An Unbeatable Tic Tac Toe game powered by AI

By Team MARTIANS

Team Members : Gowri Shankar B
V Hemantha Sandhya
Mentor : Shriladha Balasubramanian

Mission : Entertain the Mars Crew with an unbeatable Tic Tac Toe game powered by AI

- We have used the Minimax algorithm to implement the intelligence of the Computer Player. The different levels of difficulty make the game more interesting and enjoyable!
- We have also modified the program in such a way that the crew is given an option to choose between playing against the algorithm and competing against one another.

Changes made to the UI :

1. Included a header page with the title of the game and an option for the crew to choose the number of players.
2. Added an optional input field for the crew to provide their name
3. Added an automatic Response that displays the outcome at the end of the game
4. Added an option at the end of both the game pages which when clicked scrolls the page to the top(i.e, the header page)
5. Added an additional page for two player game
    a. With two input fields
    b. Another human image instead of a robot

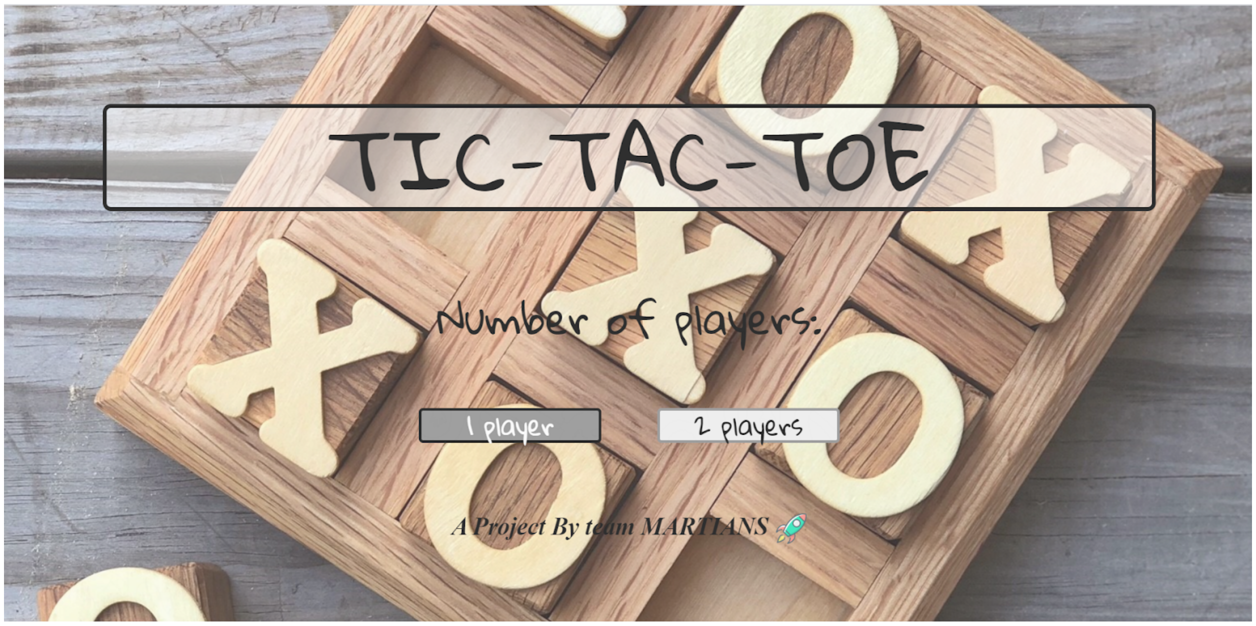
c. Removed the difficulty level field since the crew is competing against one another

Content on the front end :
Files : index.html , Style.css(stylesheet)

Contains three navigable pages :

