AI ASSIGNMENT 2

CONNECT 4

(SOUMIL GHOSH 2020B1A72102G)

Test Case Passed

```
Roll no : 2020B1A72102G
Player 2 has won. Testcase passed.
Moves : 3
```

EVALUATION FUNCTIONS USED

(Part a and Part b)

1. **Based on positions of GameTree player and Myopic player:**

   Based on how many pieces are in the window and how they are arranged, the program calculates scores.

   A winning move for the gametreeplayer is indicated by a very high positive score (1000000000) if there are four consecutive pieces of the piece of the gametreeplayer.

It gives a somewhat positive score of five if there are three consecutive pieces of the piece with one empty space, suggesting that the gametreeplayer is in an advantageous position.

It gives a lower positive score (2), indicating a respectable position for the gametreeplayer, if there are two consecutive pieces of the piece with two empty slots.

In a similar vein, it looks for the pieces of the opposition and deducts points from areas where the opposition has an advantage. A winning move for the myopicplayer is indicated by a very high negative score (-1000000000) if there are four consecutive pieces of the piece of the myopicplayer.

It gives a negative score of -4 if there are three opposition pieces in a row.

```
Average Number of Moves: 19.28
GAMETREE_PLAYER Wins: 50/50
PS C:\Users\ASUS\Downloads\ass2>
```

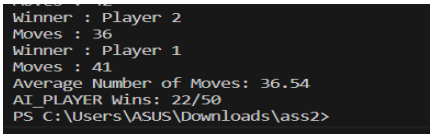
2. **Prioritising moves that maximize piece count in a window**

By increasing the score by gametreeplayer * 1000000, it gives the gametreeplayer bonus points for having more pieces in the window.

It deducts myopicplayer_count * 10000 from the score to penalize the opponent for having more pieces in the window.

By adding empty_count * 100 to the score, it gives an extra bonus for having empty spaces in the window. This promotes possible motions that might eventually result in finishing a line in turns to come.

All things considered, the assessment function's goal is to steer the AI player toward actions that both boost their odds of victory and impede their opponent's advancement. It emphasizes managing the window with the player's pieces and setting up situations where winning movements can be made in the future.

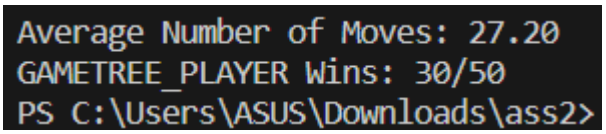


3. **Encourage the AI to make moves that lead to winning configurations**
   If all four cells in the window contain the player's pieces (GAMETREE_PLAYER), it means the player has won in this window. In this case, a score of 2000 is assigned, indicating a highly favorable state.

   If all four cells in the window contain the opponent's pieces, it means the opponent has won in this window. In this case, a score of -2000 is assigned, indicating a highly unfavorable state.

   If neither the player nor the opponent has won in this window, the function returns 0, indicating a neutral state.

   This evaluate function is a component of a bigger evaluation system that gives a state a score by taking into account various game board elements. The game-playing algorithm then uses the score that has been returned to determine which movements are more advantageous. Encouraging the AI to make actions that result in winning configurations and discouraging moves that result in losing configurations is the rationale behind this.



Comparison between the evaluation functions:

Scoring Logic:

The first function uses a more detailed scoring logic, considering the count of the gametreeplayer's pieces, myopic player’s pieces, and empty spaces in the window.

The second and third functions provide fixed scores based on the count of pieces in the window without considering the context of the game state.

Flexibility:

The first function is more flexible because it considers both gametreeplayer and myopic player counts, allowing for more nuanced evaluation.

The second and third functions have fixed scores for specific patterns, limiting their adaptability to different game situations.

Context Awareness:

The first function is context-aware, providing rewards for having more pieces and additional rewards for empty spaces, which can potentially complete a line.

The second and third functions lack this context awareness and provide fixed scores based on the count of pieces without considering the strategic importance of empty spaces.

Why the First Function Performs Better:

Context Sensitivity:

The first function's scoring is more sensitive to the context of the game, considering both the gametreeplayer's and myopicplayer's positions along with the potential for completing lines with empty spaces.

Adaptability:

The first function's adaptive scoring allows it to respond better to changing game situations, making it more versatile in evaluating different board states.

