Assignment 2 Alpha-Beta Pruning for Tic-Tac-Toe

📌 Important Information

- **Deadline**: October 31, 2024, by 23:59 (`Beijing Time`)
- **Submission File**:
  - You should additionally export your Jupyter Notebook (.ipynb file) with your answer as a **PDF**
  - Please ensure the output of autotest is viewed as a scrollable element.
- **Submission Format**:
  - Submit your answer via sustech blackboard system.
  - Place all the files (PDF and .ipynb) in a single folder, name it using your student ID_name (e.g., 12431112_WangShuoyuan), and compress it into a zip file. `The file structure should be` :

    ```
    12431112_WangShuoyuan/
    |-- coding_assignment2.pdf
    |-- coding_assignment2.ipynb
    ```

ℹ️ Introduction

In Assignment 2, you will complete the implementation of the **alpha-beta pruning algorithm** as part of a tic-tac-toe game.

The AI must make optimal moves using the minimax algorithm with alpha-beta pruning, aiming to maximize its own chances of winning while minimizing the human's chances. Alpha-beta pruning helps optimize the minimax algorithm by reducing the number of nodes evaluated, thereby improving efficiency. You will modify the provided code to complete the `minimax` function, ensuring it properly utilizes alpha-beta pruning.

⭐ Hint

- The `minimax` function has two main parts: maximizing the AI's score (`is_maximizing = True`) and minimizing the opponent's score (`is_maximizing = False`).
- You need incorporate the parameters `alpha` and `beta` to optimize the search.
- You can stop evaluating further moves in the branch of `early return` because they won't influence the final decision.
- Remember to update alpha and beta values appropriately during the recursion to ensure optimal pruning.

In [7]:

```
import math
import ipywidgets as widgets
from IPython.display import display, clear_output, HTML

# Define the board
def print_board(board):
    board_html = '<br>'.join([' | '.join(row) for row in board])
    board_html = board_html.replace(" ", " ")
    return board_html
```

```python
# Check the game state
def check_winner(board):
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] and board[i][0] != ' ':
            return board[i][0]
        if board[0][i] == board[1][i] == board[2][i] and board[0][i] != ' ':
            return board[0][i]
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return board[0][2]
    if all(cell != ' ' for row in board for cell in row):
        return 'Tie'
    return None

# Get available moves
def get_available_moves(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

# Implementation of the Minimax algorithm (including alpha-beta pruning)
def minimax(board, depth, is_maximizing, alpha, beta):
    """
    Args:
    - board (list): A 2D list representing the current state of the Tic-Tac-Toe board.
    - depth (int): The depth of the game tree, indicating how far the algorithm has searched.
    - is_maximizing (bool): Whether it's the maximizing AI's turn (True) or the minimizing hu
    - alpha (float): The best score the maximizing AI can guarantee.
    - beta (float): The best score the minimizing human can guarantee.

    Returns:
    - int: An integer representing the evaluation of the board state, adjusted for depth.
    """
    global prunes
    winner = check_winner(board)
    if winner == 'X':
        return 10 - depth
    elif winner == 'O':
        return depth - 10
    elif winner == 'Tie':
        return 0

    ### your code here ###

    if is_maximizing:
        max_eval = float('-inf')
        for move in get_available_moves(board):
            board[move[0]][move[1]] = 'X'  # AI's move
            eval = minimax(board, depth + 1, False, alpha, beta)
            board[move[0]][move[1]] = ' '
            max_aeval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:  # β剪枝
                prunes += 1
                break
        return max_eval
    else:
        min_eval = float('inf')
        for move in get_available_moves(board):
            board[move[0]][move[1]] = 'O'  # Human's move
            eval = minimax(board, depth + 1, True, alpha, beta)
            board[move[0]][move[1]] = ' '
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:  # α剪枝
                prunes += 1
                break
```

```python
        return min_eval
    pass

    ######################

# Find the best move
def find_best_move(board):
    best_val = -math.inf
    best_move = None
    for move in get_available_moves(board):
        board[move[0]][move[1]] = 'X'
        move_val = minimax(board, 0, False, -math.inf, math.inf)
        board[move[0]][move[1]] = ' '
        if move_val > best_val:
            best_val = move_val
            best_move = move
    return best_move

# Human selects a move
def human_move(board):
    with output:
        display(move_row, move_col, move_button)

def on_move_button_clicked(b):
    row = move_row.value
    col = move_col.value
    if board[row][col] == ' ':
        board[row][col] = 'O'
        current_turn.value = 'AI'
        play_game()
    else:
        with output:
            display(HTML("<p>Invalid move. Try again.</p>"))
            human_move(board)

# Main game loop
def play_game():
    global prunes
    winner = check_winner(board)
    if winner:
        update_display()
        with output:
            if winner == 'X':
                display(HTML("<p>AI wins!</p>"))
            elif winner == 'O':
                display(HTML("<p>Human wins!</p>"))
            else:
                display(HTML("<p>It's a tie!</p>"))
            display(HTML(f"<p>Total number of prune times during this game: {prunes}</p>"))
        return

    if current_turn.value == 'AI':
        prunes_before_move = prunes
        move = find_best_move(board)
        if move:
            board[move[0]][move[1]] = 'X'
            prunes_during_move = prunes - prunes_before_move
            update_display()
            with output:
                display(HTML(f"<p>Number of prune times during AI's move: {prunes_during_move
            current_turn.value = 'Human'
            play_game()
    else:
        human_move(board)

def update_display():
```

```
    with output:
        clear_output(wait=True)
        display(HTML(f"<p>Current Board:</p><p>{print_board(board)}</p>"))

def main():
    with output:
        clear_output(wait=True)
        global board, prunes
        board = [[' ' for _ in range(3)] for _ in range(3)]
        prunes = 0
        display(HTML("<p>Initial Board:</p>"))
        display(HTML(f"<p>{print_board(board)}</p>"))
        play_game()

# Widgets for user input
move_row = widgets.BoundedIntText(value=0, min=0, max=2, description='Row:')
move_col = widgets.BoundedIntText(value=0, min=0, max=2, description='Col:')
move_button = widgets.Button(description="Make Move")
move_button.on_click(on_move_button_clicked)

first_player = widgets.RadioButtons(
    options=['Human', 'AI'],
    value='Human',
    description='Who goes first?'
)

def on_first_player_selected(b):
    current_turn.value = first_player.value
    main()

start_button = widgets.Button(description="Start Game")
start_button.on_click(on_first_player_selected)

# Store the current turn
current_turn = widgets.Text(value='Human', disabled=True)

# Output widget to manage and display output
output = widgets.Output()

# Display the interface
display(first_player, start_button, output)
```

```
RadioButtons(description='Who goes first?', options=('Human', 'AI'), value='Human')
Button(description='Start Game', style=ButtonStyle())
Output()
```