<u>Page-1 : The header page</u>

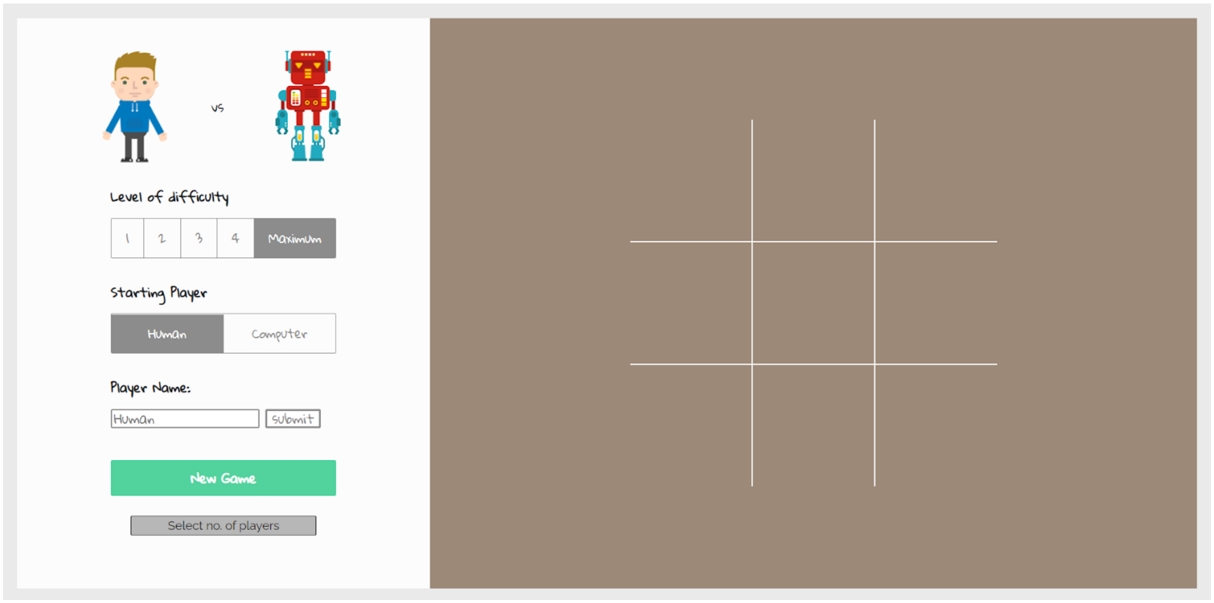
- Title of the game
- Buttons to select number of players



Page 1(The header page)

Page-2 : 1 player game

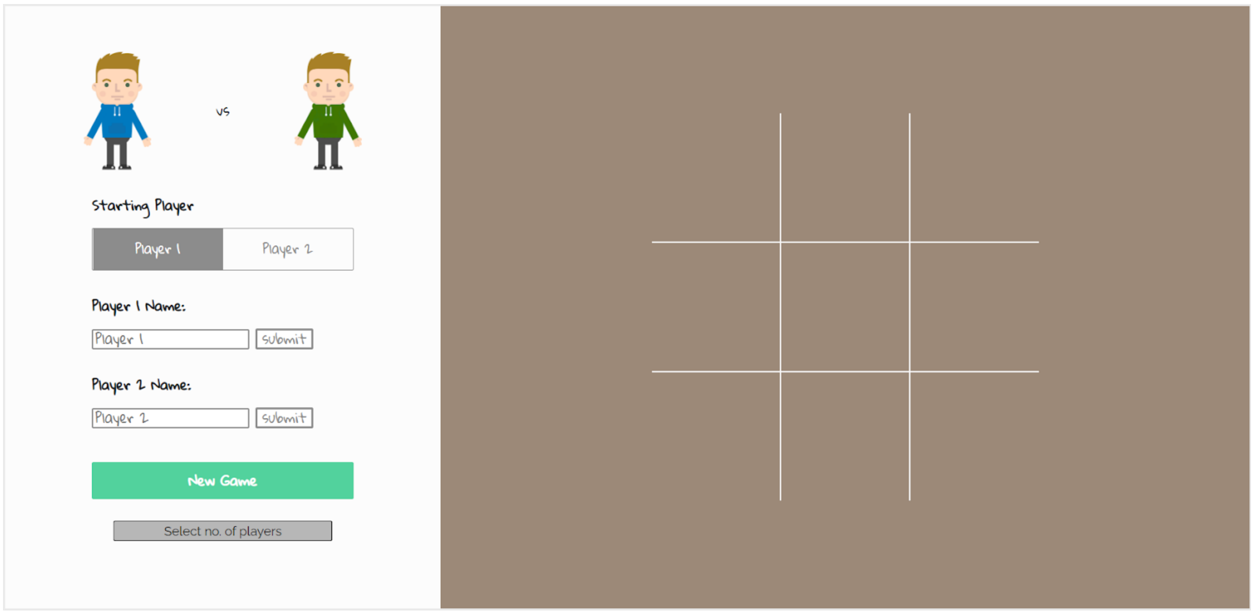
- Images of a human and a robot whose functionalities are coded in the backend
- Difficulty level field to choose the difficulty level of the game
- Starting player field to select who starts the game
- An input field to provide the player name
- New game button which resets the game with selected options
- No. of players field which helps the player to navigate to the header page
- Board which is where the game is visualised
- A response field which is initially hidden and made visible only at the end of game to display the result of the game



