Narges-babaahmadi-610398102

Overview:
This code is a Connect Four game with minMax algorithm(+alpha-beta pruning) that can be played against an AI opponent. The game uses the Pygame library for graphics and user input. The game starts by initializing the board using the Board_init() function. The initial state of the board is then printed using the print_board(board) function. Pygame is then initialized, and the size of the window and the size of the squares on the board are defined. The board is then drawn on the screen using the draw_board(board) function. The turn order for the game is randomly decided, with either the player or the AI starting first. The game then enters a while loop that continues until the game is over.
If the game is over, the code waits for three seconds before quitting.

Important Functions:

win_pos_finder: It takes in a 2D board array and a piece value as inputs and returns whether or not the given piece has won the game. It checks for win conditions in four different ways: horizontally, vertically, positively sloped diagonals, and negatively sloped diagonals.
window_score: This function scores a given window based on the number of pieces belonging to the current player and the opponent. This function is used to evaluate the strength of the current player's position on the board.
Args:
window: A list of integers representing the pieces in a specific row, column, or diagonal.
current_piece: An integer representing the piece of the current player (either PLAYER_PIECE or AI_PIECE).
Returns:
score: An integer score for the given window, reflecting the strength of the current player's position.

find_score: This function calculates the score of the given game board and current piece. This score is used to determine the best move for the current player in the Connect Four game. The function starts by scoring the center column of the game board. It counts the number of occurrences of the current piece in the center column and multiplies it by 3. This is because having pieces in the center column is generally more advantageous.
Args:
game_board: A 2D numpy array that represents the current state of the game board.
current_piece: The current piece of the player whose score is being calculated.

The function returns the final calculated score, which is used to determine the best move for the current player.

minimax: The minimax function implements the **classic minimax algorithm with alpha-beta pruning** for improved performance. The algorithm calculates the best move for the AI player by searching the game tree and evaluating the terminal states of the game. The use of alpha-beta pruning helps reduce the number of nodes in the search tree and increase the efficiency of the algorithm.
The function first gets all the valid locations on the board where a piece can be placed, and checks if the game is already in a terminal state (i.e., either the AI wins, the player wins, or the game is a draw). If the game is in a terminal state or the depth of the search has reached 0, the function returns a score representing the value of the terminal state. If the game is won by the AI, the score is set to a large positive number. If the game is won by the player, the score is set to a large negative number. If the game is a draw, the score is set to 0.

Args:
board is the current state of the game board
depth is the maximum search depth
alpha and beta are the values used in alpha-beta pruning

maximizingPlayer is a boolean value that indicates whether it is the AI's turn to make a move or not

The function returns the selected column and the calculated value.

Implementing with neural networks:

The Connect Four game is played on a 6x7 board and the objective is to connect four of one's own chips in a row, either vertically, horizontally or diagonally. The two players take turns to drop their chips in the columns of the board.
The AI player uses a **neural network** model implemented in TensorFlow to make its moves.
The model has **three dense layers** with 64 and 32 neurons and **ReLU** activation functions, and a final output layer with a single neuron and a sigmoid activation function.
The model is compiled using the **Adam optimizer** and binary cross-entropy loss function.

The game is played using the **play_game()** function. The game starts by initializing the board to an empty 6x7 matrix. The first player to play is player 1 and the players take turns making their moves. If it's player 1's turn, they are prompted to input the column they want to drop their chip in. If it's the AI player's turn, the board is first flattened into a 1-dimensional array and passed as an input to the model to get a prediction of the column to drop the chip in. The prediction is rounded to the nearest integer and used as the column. The **play_move()** function is then called to play the move on the board. The game continues until the **is_game_over()** function returns True, indicating that the game is over. The final board is displayed and a message indicating that the game is over is printed.