In [8]:

```
def auto_test(human_moves):
    global board, current_turn, prunes
    prunes = 0  # Reset prunes at the start of the game
    clear_output(wait=True)
    board = [[' ' for _ in range(3)] for _ in range(3)]
    current_turn.value = 'Human'
    move_index = 0

    print("Initial Board:")
    print_board(board)

    while True:
        winner = check_winner(board)
        if winner:
            if winner == 'X':
                print("AI wins!")
            elif winner == 'O':
                print("Human wins!")
            else:
                print("It's a tie!")
```

```python
                print(f"Total number of prune times during this game: {prunes}")  # Display the t
                break

            if current_turn.value == 'AI':
                prunes_before_move = prunes  # Store prunes before AI's move
                move = find_best_move(board)
                prunes_during_move = prunes - prunes_before_move  # Calculate prunes during this 
                if move:
                    board[move[0]][move[1]] = 'X'
                    print("\nAI ('X') moves:")
                    print_board(board)
                    print(f"Number of prune times during AI's move: {prunes_during_move}")
                    current_turn.value = 'Human'
            else:
                if move_index < len(human_moves):
                    move = human_moves[move_index]
                    row, col = move
                    if board[row][col] == ' ':
                        board[row][col] = 'O'
                        print(f"\nHuman ('O') moves to ({row}, {col}):")
                        print_board(board)
                        current_turn.value = 'AI'
                        move_index += 1
                    else:
                        print(f"Invalid move at ({row}, {col}). This cell is already occupied.")
                        break
                else:
                    print("No more moves left for the human.")
                    break

# Test example
test_moves = [(0, 0), (1, 0), (0, 2), (2, 1), (2, 2)]
auto_test(test_moves)
```

```
Initial Board:

Human ('O') moves to (0, 0):
The history saving thread hit an unexpected error (OperationalError('attempt to write a readon
ly database')).History will not be written to the database.

AI ('X') moves:
Number of prune times during AI's move: 2198

Human ('O') moves to (1, 0):

AI ('X') moves:
Number of prune times during AI's move: 91

Human ('O') moves to (0, 2):

AI ('X') moves:
Number of prune times during AI's move: 9

Human ('O') moves to (2, 1):

AI ('X') moves:
Number of prune times during AI's move: 0

Human ('O') moves to (2, 2):
It's a tie!
Total number of prune times during this game: 2298
```

In [ ]: