CSE 4082 Artificial Intelligence – Project 2 Report

Group members:

Emirkan Karabulut 150118052
Ramazan Karkin 150119512

The function game_board() : Creates and returns the initial game board, which is filled with -1.

The function move_piece(game_board, player, column): Allows a player to add a piece to a specific column of the board, if that column is not already full.

The function win_condition(game_board, player): Checks whether the player has won by having four consecutive pieces either horizontally, vertically, diagonally or anti diagonally.

The function print_board(board): This function after printing the Connect-Four game playfield to the console. According to the moves of the players, it will print the moves to the console by using the 'X' symbol for the first player and the 'O' symbol for the second player.

The function is_playable((game_board, column): it will check whether the column passed as a parameter is a valid move or not.

The function choice(board): It will check board for valid move and if there is a valid move then it will choose a random column.

The function evaluate_heuristic1(board, player): First two if statement checks If player1 has won the function returns positive infinity, and if player2 has won returns negative infinity. Returning positive or negative infinity effectively assigns the highest possible or lowest possible score to the board state. It is used to evaluate the current board state and assign a score to it, based on sequences of length. The function iterates over each cell in the board and checks if the cell belongs to the current player. If the cell belongs to the current player, the function checks the horizontal, vertical, diagonal, and anti-diagonal sequences of length 3 in the board and adds 1 to the score for each sequence that can be completed by the current player, by comparing the next cells with current cell and current player. The minimax function uses the scores returned by evaluate_heuristic1 to make decisions on the best move to make, by maximizing the score for the current player and minimizing the score for the opposing player.

The function evaluate_heuristic2(board, player): This heuristic function uses a weight matrix to assign a score to each cell on the board.We have given weight to each position according to the positions we think are more important, with higher weights given to cells in the center of the board. This way, the AI player will prioritize moves that are made in the center of the board.

The function evaluate_heuristic3(board, player): This evaluate function uses heuristic2 and the number of occurrences of two adjacent same values (either the current player or the opponent) horizontally, vertically, or diagonally on the board. After heuristic2 assigned a score to each cell on the board according to the weight matrix, then for each occurrence of two adjacent same values, we will triple the score. This function is used in the minimax algorithm as a way to evaluate the potential outcomes of a given board state and determine the best move to make.

The function minimax(board, player, depth=4, alpha=-float("inf"), beta=float("inf"), eval_func=evaluate_heuristic1): It is an implementation of the Minimax algorithm.The algorithm uses this score to determine the best move to make by maximizing the score for the current player and minimizing the score for the opposing player. The implementation approach which we applied is by recursively calling the minimax function for each playable column on the board, and finding the alpha(best score for the maximizer) and beta(best score for the minimizer) values which are used to prune the search tree to avoid unnecessary work. The function takes the current state of the board, the player that is making the move, alpha, beta, and evaluation function as inputs.

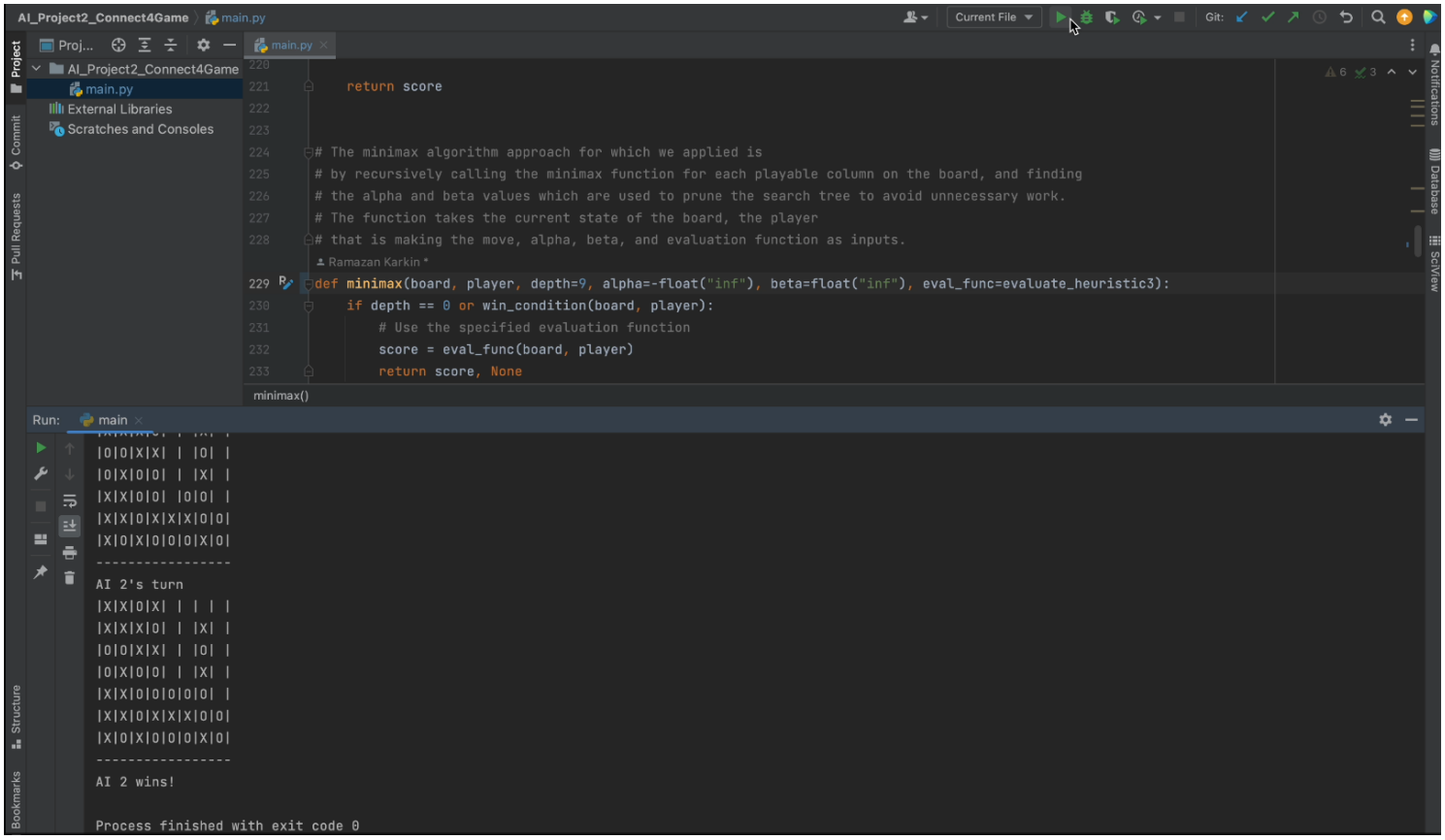
The function starts by checking if the search has reached the maximum search depth specified by the depth parameter, or if the current player has won the game. If either of these conditions is true, the function calls the evaluation function passed in eval_func parameter, which in this case is evaluate_heuristic1, to return the heuristic score of the current board state for the given player.

If the search is not at the maximum depth or the game is not over, the function iterates through all columns of the board, and for each playable column, it performs the following steps:

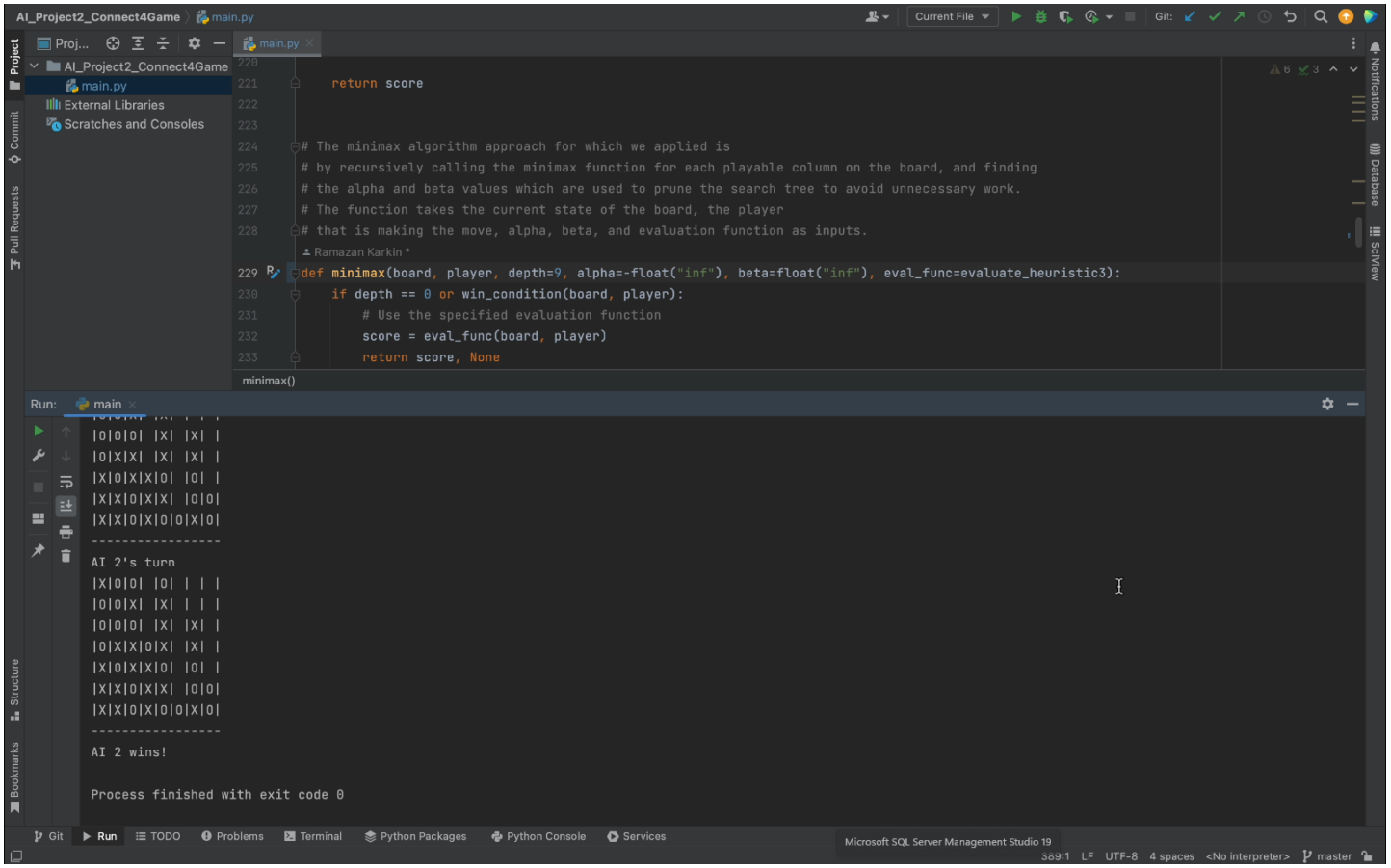
it creates a copy of the board. Then the move_piece(temp_board, player, column) function to make a move in that column. After that it recursively calls the minimax function with the modified copy of the board and the other player as the active player. it negates the returned score of the recursive call. if the current score is greater than the current alpha the function updates the alpha and best_column with the current score and column. In case when alpha is greater than or equal to beta, the function breaks out of the loop return the alpha and best_column. In the case where no valid move is found, the function calls choice(board) which selects a random playable column.

In summary, the function determine the best move to make by simulating all possible moves to a certain depth and choosing the move that results in the best score according to the evaluation function passed.

