Final Project

Project Title: Heuristic Alpha-Beta Tree Search Algorithm :

In this project, I implemented an Alpha-Beta tree experimental search algorithm in Python. The alpha-beta tree hemp search algorithm is improved minimax algorithm for AI agents to play games. The project aims to build a file He can play a game using Heuristic Alpha-Beta Tree Search algorithm. I chose tic-tac-toe for this project

The alpha beta tree search algorithm is an iterative algorithm that explores the game tree to find the best move for the AI operator. At each tree level, the algorithm uses alpha-beta pruning to get rid of unnecessary branches and reduce the number of nodes that need to be explored. The algorithm also uses the heuristic evaluation function to estimate the state value of each game. Give priority to exploring more promising branches. To implement the beta tree search algorithm, I define an iterative function that inputs the current game state, depth, player, and alpha and beta values. The function should be to return the best move for the current player in the current position. I tested the software by playing against it (AI vs. Human).

Title:Tic-Tac-Toe using Heuristic Alpha-Beta Tree Search Algorithm : OpenAI's ChatGPT AI language model Date Accessed:Jun 5 ,2023

- Q give me Tic-Tac-Toe using Alpha-Beta Tree Search Algorithm in python code
- Q Make the human options 1...9 instead of 0.2 to move
- Q Add an option for the player to start first or for the AI to start first

Modifications: I added the code to play (AI vs AI) And choose the player to play with 'x' or 'o' and I made some adjustments the strings of the results.

I took a few lines from https://github.com/anmolchandelCO180309 Like the format of the board

There are 11 functions used in this project. Let's explain each one separately :

1-This function creates and returns an empty game board as a 2D list. It initializes all cells with the EMPTY constant.

```
# Function to create an empty board
def create_board():
    return [[EMPTY] * N for _ in range(N)]
```

2- This function takes a game board and a move as input and checks if the move is correct. It converts the move to the coordinates of the canvas using the move_to_coordinate function and checks if the corresponding cell is empty

```
# Function to check if a move is valid
def is_valid_move(board, move):
    row, col = move_to_coordinate(move)
    return board[row][col] == EMPTY
```

3- This function updates the game board by making a move. It converts the move to board coordinates using the move_to_coordinate function and assigns the corresponding cell to the player's value (PLAYER_X or PLAYER_O)

```
# Function to make a move
def make_move(board, move, player):
    row, col = move_to_coordinate(move)
    board[row][col] = player
```

4- This function takes a move as input and converts it to board coordinates (row, col) using simple arithmetic calculations. It uses integer division (//) and modulo (%) operators to determine the row and column indices

```
# Function to convert a move to board coordinates
def move_to_coordinate(move):
    row = (move - 1) // N
    col = (move - 1) % N
    return row, col
```

5- This function takes row and column indices as input and converts them to a move using the inverse calculation of the move_to_coordinate function

```
# Function to convert board coordinates to a move
def coordinate_to_move(row, col):
    return row * N + col + 1
```

6- This function checks if the game has ended by examining the current state of the game board. It checks for winning conditions in rows, columns, and diagonals. If any player has a sum of N (either N or -N) in a row, column, or diagonal, it returns True. It also checks if the board is full, resulting in a draw

```python
# Function to convert board coordinates to a move
def coordinate_to_move(row, col):
    return row * N + col + 1

# Function to check if the game has ended
def game_over(board):
    # Check rows
    for i in range(N):
        if abs(sum(board[i])) == N:
            return True

    # Check columns
    for j in range(N):
        if abs(sum(row[j] for row in board)) == N:
            return True

    # Check diagonals
    diagonal1 = [board[i][i] for i in range(N)]
    diagonal2 = [board[i][N - 1 - i] for i in range(N)]
    if abs(sum(diagonal1)) == N or abs(sum(diagonal2)) == N:
        return True

    # Check if the board is full (draw)
    if all(board[i][j] != EMPTY for i in range(N) for j in range(N)):
        return True

    return False
```

