

Artificial Intelligence

Laboratory 2

Variant 1

Our code represents a simple command-line implementation of the classic Tic Tac Toe game using Python. The game is played between a player (represented by "X") and a computer (represented by "O"). The board is represented as a 3x3 matrix, where each cell of the matrix can be either empty or contain a player's move.

The main program is divided into several functions, including:

- **draw_board(board)**: A function that takes the current board as input and prints it to the console in a visually pleasing format. The board is represented using Unicode characters to draw the border around the cells.
- **is_valid_move(board, row, col)**: A function that takes the current board and a row and column position as input and returns whether the given move is valid (i.e., the cell is empty).
- **make_move(board, row, col, player)**: A function that takes the current board, a row and column position, and the current player as input and makes the move on the board.
- **is_game_over(board)**: A function that takes the current board as input and returns whether the game is over (i.e., either a player has won or the board is full).
- **get_score(board)**: A function that takes the current board as input and returns the score for the board. The score is determined by checking if any player has won the game, with a score of 1 for the computer and -1 for the player if the computer wins, and a score of -1 for the computer and 1 for the player if the player wins. If there is no winner, the score is 0.
- **get_best_move(board, depth, alpha, beta, player)**: A function that takes the current board, the depth of the search (i.e., the maximum number of moves ahead to consider), the alpha and beta values for alpha-beta pruning, and the current player as input and returns the best move and score for the given player. The function recursively searches through the game tree to find the best move for the player using the minimax algorithm with alpha-beta pruning.
- **play_game()**: The main function that runs the game. It initializes the board, and then loops through each turn, alternating between the player and the computer. In each turn, it first draws the board, then prompts the player for their move if it's their turn, and makes the move if it's valid. If the game is over, it prints the final board and the result of the game.

Players take turns placing their symbol (X or O) on an empty space on the board, X goes first.

The game ends when one player gets three in a row (horizontally, vertically, or diagonally), or when all the spaces on the board are filled.

To make a move, you will be prompted to enter the row and column of the space you want to place your symbol in. Rows and columns are numbered from 1 to 3.

The computer will play as the second player (O), and will make its moves automatically.

The computer is not always making the smartest move because the `get_best_move()` function uses a minimax algorithm with alpha-beta pruning to search for the best move, but the depth of the search is limited to a fixed value of 3. This means that the computer will not be able to see more than 3 moves ahead, which can limit its ability to make the smartest move in some situations.

Additionally, the function also uses a heuristic score function that checks for wins and losses in the current board state, but it does not consider other factors that can affect the game, such as the placement of pieces on the board or potential future moves. This can also limit the computer's ability to make the smartest move.

Finally, the function randomly selects one of the best moves if there are multiple moves with the same score. This can sometimes result in the computer making a suboptimal move.

The game starts by asking the user if he want to play with ('X') or ('O'). If the player chooses to be ('X') he plays first. The program displays an empty board and is asking for a player to choose a row and a column between 1-3:

```
Do you want to be X or O? x

  |  |
  --
  |  |
  --
  |  |

Enter row (1-3): 3
Enter column (1-3): 3
```

After that the program prints the player's move and after that it's the computer's turn to make a move.

```
Enter row (1-3): 3
Enter column (1-3): 3

  |  |
  --
  |  |
  --
  |  X

Computer is thinking...

O  |  |
  --
  |  |
  --
  |  X
```

The game continues until one of the players wins or the game is a tie. Then it displays a message about the result before the program finishes.