                                         | 2/50 [00
Moves: 14
GameTreePlayer Moves: 7
6%|███████▌                                                                                                                      | 3/50 [00
Moves: 3
GameTreePlayer Moves: 2
8%|██████████▏                                                                                                                   | 4/50 [0
Moves: 17
GameTreePlayer Moves: 9
10%|████████████▌                                                                                                                 | 5/50 [
Moves: 34
GameTreePlayer Moves: 17
12%|███████████████▏                                                                                                              | 6/50 [
Moves: 3
GameTreePlayer Moves: 2
14%|█████████████████▋                                                                                                            | 7/50 [
Moves: 3
GameTreePlayer Moves: 2
16%|████████████████████▏                                                                                                         | 8/50 [
Moves: 3
GameTreePlayer Moves: 2
18%|██████████████████████▋                                                                                                       | 9/50 [
Moves: 13
GameTreePlayer Moves: 7
20%|█████████████████████████▏                                                                                                   | 10/50 [
Moves: 3
GameTreePlayer Moves: 2
22%|███████████████████████████▋                                                                                                 | 11/50 [
Moves: 11
GameTreePlayer Moves: 6
24%|██████████████████████████████▏                                                                                              | 12/50 [
Moves: 3
GameTreePlayer Moves: 2
26%|████████████████████████████████▊                                                                                            | 13/50 [
Moves: 3
GameTreePlayer Moves: 2
28%|███████████████████████████████████▎                                                                                         | 14/50 [
Moves: 3
GameTreePlayer Moves: 2
30%|█████████████████████████████████████▊                                                                                       | 15/50 [
Moves: 14
GameTreePlayer Moves: 7
32%|████████████████████████████████████████▎                                                                                    | 16/50 [
Moves: 15
GameTreePlayer Moves: 8
34%|██████████████████████████████████████████▊                                                                                  | 17/50 [
Moves: 20
GameTreePlayer Moves: 10
36%|█████████████████████████████████████████████▎                                                                               | 18/50 [
Moves: 3
GameTreePlayer Moves: 2
38%|███████████████████████████████████████████████▉                                                                             | 19/50 [
Moves: 15
GameTreePlayer Moves: 8
40%|██████████████████████████████████████████████████▍                                                                          | 20/50 [
Moves: 11
GameTreePlayer Moves: 6
42%|████████████████████████████████████████████████████▉                                                                        | 21/50 [
Moves: 13
GameTreePlayer Moves: 7
44%|███████████████████████████████████████████████████████▍                                                                     | 22/50 [
Moves: 3
GameTreePlayer Moves: 2
46%|█████████████████████████████████████████████████████████▉                                                                   | 23/50 [
Moves: 3
GameTreePlayer Moves: 2
48%|████████████████████████████████████████████████████████████▍                                                                | 24/50 [
Moves: 28
GameTreePlayer Moves: 14
50%|███████████████████████████████████████████████████████████████                                                              | 25/50 [
Moves: 13
GameTreePlayer Moves: 7
52%|█████████████████████████████████████████████████████████████████▌                                                           | 26/50 [
Moves: 15
GameTreePlayer Moves: 8
54%|████████████████████████████████████████████████████████████████████                                                         | 27/50 [
Moves: 3
```

```
GameTreePlayer Moves: 2
56%|██████████████████████████████████████████████████████████████████████▌                                                       | 28/50 [
Moves: 3
GameTreePlayer Moves: 2
58%|█████████████████████████████████████████████████████████████████████████                                                     | 29/50 [
Moves: 13
GameTreePlayer Moves: 7
60%|███████████████████████████████████████████████████████████████████████████▌                                                  | 30/50 [
Moves: 3
GameTreePlayer Moves: 2
62%|██████████████████████████████████████████████████████████████████████████████                                                | 31/50 [
Moves: 3
GameTreePlayer Moves: 2
64%|████████████████████████████████████████████████████████████████████████████████▋                                             | 32/50 [
Moves: 3
GameTreePlayer Moves: 2
66%|███████████████████████████████████████████████████████████████████████████████████▏                                          | 33/50 [
Moves: 3
GameTreePlayer Moves: 2
68%|█████████████████████████████████████████████████████████████████████████████████████▋                                        | 34/50 [
Moves: 5
GameTreePlayer Moves: 3
70%|████████████████████████████████████████████████████████████████████████████████████████▏                                     | 35/50 [
Moves: 15
GameTreePlayer Moves: 8
72%|██████████████████████████████████████████████████████████████████████████████████████████▋                                   | 36/50 [
Moves: 3
GameTreePlayer Moves: 2
74%|█████████████████████████████████████████████████████████████████████████████████████████████▏                                | 37/50 [
Moves: 3
GameTreePlayer Moves: 2
76%|███████████████████████████████████████████████████████████████████████████████████████████████▊                              | 38/50 [
