A logo with a blue circle and red stripes

Description automatically generated

**T.C.**

**MARMARA UNIVERSITY**

**FACULTY of ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**CSE 4082– Project 3**

Group Members

150119052 Sadık AKGEDİK

150120518 Evren KOYMAT

150118008 Osman Tunç POYRAZ

**1. Problem description**

The project aims to implement a Python script for playing the SOS game, where human players can play against an AI opponent utilizing the minimax algorithm. The SOS game is played on a 5x5 board initially filled with blanks, with 'S' written on each corner. Players take turns placing either 'S' or 'O' on an empty cell, and the goal is to form the word "SOS" horizontally, vertically, or diagonally. The AI opponent uses the minimax algorithm to make intelligent moves, while the human player provides their moves through user input.

**2. Classes**

**Node:** Represents a node in the game tree with attributes such as the current board state, scores for both players, and a list of child nodes.

A computer screen shot of a program

Description automatically generated

**3. Methods and Functions**

* **sos\_game():** Initializes the SOS game board.
* **players():** Creates a list of players, initially set as ('Player 1', 0) and ('Player 2', 0).
* **print\_board(board):** Prints the current state of the game board.
* **get\_move():** Takes user input for a move (row, column, letter) and returns it as a list.
* **check\_move(move, board):** Checks if the move is valid on the current board.
* **update\_board(move, board):** Updates the board with the given move.
* **add\_player\_SOS\_move(...):** Adds a player's move to the corresponding SOS list.
* **check\_sos\_human(move, board, player):** Checks if the move creates an "SOS" pattern for the human player.
* **check\_sos(node, move, board, player):** Checks if the move creates an "SOS" pattern for the AI player.
* **sos\_game\_start():** Initiates the SOS game, allowing players to take turns and updating the board accordingly.
* **count\_sos(board):** Counts the occurrences of "SOS" patterns on the board.
* **heuristic\_one(node, maximizing\_player):** Evaluates the board based on the difference in scores between players.
* **heuristic\_two():**-
* **generate\_possible\_moves(board):** Generates a list of possible moves on the current board.
* **undo\_move(move, board, current\_player):** Undoes the given move on the board.
* **ai\_move(board):** Utilizes the minimax algorithm to determine the AI's move.
* **minimax(node,depth, maximizing\_player, alpha, beta):**

sdas

**4. User Interface**

The user interacts with the script through the console. For each turn, the current state of the board is displayed, prompting the player (human or AI) to make a move. Invalid moves are handled with appropriate messages. The game continues until the board is full, and the final scores are displayed.

**5. Execution Flow**

* The SOS game is initialized with a 5x5 board.
* Players are created with initial scores.
* Players take turns making moves, either through user input or AI moves.
* The board is updated after each move.
* "SOS" patterns are checked and scores are updated accordingly.
* The game continues until the board is full.
* The final scores are displayed.

**6. The Output**

The output consists of the current state of the board after each move, informing the players of their scores and displaying any "SOS" patterns found. The game concludes with a summary of the final scores, indicating the winner or a possible tie. The AI opponent's moves showcase its strategic decision-making using the minimax algorithm.

A screen shot of a computer program

Description automatically generated A computer screen shot of a game

Description automatically generated

Above is an example output at depth level-1. Our program only works at depth level one and there is only an ai vs human mode in it. The heuristic used is explained before as: Evaluates the board based on the difference in scores between players.