7- This function evaluates the current state of the board and returns a score. It checks for winning conditions in rows, columns, and diagonals, similar to the game_over function. If player X wins, it returns 1. If player O wins, it returns -1. If the game is a draw or still in progress, it returns 0

```python
# Function to evaluate the current state of the board
def evaluate(board):
    # Check rows
    for i in range(N):
        row_sum = sum(board[i])
        if row_sum == N:
            return 1  # Player X wins
        elif row_sum == -N:
            return -1  # Player O wins

    # Check columns
    for j in range(N):
        col_sum = sum(row[j] for row in board)
        if col_sum == N:
            return 1
        elif col_sum == -N:
            return -1

    # Check diagonals
    diagonal1 = [board[i][i] for i in range(N)]
    diagonal2 = [board[i][N - 1 - i] for i in range(N)]
    diagonal1_sum = sum(diagonal1)
    diagonal2_sum = sum(diagonal2)
    if diagonal1_sum == N or diagonal2_sum == N:
        return 1
    elif diagonal1_sum == -N or diagonal2_sum == -N:
        return -1

    return 0  # Draw or game not yet finished
```

8- This function performs the Alpha-Beta pruning algorithm for finding the optimal move. It uses a recursive approach to explore the game tree up to a certain depth. It evaluates the game state using the evaluate function and applies the Alpha-Beta pruning technique to improve efficiency. The alpha and beta parameters represent the lower and upper bounds for the best possible outcome. The player parameter indicates whose turn it is. The function returns the evaluation score for the current game state

```
# Function to perform the Alpha-Beta tree search
def alpha_beta_search(board, depth, alpha, beta, player):
    if game_over(board) or depth == 0:
        return evaluate(board)

    if player == PLAYER_X:
        max_eval = -math.inf
        for move in range(1, N * N + 1):
            if is_valid_move(board, move):
                make_move(board, move, PLAYER_X)
                eval_score = alpha_beta_search(board, depth - 1, alpha, beta, PLAYER_O)
                make_move(board, move, EMPTY)
                max_eval = max(max_eval, eval_score)
                alpha = max(alpha, eval_score)
                if beta <= alpha:
                    break  # Beta cutoff
        return max_eval

    else:  # Player_O's turn
        min_eval = math.inf
        for move in range(1, N * N + 1):
            if is_valid_move(board, move):
                make_move(board, move, PLAYER_O)
                eval_score = alpha_beta_search(board, depth - 1, alpha, beta, PLAYER_X)
                make_move(board, move, EMPTY)
                min_eval = min(min_eval, eval_score)
                beta = min(beta, eval_score)
                if beta <= alpha:
                    break  # Alpha cutoff
        return min_eval
```

9- This function finds the best move for a given player using the Alpha-Beta tree search. It iterates over all possible moves, evaluates each move using the alpha_beta_search function, and keeps track of the best move based on the evaluation score. The depth parameter determines the depth of the tree search. The function returns the best move

```
# Function to find the best move using Alpha-Beta tree search
def find_best_move(board, depth, player):
    best_eval = -math.inf if player == PLAYER_X else math.inf
    best_move = None

    for move in range(1, N * N + 1):
        if is_valid_move(board, move):
            make_move(board, move, player)
            eval_score = alpha_beta_search(board, depth - 1, -math.inf, math.inf, -player)
            make_move(board, move, EMPTY)

            if player == PLAYER_X and eval_score > best_eval:
                best_eval = eval_score
                best_move = move
            elif player == PLAYER_O and eval_score < best_eval:
                best_eval = eval_score
                best_move = move

    return best_move
```

10- This function creates a visual representation of the Tic-Tac-Toe board by printing the board values using the symbols 'X', 'O', and ' ' in a formatted grid

```python
# Function to print the board
def Gameboard(board):
    chars = {1: 'X', -1: 'O', 0: ' '}
    for x in board:
        for y in x:
            ch = chars[y]
            print(f'| {ch} |', end='')
        print('\n' + '---------------')
    print('===============')
```

11- The play_game function is the main function that controls the flow of the Tic-Tac-Toe game. Here's a brief explanation of the function :

1-It starts by creating an empty game board using the create_board function.