Page 2 (1 player game)

Page-3 : 2 player game

- Images of 2 humans whose functionalities are coded in the backend
- Starting player field to select who starts the game
- Two input fields to provide the player names
- New game button which resets the game with selected options
- No. of players field which helps the player to navigate to the header page
- Board which is where the game is visualised
- A response field which is initially hidden and made visible only at the end of game to display the result of the game



Page 3 (2 player game)

All the stylings and animations of these elements are done in stylesheet - Style.css

Backend files :

1. board.js :

    - It contains a class called move which defines a move object with 2 attributes : row and col for row number and column number of the grid
    - Contains a class board which gives the functionalities of the board
        - Attributes : (grid: 2-D array)(win_row: character)(win_direction: character)
        - Functions :

Function name	Return value/functionality	Parameters

printBoard()	Logs the current state of formatted board to the console	None
isEmpty()	Checks if the board is empty and returns a boolean value	None
isFull()	Checks if the board has no grids available and returns a boolean value	None
insert()	Updates the board with given symbol in the given grid	symbol:character , i:row, j:column
remove()	Removes the symbol in the given grid from the board	i :row, j: column
getPossibleMoves()	It returns all the empty grids of the board(array of objects of class move)	None
evaluate()	Returns the score of the board: 10 - if computer wins -10 - if human wins 0 - otherwise	None

2. computer.js :

    - Contains a class computer which has the minimax algorithm, the base of the project

- This class has 1 attribute max_depth which defines the maximum depth that the algorithm should work until. This enables us to give the difficulty level for the game.
- It also has two functions :
    - findBestMove() :
        - This function evaluates all the available moves using minimax() and then returns the best move/moves the maximizer(computer player here) can make.
        - It takes a board object as parameter
    - minimax() :
        - This function considers all the possible ways the game can go from the present state and returns the best value for that move, <u>assuming the opponent also plays optimally</u>

        - While returning, to make the AI smarter current depth is subtracted from the value of best move and then returned. This makes sure that the algorithm chooses the victory which takes the least number of moves and in case of a loss it will try to prolong the game and play as many moves as possible.

        - It takes three parameters :
            - Board - board object
            - depth - present depth
            - isMax - a boolean variable that tells if it’s the turn of maximizer or not

3. main.js :

- Contains the code which links the UI and the algorithm(AI). It has responsive functions for the activity done on the html page
- It is like the AI agent program that has agent functions that map any given percept sequence to an action.
- Since this is a software agent, clicks,keystrokes and scrolls are its sensory inputs and it reacts accordingly through the displaying on the screen, writing files etc.(UI changes)
- It has helper functions that add or remove(maintain) classes to different html elements.(Taken from http://jaketrent.com/post/addremove-classes-raw-javascript/)
- Functions to draw the winning lines when some player wins by adding classes to the corresponding elements for animation