Nuanced Evaluation:

The first function provides a nuanced evaluation by distinguishing between gametreeplayer and myopic counts, making it capable of capturing more subtle aspects of the game state.

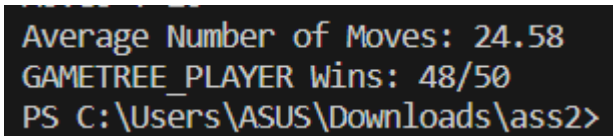
MOVE ORDERING HEURISTIC (Part C)

This move ordering heuristic guides the search algorithm to explore moves that are likely to lead to favorable positions with more potential connections for the AI player, helping to prune the search space more effectively. This can lead to faster and more efficient decision-making in the game.

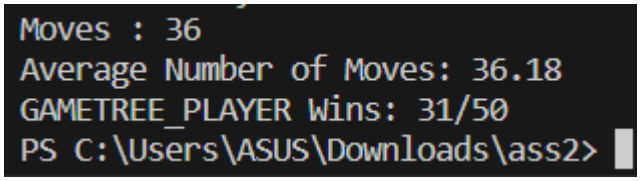
In this modified code, the move_order_heuristic function evaluates the potential connections each move can create for both the AI player and the human player. The heuristic value is then computed as the difference between AI connections and human connections. This heuristic aims to prioritize moves that maximize the AI player's chances of forming winning combinations while minimizing the human player's opportunities.

This move ordering heuristic guides the search algorithm to explore moves that are likely to lead to favorable positions with more potential connections for the AI player, helping to prune the search space more effectively. This can lead to faster and more efficient decision-making in the game. By considering potential connections, the move ordering heuristic aims to guide the AI player towards making moves that increase its chances of winning in subsequent turns.

When used with evaluation function 1

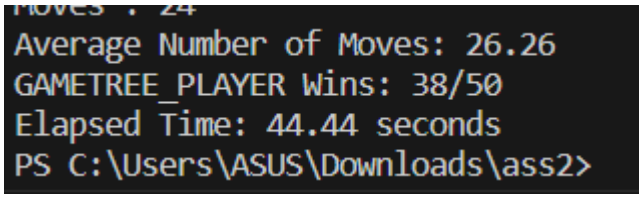


When used with evaluation function 2

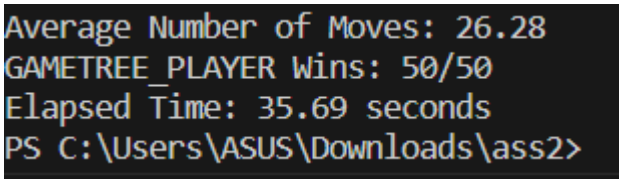


When used with evaluation function 3

Without heuristic



With Heuristic



Using the move ordering heuristic decreases the average number of moves required for the gametreeplayer to win.

As we can see, when the heuristic is used the elapsed time also decreases.

It also increases the GAMETREE_PLAYER wins out of 50 in evaluation functions 2 and 3.

CUTOFF DEPTH

For evaluation function 1

When cutoff depth = 3

```
Average Number of Moves: 20.14
AI_PLAYER Wins: 45/50
PS C:\Users\ASUS\Downloads\ass2>
```

When cutoff depth = 5

```
  Moves : 18
  Average Number of Moves: 19.62
  AI_PLAYER Wins: 49/50
```

For evaluation function 3 with heuristic function

When cutoff depth = 3

```
Moves : 28
Average Number of Moves: 25.06
GAMETREE_PLAYER Wins: 46/50
PS C:\Users\ASUS\Downloads\ass2>
```

When cutoff depth = 5

```
Moves : 20
Average Number of Moves: 27.78
GAMETREE_PLAYER Wins: 49/50
PS C:\Users\ASUS\Downloads\ass2>
```

For Evaluation function 2

When cutoff depth = 3

```
 Average Number of Moves: 27.06
 GAMETREE_PLAYER Wins: 18/50
 Elapsed Time: 7.99 seconds
```

When cutoff depth = 5

```
Average Number of Moves: 37.58
GAMETREE_PLAYER Wins: 19/50
Elapsed Time: 144.19 seconds
```

The algorithm investigates the game tree more thoroughly and takes into account more potential moves as you increase the cutoff depth. The algorithm will be able to make decisions with greater knowledge since it will fully comprehend the possible outcomes of every action. Deeper searches, however, also take longer and more processing power.