David Vences
801045180 ITCS-6150

3D Tic-Tac-Toe Project

Game Background
This tic-tac-toe game is a two-player game, where one player is an AI player. This implementation of the game is three dimensional and consists of a 4x4x4 grid. The game starts with an empty grid and players take turns positioning their symbols on the board until one player wins or the game ends in a tie. The player who can align four of their symbols in a straight line is considered the winner.

Game Rules
The human player will be player X and will get to play the first move. The AI player will be player O and its move will be calculated after player X plays their move.

The ends if any of the following cases are achieved:

- A player, after making their move, has 4 of their symbols in a straight line and therefore is considered the winner.
- The board is full, and no player was able to achieve having 4 of their symbols in a straight line.

Usage
This program runs in browser; Simply download the project folder and open the index.html file in a browser.

System Flow
Below is a high-level overview of the system flow of the game. The program starts by asking the user what game difficulty they would like to play at. Once the user chooses a difficulty the game board will load and wait for the users first move. Once the first move is made the game begins. Placing a move will do two things:

1. Place the players piece on the board, both on the front end and the back end.
2. Check the current board state to see if the game is over by either one of the two players winning or by a tie. A tie would mean that the board is full, and no player won.

When the human player places their symbol on the board it will trigger the AIMove function. This function is responsible for calling the minimax function and sending back what move should be made. The minimax function is based on the minimax algorithm and is the main engine that runs the AI.

The minimax algorithm is often sued as a recursive strategy for two player games where players take turns making their move. Players make their moves with the aim of maximizing the minimum value of their opponent's future moves. The implementation of the minimax algorithm used for this game uses alpha-beta pruning in order to "prune" any branches that we know won't result in a better score. This algorithm uses alpha and beta values. The alpha value represents the highest score the maximizing player receives, and the beta value represents the lowest value the minimizing player receives. Once the beta value is less than or equal the alpha value then it is an indication that the move is not favorable for the parent node and prunes the remaining branches of that node.

David Vences
801045180

Pseudocode of the minimax algorithm used:

```
function miniMax(Board, depth, maximizingPlayer , alpha, beta)
    if depth is met or node is a terminal node
        return the value of node

    if maximizingPlayer
        bestValue = negative infinity
        for each child of node
            value = miniMax (board, depth + 1, false , alpha, beta,)
            bestValue = max(bestValue, value)
            alpha = max(alpha, bestValue)
            if beta <= alpha
                break // Prune
        return bestValue

    else
        bestValue = positive infinity
        for each child of node
            value = miniMax (board, depth + 1, true , alpha, beta)
            bestValue = min(bestValue, value)
            beta = min(beta, bestValue)
            if beta <= alpha
                break // Prune
        return bestValue
```

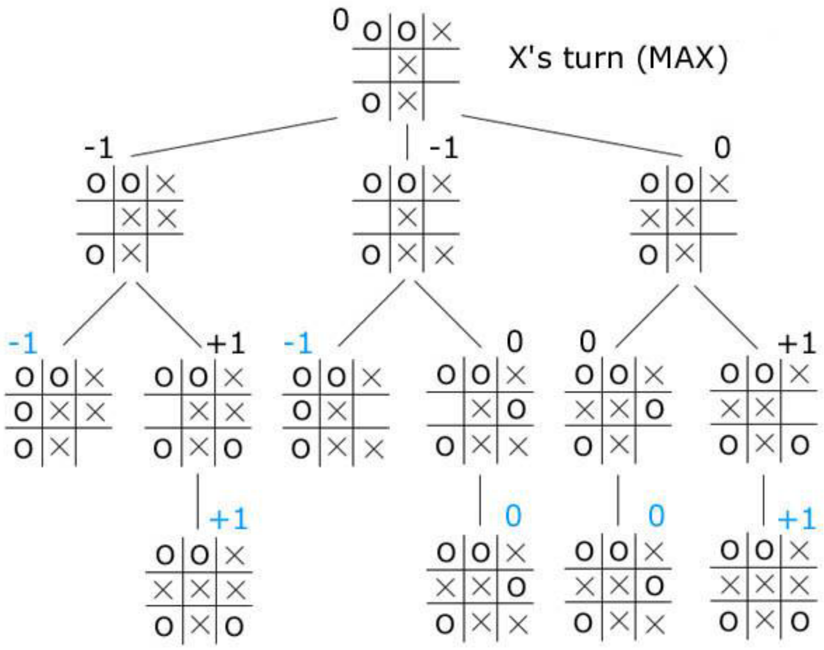


Figure 1. Visual representation of minimax algorithm with alpha beta pruning.