2-It prompts the user to choose between playing against a human (AI vs Human) or watching AI vs AI gameplay.

3-If the user chooses to play against a human, it asks if they want to start first and prompts for their preferred symbol (X or O).

4-If the user chooses to watch AI vs AI gameplay, the game will be played automatically.

5-The variable depth is set to the maximum number of possible moves on the board

6-The game starts with a while loop that continues until the game is over Inside the loop, it prints the current state of the board using the print_board function

7-Depending on the current player (PLAYER_X or PLAYER_O), it prompts for a move from the user or selects the best move using the find_best_move function.

8-The move is validated using the is_valid_move function, and if it is valid, the move is made on the board using the make_move function After each move, the current player is switched

9-The loop continues until the game is over, which is determined by the game_over function

10-If the game is over, it evaluates the board to determine the winner using the evaluation function. Depending on the result, it prints the appropriate message

11-The game then either ends or continues depending on the user's choice to play again

In summary, the play_game function controls the game flow, handles user input, AI moves, and checks for a winner or draw at each step. It provides a user-friendly interface for playing Tic-Tac-Toe against a human or watching AI vs AI gameplay

```
def play_game():
    board = create_board()

    print("Welcome to Tic-Tac-Toe!💪💪💪")
    choice = input('1- (AI vs Human)\n2- (AI vs AI) : ')
    if choice =='1' :

        player_starts = input("Do you want to start first? (y/n): ")
        if player_starts.lower() == 'y':
            choice = input('Choose X or O\nChosen: ').upper()
            if choice == 'X' :
                current_player = PLAYER_X
            else:
                current_player = PLAYER_O

        else:
            print('You play as o')
            current_player = PLAYER_O


        depth = N * N

        while True:
            print_board(board)


            if current_player == PLAYER_X:
                print("Player X's turn:")
                move = int(input("Enter your move (1-9): "))
                if 1 <= move <= N * N and is_valid_move(board, move):
                    make_move(board, move, current_player)
                    current_player = PLAYER_O
                else:
                    print("Invalid move. Try again.😉")
            else:
                print("Player O's turn:")
                best_move = find_best_move(board, depth, current_player)
                if best_move is not None:
                    make_move(board, best_move, current_player)
                    current_player = PLAYER_X
```

```python
            if game_over(board):
                winner = evaluate(board)
                if winner == 1:
                    print("Player X wins!🎉🎉✨")
                elif winner == -1:
                    print("Player O wins!🎉🎉✨")
                else:
                    print("It's a draw! Try agane 😉")
                break
    else :
        current_player = PLAYER_X
        print("AI is Player O.")

        while True:
            print_board(board)

            if game_over(board):
                winner = evaluate(board)
                if winner == 1:
                    print("Player X wins!🎉🎉✨")
                elif winner == -1:
                    print("Player O wins!🎉🎉✨")
                else:
                    print("It's a draw! Try agane 😉")
                break

            if current_player == PLAYER_X:
                print("Player X's turn:")
                best_move = find_best_move(board, N * N, PLAYER_X)
                make_move(board, best_move, PLAYER_X)
                current_player = PLAYER_O
            else:
                print("Player O's turn:")
                best_move = find_best_move(board, N * N, PLAYER_O)
                make_move(board, best_move, PLAYER_O)
                current_player = PLAYER_X


play_game()
```

Results :

