Course: Artificial Intelligence

Project 2: Game Construction

Name: Konstantinos Vardakas

Student ID: 522

Email: pcs0522@uoi.gr

Introduction

This report describes the implementation of a program in C language for a two-player game (MAX and MIN) on a 4x3 grid. The program allows the MAX player (computer) to play optimally against the MIN player (user) using the MINIMAX algorithm with recursion.

Definition of Game State

The state of the game is represented by a 4x3 grid, implemented as a two-dimensional array char board[ROWS][COLS], where ROWS=4 and COLS=3. Each cell of the grid can contain:

- X: Placed by player MAX.
- O: Placed by player MIN.
- _: Empty space.

The initial state of the game is defined to include an "X" and an "O" in random, non-consecutive positions (horizontal, vertical, or diagonal), reducing the state space. The non-adjacency condition is checked with the is_adjacent function, which uses the Chebyshev distance (max(|x1, x2|), max(|y1, y2|)) to ensure that the positions are not adjacent.

The players take turns, with MAX playing first, placing the corresponding symbol in any empty space. The game ends when at least one trio of XOX or OXO is formed (horizontally, vertically, or diagonally) or when the grid is filled without trios (tie).

Final State Evaluation

The evaluation of the final states in the game is based on the number of XOX and OXO triplets formed on the grid. Specifically, the state is considered a win for the MAX player when the number of XOX triplets exceeds the number of OXO triplets, with a value of +10. Conversely, the situation is considered a win for the MIN player when the number of OXO triplets is greater than the number of XOX triplets, with a value of -10. In the event of a tie, i.e. when the number of XOX triplets is equal to the number of OXO triplets or when the grid is complete with no triplets, the evaluation value is 0.

The evaluation process is implemented through the is_game_over function, which examines the grid to count all XOX and OXO triplets. The function returns 1 for a MAX win, -1 for a MIN win, 2 for a tie, or 0 if the game continues (i.e., there are no triplets and the grid is not complete). The evaluation values +10, -10, and 0 were chosen to clearly distinguish between win, loss, and draw situations, while being compatible with the MINIMAX algorithm, which seeks to maximize the value for MAX and minimize it for MIN.

Implementation of MiniMax

The implementation of the Minimax algorithm is based on the minimax(int is_max) function, which implements recursive exploration of all possible grid states to select the optimal move. At the beginning of each call, is_game_over is called, which checks whether the game has ended, returning a positive value for a MAX player win, a negative value for a MIN win, and zero for a draw. These values are used to evaluate the leaves in the game tree.

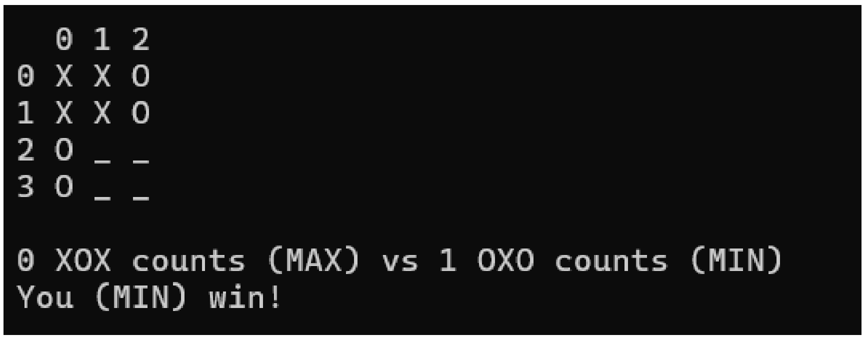
The algorithm works by switching roles between the player who maximizes the score (MAX, program) and the player who minimizes it (MIN, user). For each move, the function tests all available positions, temporarily places the corresponding symbol, recursively calls minimax for the next situation, and then cancels the move by replacing the character "_" in that cell. MAX selects the maximum value from the available moves, while MIN selects the minimum, thus forming the optimal strategy for each player.