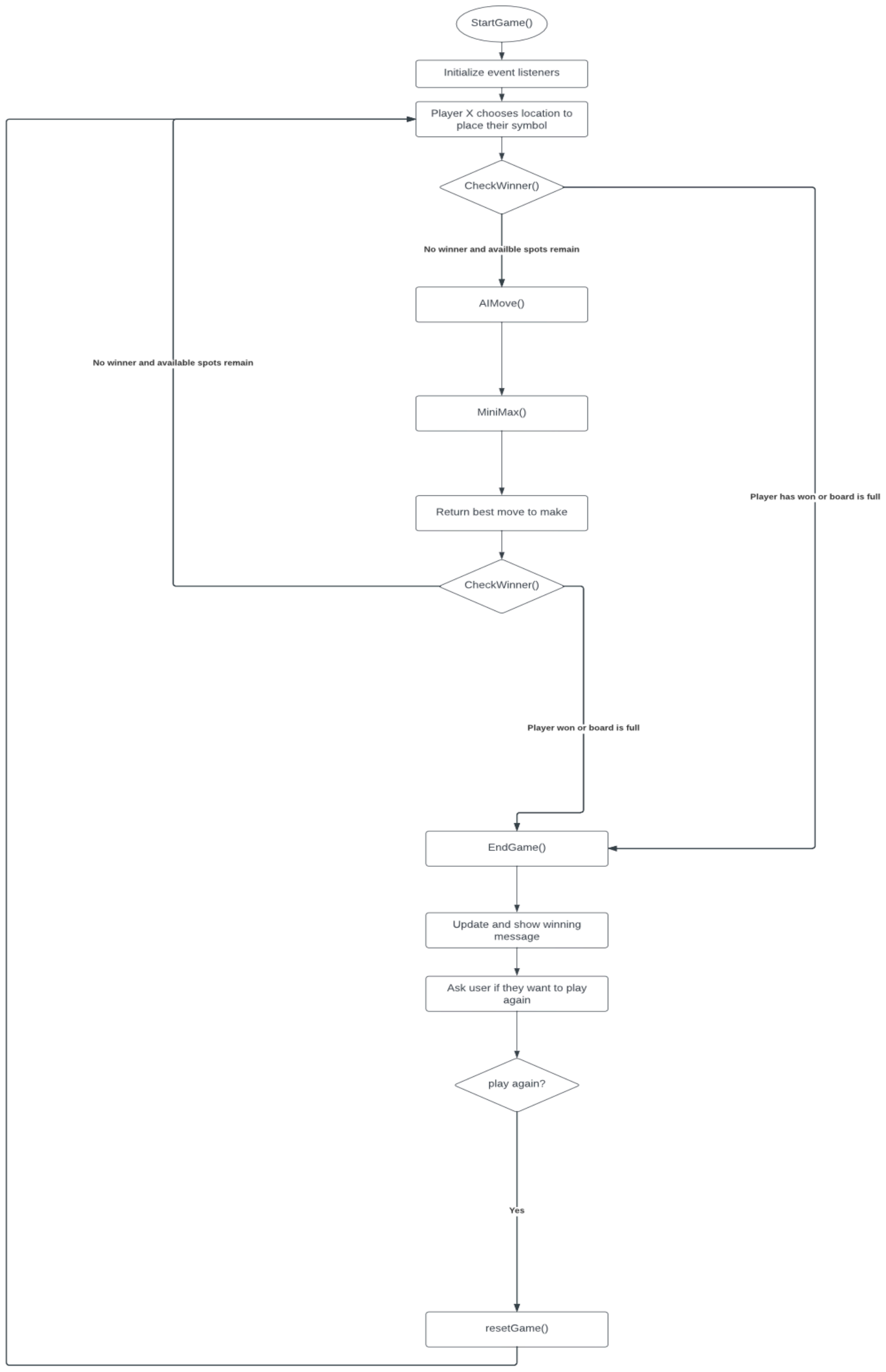

StartGame()
Initialize event listeners
Player X chooses location to place their symbol
CheckWinner()
No winner and availble spots remain
AIMove()
MiniMax()
Return best move to make
CheckWinner()
No winner and available spots remain
Player has won or board is full
Player won or board is full
EndGame()
Update and show winning message
Ask user if they want to play again
play again?
Yes
resetGame()