1-

```
Welcome to Tic-Tac-Toe!💪💪💪
1- (AI vs Human)
2- (AI vs AI) : 2
AI is Player O.
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player X's turn:
| X ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| X ||   ||   |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
```

```
Player X's turn:
| X || X ||   |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| X || X || O |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
===============
Player X's turn:
| X || X || O |
---------------
|   || O ||   |
---------------
| X ||   ||   |
---------------
===============
```

```
===============
Player X's turn:
| X || X || O |
---------------
| O || O || X |
---------------
| X ||   ||   |
---------------
===============
Player O's turn:
| X || X || O |
---------------
| O || O || X |
---------------
| X || O ||   |
---------------
===============
Player X's turn:
| X || X || O |
---------------
| O || O || X |
---------------
| X || O || X |
---------------
===============
It's a draw! Try agane 😉
```

2-

-

```
Welcome to Tic-Tac-Toe!💪💪💪
1- (AI vs Human)
2- (AI vs AI) : 1
Do you want to start first? (y/n): y
Choose X or O
Chosen: x
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player X's turn:
Enter your move (1-9): 5
|   ||   ||   |
---------------
|   || x ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| o ||   ||   |
---------------
|   || x ||   |
---------------
|   ||   ||   |
---------------
===============
Player X's turn:
Enter your move (1-9): 3
| o ||   || x |
---------------
|   || x ||   |
---------------
|   ||   ||   |
---------------
```

```
===============
Player X's turn:
Enter your move (1-9): 3
| o ||   || x |
---------------
|   || x ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| o ||   || x |
---------------
|   || x ||   |
---------------
| o ||   ||   |
---------------
===============
Player X's turn:
Enter your move (1-9): 6
| o ||   || x |
---------------
| O || x || x |
---------------
| o ||   ||   |
---------------
===============
Player O's turn:
Player O wins!🎉🎉✨
```

3-

```
Welcome to Tic-Tac-Toe!💪💪💪
1- (AI vs Human)
2- (AI vs AI) : 1
Do you want to start first? (y/n): n
You play as o
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| o ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player X's turn:
Enter your move (1-9): 2
| o || x ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
Player O's turn:
| o || x ||   |
---------------
| o ||   ||   |
---------------
|   ||   ||   |
---------------
---------------
```

```
===============
Player X's turn:
Enter your move (1-9): 7
| o || x ||   |
---------------
| o ||   ||   |
---------------
| x ||   ||   |
---------------
===============
Player O's turn:
| o || x ||   |
---------------
| o || o ||   |
---------------
| x ||   ||   |
---------------
===============
Player X's turn:
Enter your move (1-9): 6
| o || x ||   |
---------------
| o || o || x |
---------------
| x ||   || O |
---------------
===============
Player O's turn:
Player O wins!🎉🎉✨
```

—

Conclusion:

In conclusion, this project titled "Tic-Tac-Toe using Heuristic Alpha-Beta Tree Search Algorithm" implemented an Alpha-Beta tree search algorithm in Python. The goal was to develop an AI agent capable of playing the game of Tic-Tac-Toe using an improved minimax algorithm. The algorithm utilizes the alpha-beta pruning technique to optimize the search process and uses a heuristic evaluation function to prioritize exploring more promising branches of the game tree

The project successfully implemented the game logic for Tic-Tac-Toe, allowing users to play against the AI agent or watch AI vs AI gameplay. The AI agent makes its moves based on the Alpha-Beta tree search algorithm, aiming to find the best move that maximizes its chances of winning. The game also includes options for the player to choose the starting player and their preferred symbol

Throughout the project, several functions were defined to handle different aspects of the game, such as creating the game board, validating moves, updating the board, checking for a game over condition, evaluating the board state, and performing the Alpha-Beta pruning algorithm

The project concludes by providing a user-friendly interface through the play_game function, which controls the flow of the game, handles user input, AI moves, and determines the winner or draw. The implementation includes a visual representation of the board using symbols, making it easy for users to understand the game state

Overall, this project demonstrates the application of the Heuristic Alpha-Beta Tree Search Algorithm in the context of Tic-Tac-Toe, showcasing the benefits of optimization techniques and heuristic evaluation functions in improving AI gameplay