The computer_move() function uses the Minimax algorithm to select the best possible move for the computer, which plays as player "X." For each available position on the grid, it temporarily places the symbol "X" and calls the minimax(0) function to calculate the performance of this possible move, assuming that the next player (the user, MIN) will try to minimize the score. After evaluation, the move is canceled and the best one in terms of score is retained. Finally, the computer places the "X" in the position with the highest score, ensuring an optimal strategy based on Minimax's predictions.

Possible terminal situations

The following are the possible scenarios for the final situation:

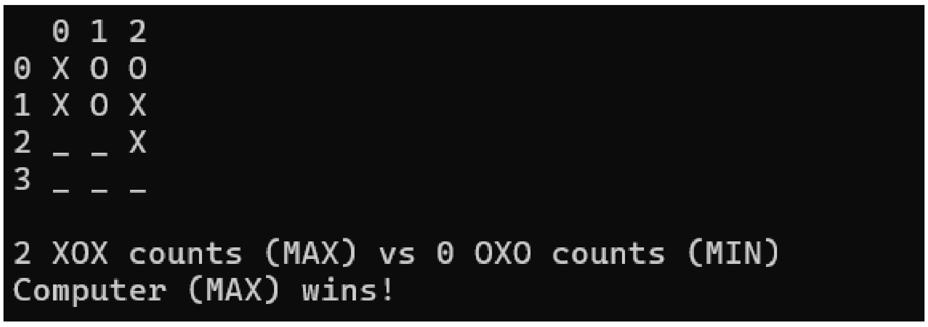
1) Player wins (MIN)



```
  0 1 2
0 X X O
1 X X O
2 O _ _
3 O _ _

0 XOX counts (MAX) vs 1 OXO counts (MIN)
You (MIN) win!
```

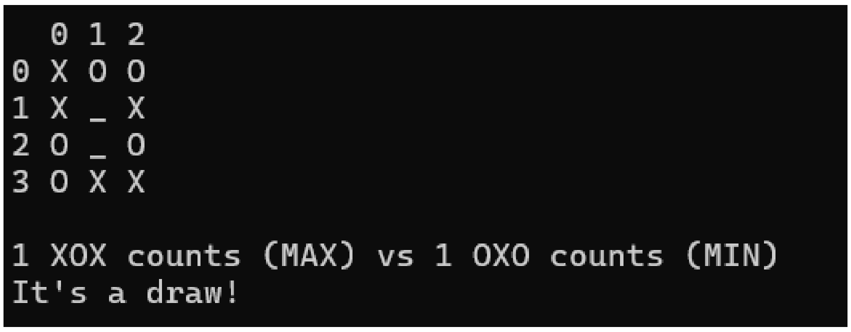
2) Program wins (MAX)



```
  0 1 2
0 X O O
1 X O X
2 _ _ X
3 _ _ _

2 XOX counts (MAX) vs 0 OXO counts (MIN)
Computer (MAX) wins!
```

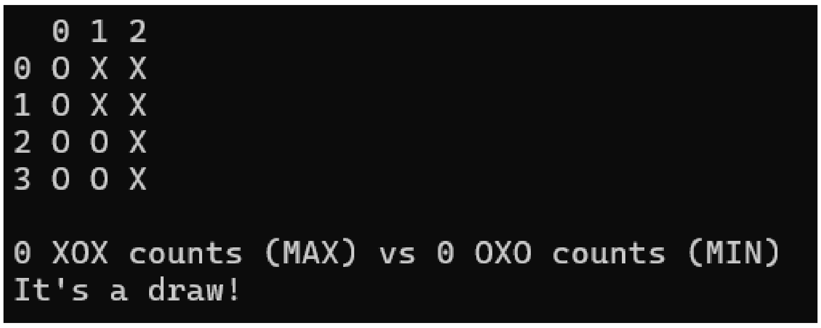
3) Draw due to equal numbers of XOX and OXO



```
  0 1 2
0 X O O
1 X _ X
2 O _ O
3 O X X

1 XOX counts (MAX) vs 1 OXO counts (MIN)
It's a draw!
```

4) Draw due to absence of XOX or OXO



```
  0 1 2
0 O X X
1 O X X
2 O O X
3 O O X

0 XOX counts (MAX) vs 0 OXO counts (MIN)
It's a draw!
```