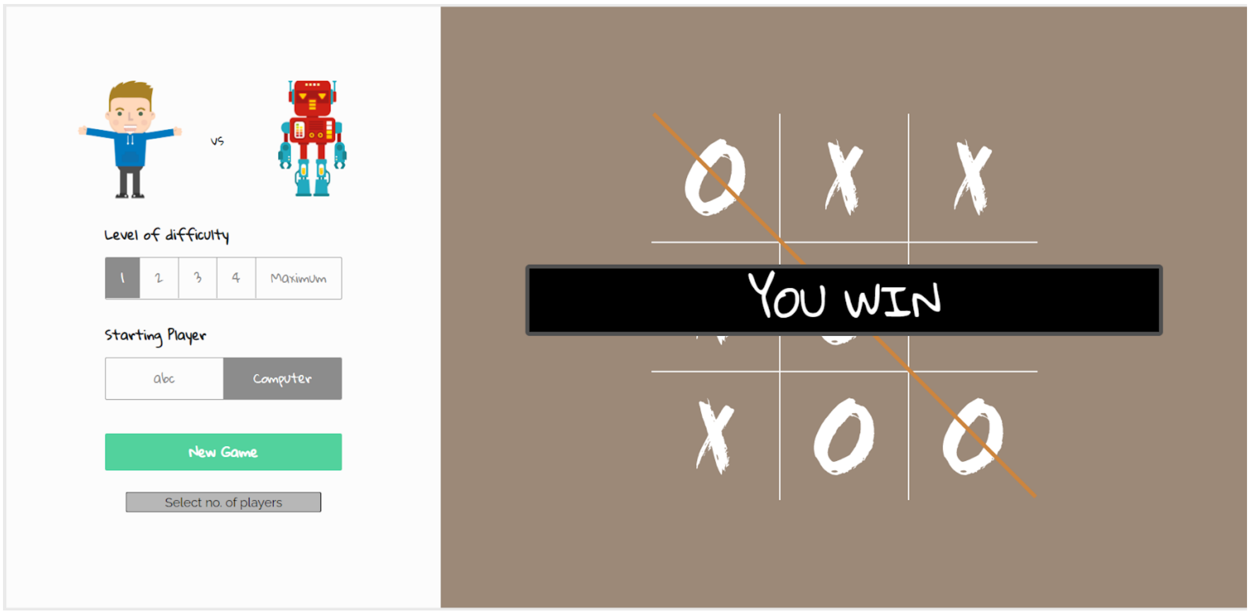
- newGame() - Function to start a new 1 player game
    - It takes the max_depth and starting_player as parameters
    - Resetting the classes for different elements
    - Initializing objects and variables required
    - If computer has to make the first move, choose a random move as long as it is a corner or the center
    - Add event listeners and accordingly call different functions and add/remove classes etc.
    - After every move update the UI accordingly
    - If the board if full or any player wins(found using evaluate() function of board class) then the game ends displaying the result over the board

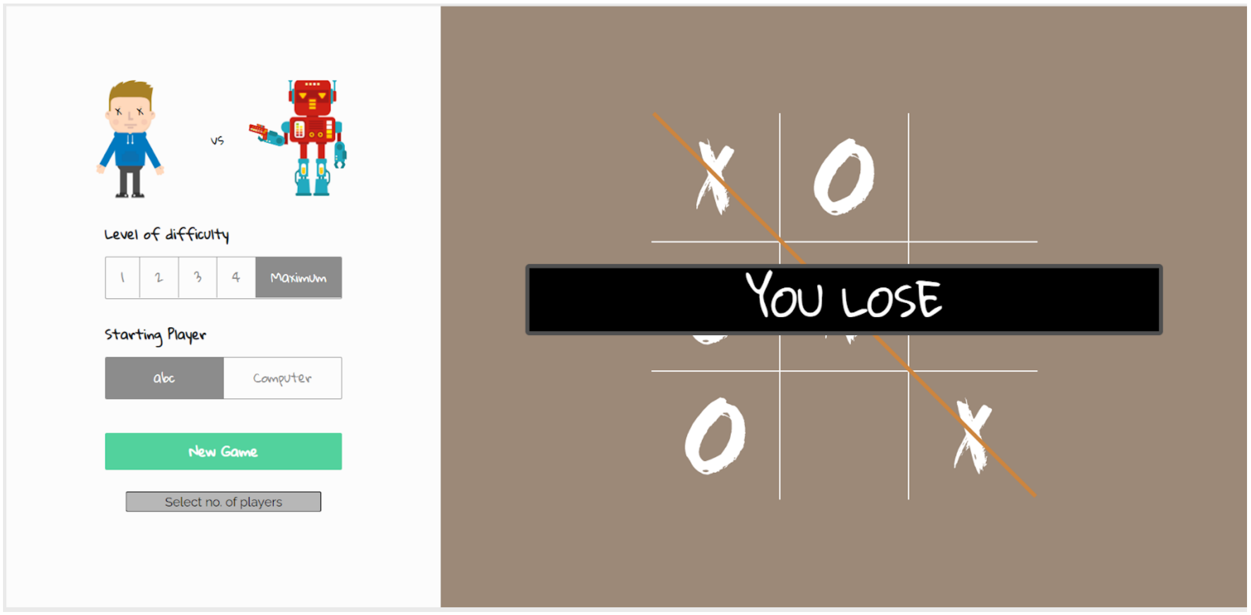
- newGame2() - Function to start a new 2 player game
    - It takes the starting_player as parameter
    - Resetting the classes for different elements
    - Initializing objects and variables required
    - Add event listeners and accordingly call different functions and add/remove classes etc.
    - player_turn is a variable used to keep track of which player's turn it is and after each player makes his move the variable is toggled to switch turns
    - After every move update the UI accordingly
    - If the board if full or any player wins(found using evaluate() function of board class) then the game ends displaying the result over the board