Figure 2. System flow diagram

Data Flow

The external entities for the data flow of this game are the human player and the AI Player. The processes that take place during the game are AIMove(), CalculateHash(), CheckWinner(), MiniMax(), and ResetGame(). The data stores used by the game are xoBoard_0, xoBoard_1, xoBoard_2, xoBoard_3, and the TranspositionalTable. The game starts once the human player makes the first move, once the human players move is chosen the game board where the symbol was placed will be updated to reflect the change. Next, the program will calculate the AI's next move. The AIMove function utilizes the minimax function in order to get the best move. In order to reduce the number of times that each board needs to evaluate the program uses a transpositional table. Once a board that has been evaluated for the first time and a score has been given a hash of the table will be created and added to the transpositional table, along with its depth and score. The next time the same board needs to be checked it will get the result instantly from the transpositional table rather than having to reevaluate the same board multiple times. See diagram below for an overview of the data flow.

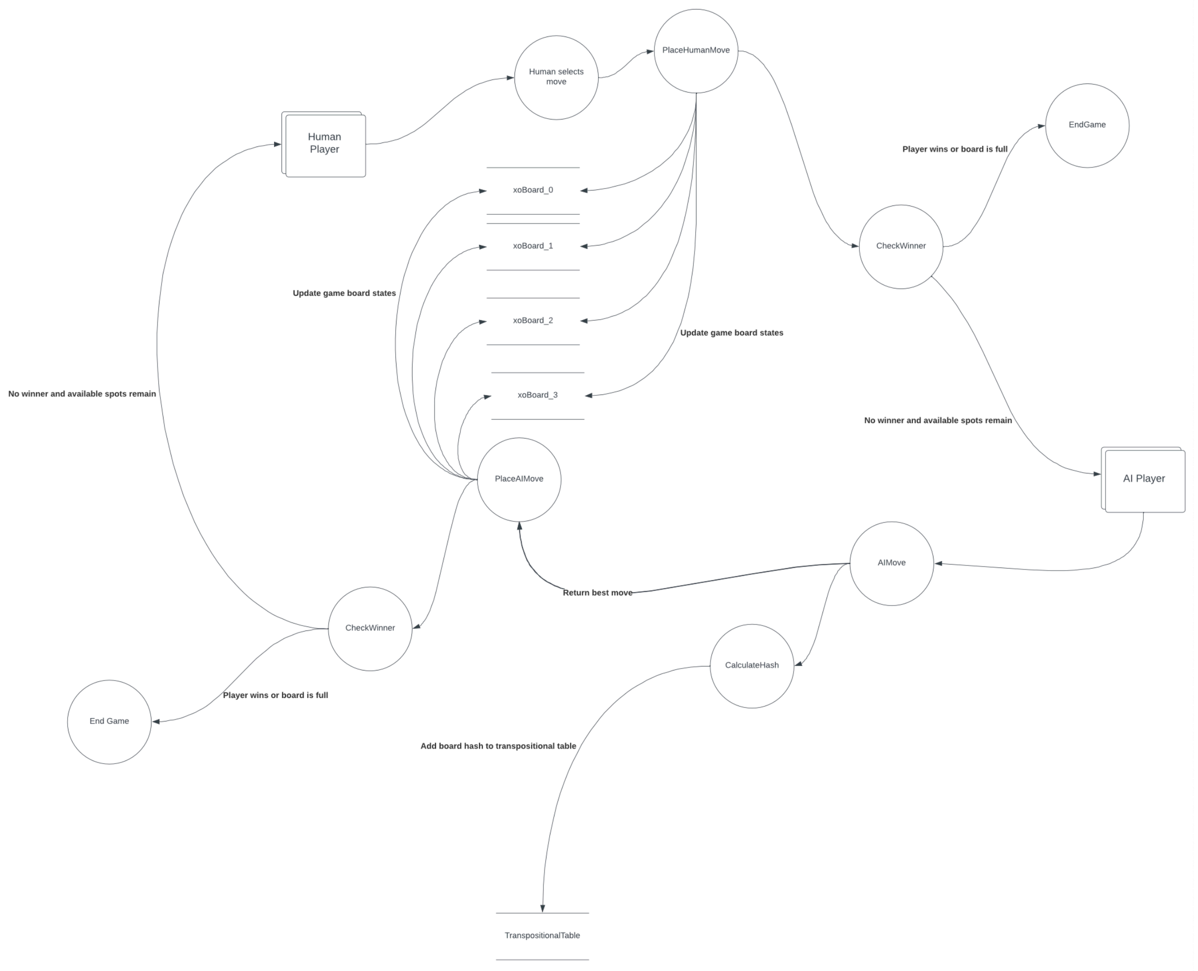


Figure 3. Data Flow Diagram

Input Design
The human player hovers over the board to see possible locations for their symbol, the letter "X", and then chooses a spot to place it. Once the symbol is placed the AI will calculate their move and place its symbol as well. See below for images of input process and design.

- The game starts with a pop up that asks the user to choose their difficulty:

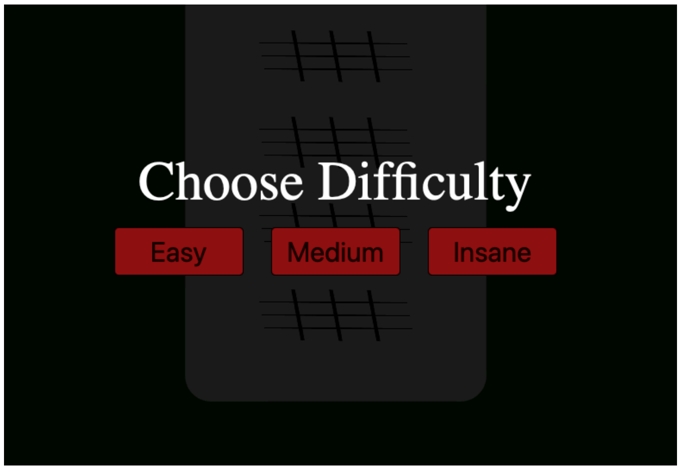
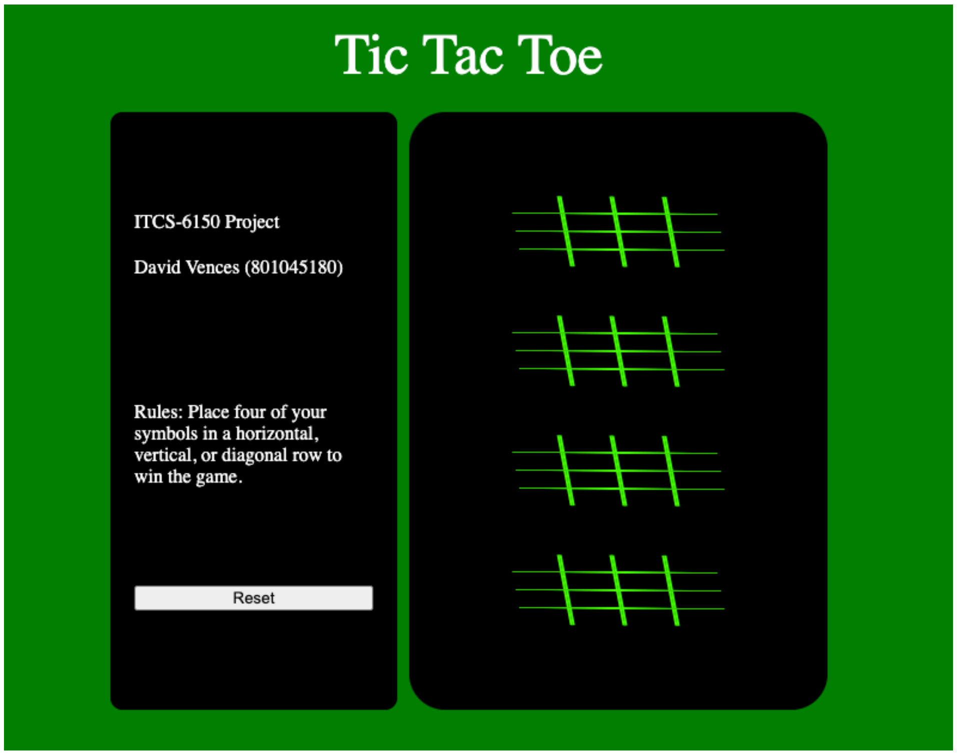


Figure 4. Start screen asking user to choose difficulty.