Moves: 32
GameTreePlayer Moves: 16
78%|██████████████████████████████████████████████████████████████████████████████████████████████████▎                           | 39/50 [
Moves: 13
GameTreePlayer Moves: 7
80%|████████████████████████████████████████████████████████████████████████████████████████████████████▊                         | 40/50 [
Moves: 15
GameTreePlayer Moves: 8
82%|███████████████████████████████████████████████████████████████████████████████████████████████████████▎                      | 41/50 [
Moves: 16
GameTreePlayer Moves: 8
84%|█████████████████████████████████████████████████████████████████████████████████████████████████████████▊                    | 42/50 [
Moves: 15
GameTreePlayer Moves: 8
86%|████████████████████████████████████████████████████████████████████████████████████████████████████████████▎                 | 43/50 [
Moves: 15
GameTreePlayer Moves: 8
88%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████▉               | 44/50 [
Moves: 3
GameTreePlayer Moves: 2
90%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍            | 45/50 [
Moves: 3
GameTreePlayer Moves: 2
92%|███████████████████████████████████████████████████████████████████████████████████████████████████████████████████▉          | 46/50 [
Moves: 17
GameTreePlayer Moves: 9
94%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍       | 47/50 [
Moves: 13
GameTreePlayer Moves: 7
96%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▉     | 48/50 [
Moves: 17
GameTreePlayer Moves: 9
98%|███████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍  | 49/50 [
Moves: 10
GameTreePlayer Moves: 5
100%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 50/50
```

We’re able to observe an average iteration time of approximately 2.1 seconds with a total runtime of 105 seconds.

Runtime with Move Ordering:

```
0%|                                                                                                                                       |
Moves: 3
GameTreePlayer Moves: 2
  2%|██▌                                                                                                                          | 1/50
Moves: 19
GameTreePlayer Moves: 10
  4%|█████                                                                                                                        | 2/50
Moves: 9
GameTreePlayer Moves: 5
  6%|███████▌                                                                                                                     | 3/50
Moves: 3
GameTreePlayer Moves: 2
  8%|██████████                                                                                                                   | 4/50
Moves: 3
GameTreePlayer Moves: 2
 10%|████████████▌                                                                                                                | 5/50
Moves: 13
GameTreePlayer Moves: 7
 12%|███████████████                                                                                                              | 6/50
Moves: 15
GameTreePlayer Moves: 8
 14%|█████████████████▋                                                                                                           | 7/50
Moves: 36
GameTreePlayer Moves: 18
 16%|████████████████████▏                                                                                                        | 8/50
Moves: 20
GameTreePlayer Moves: 10
 18%|██████████████████████▋                                                                                                      | 9/50
Moves: 15
GameTreePlayer Moves: 8
 20%|█████████████████████████▏                                                                                                    | 10/50
Moves: 11
GameTreePlayer Moves: 6
 22%|███████████████████████████▋                                                                                                  | 11/50
Moves: 13
GameTreePlayer Moves: 7
 24%|██████████████████████████████▏                                                                                               | 12/50
Moves: 9
GameTreePlayer Moves: 5
 26%|████████████████████████████████▊                                                                                             | 13/50
Moves: 3
GameTreePlayer Moves: 2
 28%|███████████████████████████████████▎                                                                                          | 14/50
Moves: 20
GameTreePlayer Moves: 10
 30%|█████████████████████████████████████▊                                                                                        | 15/50
Moves: 11
GameTreePlayer Moves: 6
 32%|████████████████████████████████████████▎                                                                                     | 16/50
Moves: 31
GameTreePlayer Moves: 16
 34%|██████████████████████████████████████████▊                                                                                   | 17/50
Moves: 14
GameTreePlayer Moves: 7
 36%|█████████████████████████████████████████████▎                                                                                | 18/50
Moves: 15
GameTreePlayer Moves: 8
 38%|███████████████████████████████████████████████▉                                                                              | 19/50
Moves: 19
GameTreePlayer Moves: 10
 40%|██████████████████████████████████████████████████▍                                                                           | 20/50
Moves: 14
GameTreePlayer Moves: 7
 42%|████████████████████████████████████████████████████▉                                                                         | 21/50
Moves: 30
GameTreePlayer Moves: 15
 44%|███████████████████████████████████████████████████████▍                                                                      | 22/50
Moves: 3
GameTreePlayer Moves: 2
 46%|█████████████████████████████████████████████████████████▉                                                                    | 23/50
Moves: 11
GameTreePlayer Moves: 6
 48%|████████████████████████████████████████████████████████████▍                                                                 | 24/50
Moves: 15
GameTreePlayer Moves: 8