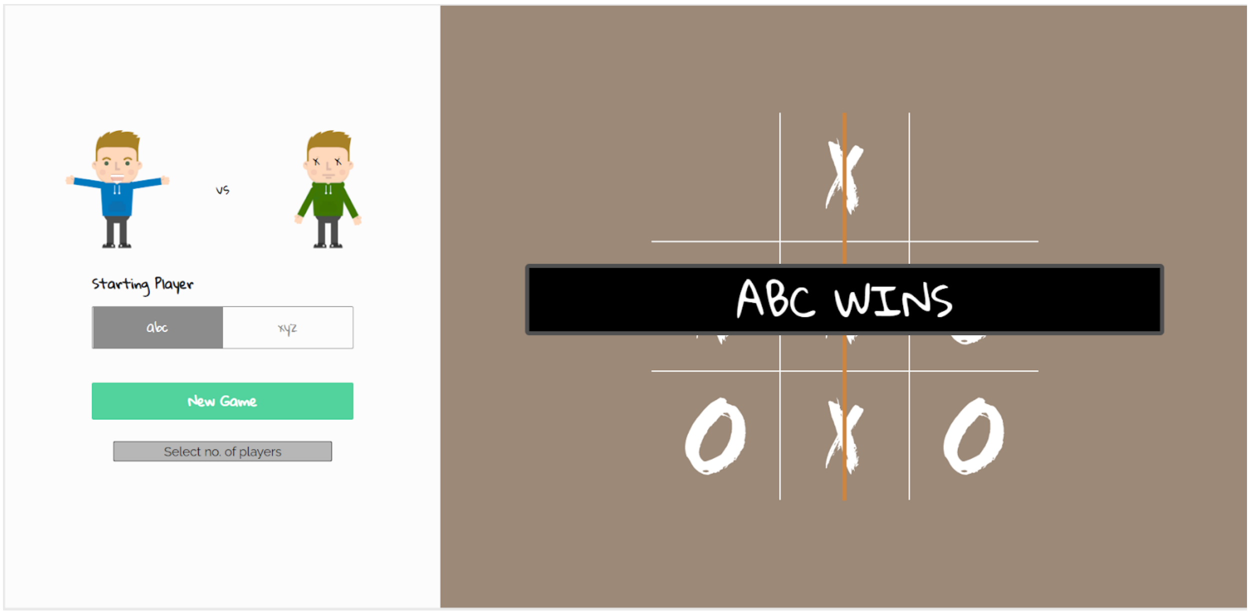
<u>Oops Aspects of the program:</u>

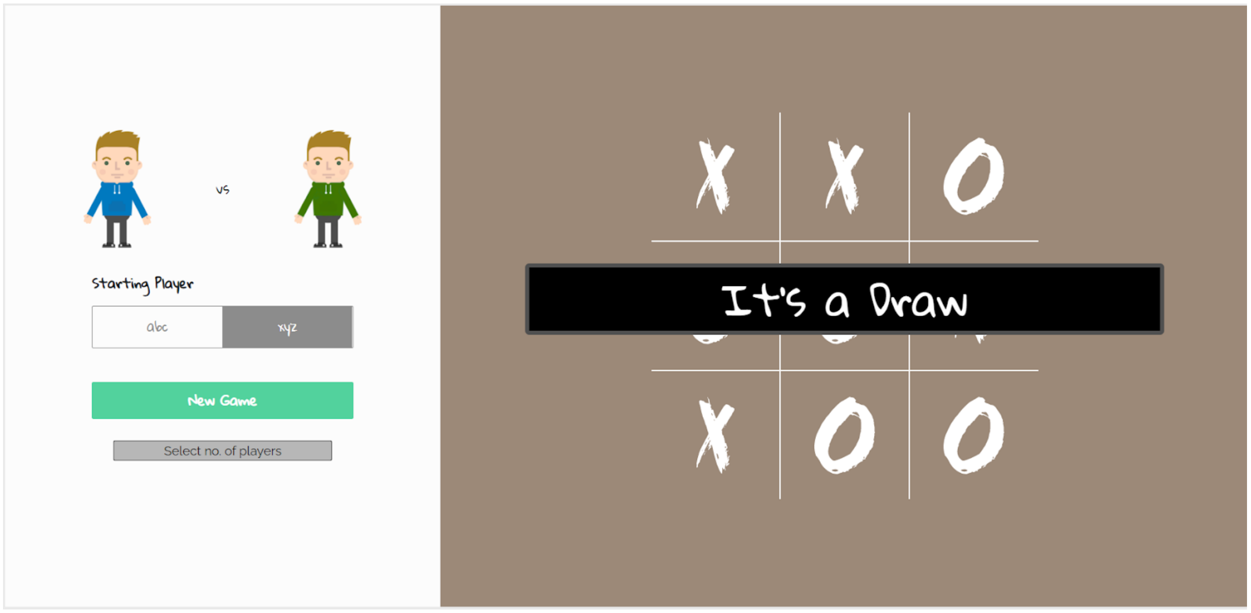
- This project is done using object oriented programming paradigm. There are 3 classes used : move, board, computer
- Each of them have separate attributes and methods to perform functionalities of the class
- The board class derives objects of class move in its methods,so there is an aggregation between the board class and the move class