- Once a difficulty has been chosen the game begins with an empty 4x4x4 board.



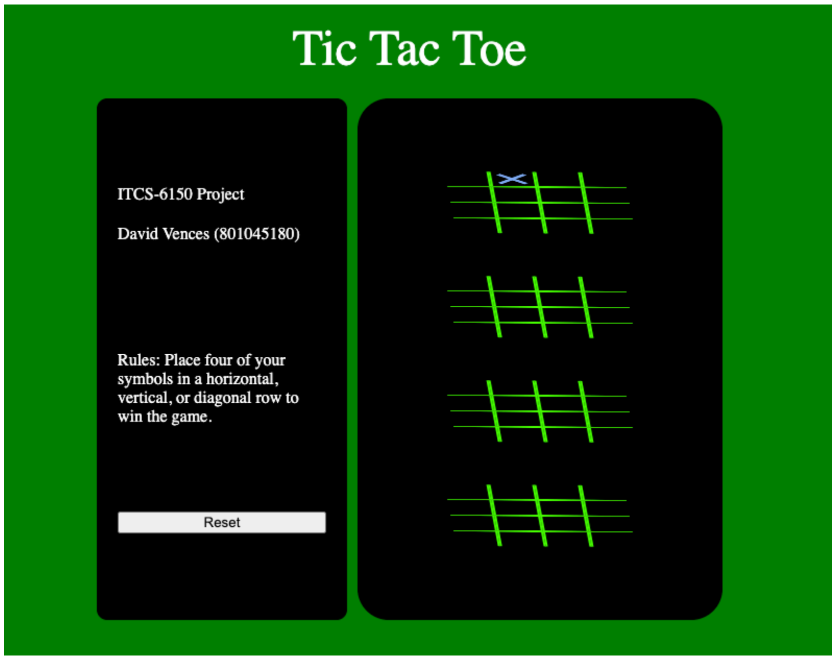
ITCS-6150 Project

David Vences (801045180)

Rules: Place four of your symbols in a horizontal, vertical, or diagonal row to win the game.

Figure 5. Empty starting board.

- Whenever the human player hovers over an open spot on the board a “shadow” of their symbol appears on the board to signify that the position is available to choose.