 50%|███████████████████████████████████████████████████████████████                                                               | 25/50
Moves: 15
```

```
GameTreePlayer Moves: 8
 52%|█████████████████████████████████████████████████████████████████▌                                                            | 26/50
Moves: 3
GameTreePlayer Moves: 2
 54%|████████████████████████████████████████████████████████████████████                                                          | 27/50
Moves: 17
GameTreePlayer Moves: 9
 56%|██████████████████████████████████████████████████████████████████████▌                                                       | 28/50
Moves: 17
GameTreePlayer Moves: 9
 58%|█████████████████████████████████████████████████████████████████████████                                                     | 29/50
Moves: 3
GameTreePlayer Moves: 2
 60%|███████████████████████████████████████████████████████████████████████████▌                                                  | 30/50
Moves: 15
GameTreePlayer Moves: 8
 62%|██████████████████████████████████████████████████████████████████████████████                                                | 31/50
Moves: 17
GameTreePlayer Moves: 9
 64%|████████████████████████████████████████████████████████████████████████████████▋                                             | 32/50
Moves: 3
GameTreePlayer Moves: 2
 66%|███████████████████████████████████████████████████████████████████████████████████▏                                          | 33/50
Moves: 19
GameTreePlayer Moves: 10
 68%|█████████████████████████████████████████████████████████████████████████████████████▋                                        | 34/50
Moves: 3
GameTreePlayer Moves: 2
 70%|████████████████████████████████████████████████████████████████████████████████████████▏                                     | 35/50
Moves: 3
GameTreePlayer Moves: 2
 72%|██████████████████████████████████████████████████████████████████████████████████████████▋                                   | 36/50
Moves: 33
GameTreePlayer Moves: 17
 74%|█████████████████████████████████████████████████████████████████████████████████████████████▏                                | 37/50
Moves: 15
GameTreePlayer Moves: 8
 76%|███████████████████████████████████████████████████████████████████████████████████████████████▊                              | 38/50
Moves: 15
GameTreePlayer Moves: 8
 78%|██████████████████████████████████████████████████████████████████████████████████████████████████▎                           | 39/50
Moves: 11
GameTreePlayer Moves: 6
 80%|████████████████████████████████████████████████████████████████████████████████████████████████████▊                         | 40/50
Moves: 3
GameTreePlayer Moves: 2
 82%|███████████████████████████████████████████████████████████████████████████████████████████████████████▎                      | 41/50
Moves: 3
GameTreePlayer Moves: 2
 84%|█████████████████████████████████████████████████████████████████████████████████████████████████████████▊                    | 42/50
Moves: 17
GameTreePlayer Moves: 9
 86%|████████████████████████████████████████████████████████████████████████████████████████████████████████████▎                 | 43/50
Moves: 3
GameTreePlayer Moves: 2
 88%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████▉               | 44/50
Moves: 26
GameTreePlayer Moves: 13
 90%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍            | 45/50
Moves: 3
GameTreePlayer Moves: 2
 92%|███████████████████████████████████████████████████████████████████████████████████████████████████████████████████▉          | 46/50
Moves: 3
GameTreePlayer Moves: 2
 94%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍       | 47/50
Moves: 15
GameTreePlayer Moves: 8
 96%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▉     | 48/50
Moves: 19
GameTreePlayer Moves: 10
 98%|███████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████▍  | 49/50
Moves: 3
GameTreePlayer Moves: 2
100%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 50/50
```

We’re able to observe an average iteration time of approximately 2.6 seconds with a total runtime of 130 seconds.

Hence we can conclude that the move ordering heuristic in fact slows the program instead of speeding it up. This could possibly be due to an increase in recursive calls because of the particular evaluation heuristic (Evaluation Heuristic 1) I have used.

Part D

Increasing the cut-off depth from 3 to 5 increased the frequency of winning as it allows the AI player to explore deeper into the game tree and potentially make better decisions. My observations:

1. **More Wins:** A higher cutoff depth allowed the algorithm to make more informed decisions by looking ahead at a greater number of possible moves. The win rate increased from 86% to 90%.
2. **Decreased Average Moves Before a Win:** With a deeper search, the algorithm found winning moves more quickly, reducing the average number of moves before a win. The number of games in which total moves equaled 3 and the GameTreePlayer won, drastically increased.

Conclusions

Based on the results of the Connect 4 Artificial Intelligence Assignment 2, several observations can be made:

1. Firstly, the move ordering heuristic implemented in the program did not yield the expected outcome of speeding up the program. Instead, it seemed to slow it down, possibly due to an increase in recursive calls caused by the specific evaluation heuristic used.
2. Additionally, increasing the cut-off depth from 3 to 5 in the game tree search had a positive impact on the performance of the AI player. With a deeper search, the algorithm was able to make more informed decisions and achieve a higher win rate of 90%. Moreover, the average number of moves before a win decreased, indicating that the AI player found winning moves more quickly.
3. In conclusion, while the move ordering heuristic did not produce the desired results, increasing the cut-off depth proved to be beneficial for the Connect 4 AI. These findings highlight the importance of carefully selecting and optimizing different components of an AI algorithm to enhance its performance and decision-making capabilities.