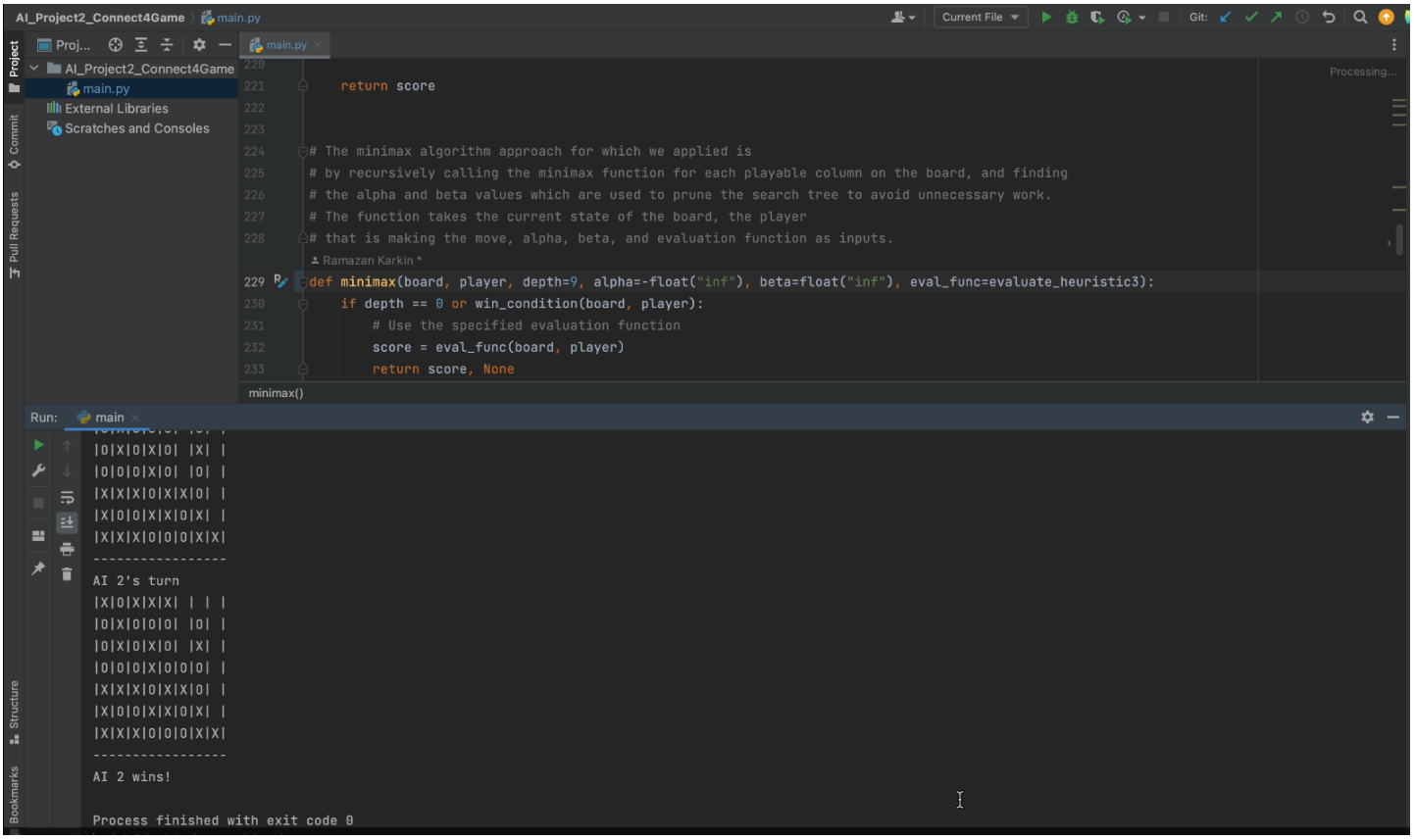
Video link: https://vimeo.com/789315930
drive link:
https://drive.google.com/drive/u/0/folders/1Mr42mv32CSYlqDVdMmwVrw4OaG5pH82c
AI player using h1 vs AI player h2: Video duration time 0.00 - 8.34 you can see



AI player using h2 vs AI player using h3: Video duration time 8.53 - 30.00



AI player using h1 vs AI player using h3: Video duration time 30.15 - 37.37



Human Player vs the Best AI Player Configuration:Video duration time 37.53 - 39.30