- The main.js file derives objects of all these three classes to act as actuators that function in response to the input from the sensors(The UI: click,keypress etc.)

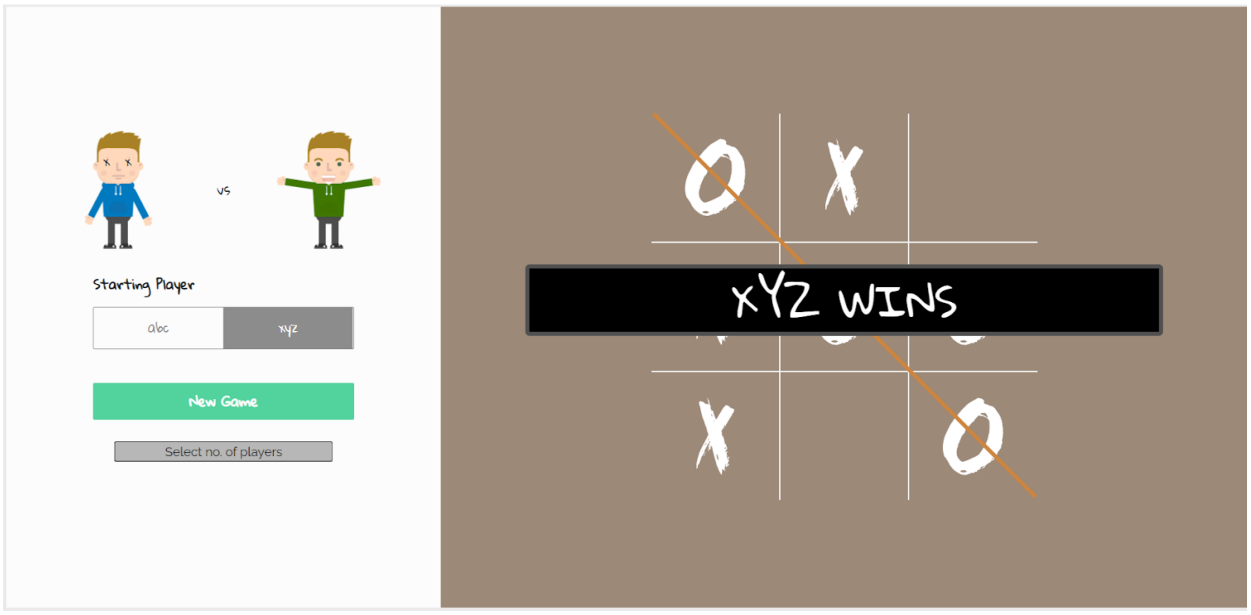
Working screenshots of the project :






vs
Starting Player
abc
xyz
New Game
Select no. of players
ABC WINS


vs
Starting Player
abc
xyz
New Game
Select no. of players
It's a Draw


vs
Starting Player
abc
xyz
New Game
Select no. of players
XYZ WINS

THANK YOU