Tic Tac Toe

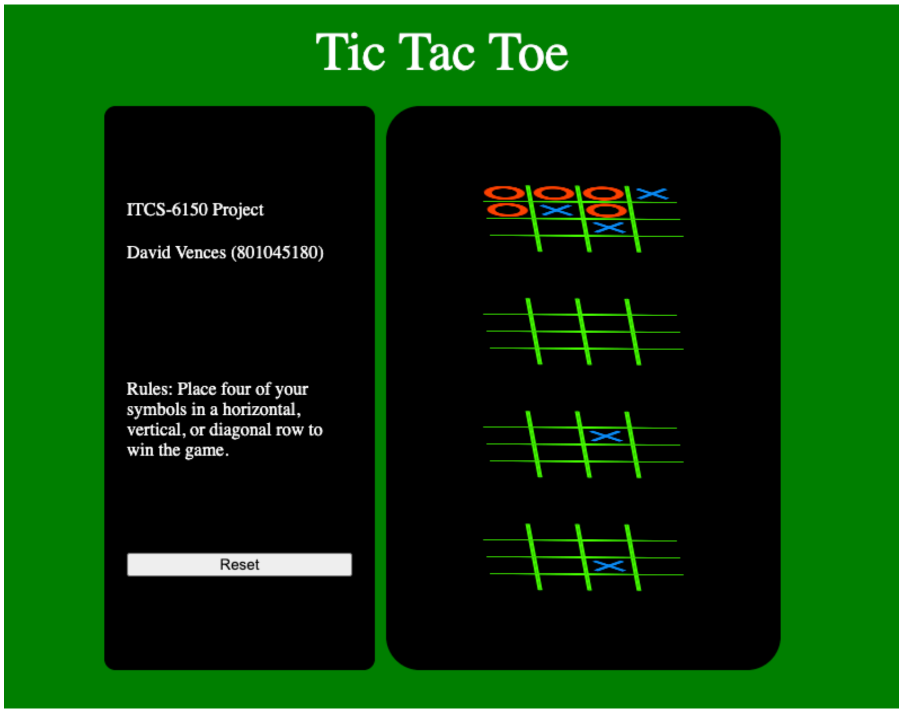
ITCS-6150 Project

David Vences (801045180)

Rules: Place four of your symbols in a horizontal, vertical, or diagonal row to win the game.

Figure 6. Hovering over an open spot shows the players symbol’s “shadow”.

- Once each player has made several moves the board will start to fill up:



Tic Tac Toe

ITCS-6150 Project

David Vences (801045180)

Rules: Place four of your symbols in a horizontal, vertical, or diagonal row to win the game.

Figure 7. Board state after several moves by each player.

Output Design

Once the game has ended, by either a player's win or a tie, a pop up will appear stating the result of the game and asking the player to choose a difficulty. Clicking on one of the difficulty levels will reset the board and start a new game with the chosen difficulty.

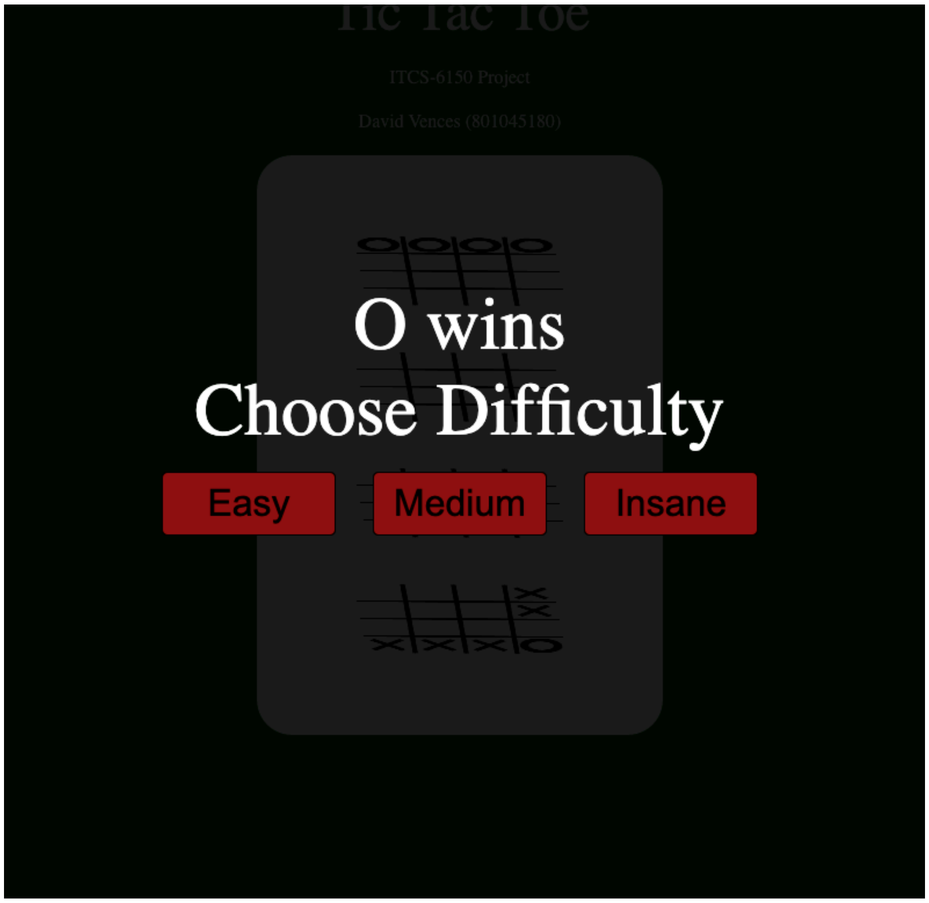


Figure 8. Pop up that appears once the game has ended.