3D TIC TAC TOE

TEAM 26 | Intelligent Systems Fall 2023

Team Members
1. Mani Kanth Dhotre
2. Ram Sirusanagandla
3. Deepthi Chinthanippu
4. Jaideep Gurrapu

PROJECT DESCRIPTION

Introduction

- The 3D Tic Tac Toe project represents an innovative move into the realm of board game design, by combining familiar board with a quantum twist.
- In this endeavor, a groundbreaking 4x4x4 quantum box is introduced that will widen the traditional playfield to new dimensions.

Gaming Framework

- A 4x4x4 game board forming a three-dimensional arena for players is included as part of the 3D Tic Tac Toe concept.
- The aim of this event is to strategically connect four pieces in a chain across various levels, calling on participants to think critically within a spatially dynamic and quantumly inspired environment.

Success Metrics

- The trick to success is to skillfully assemble four game pieces in a row, which extend beyond one plane and encompass the whole range of dimensions.
- In the standard tic tac toe gaming experience, this distinctive win condition provides a complex and deep level of detail.

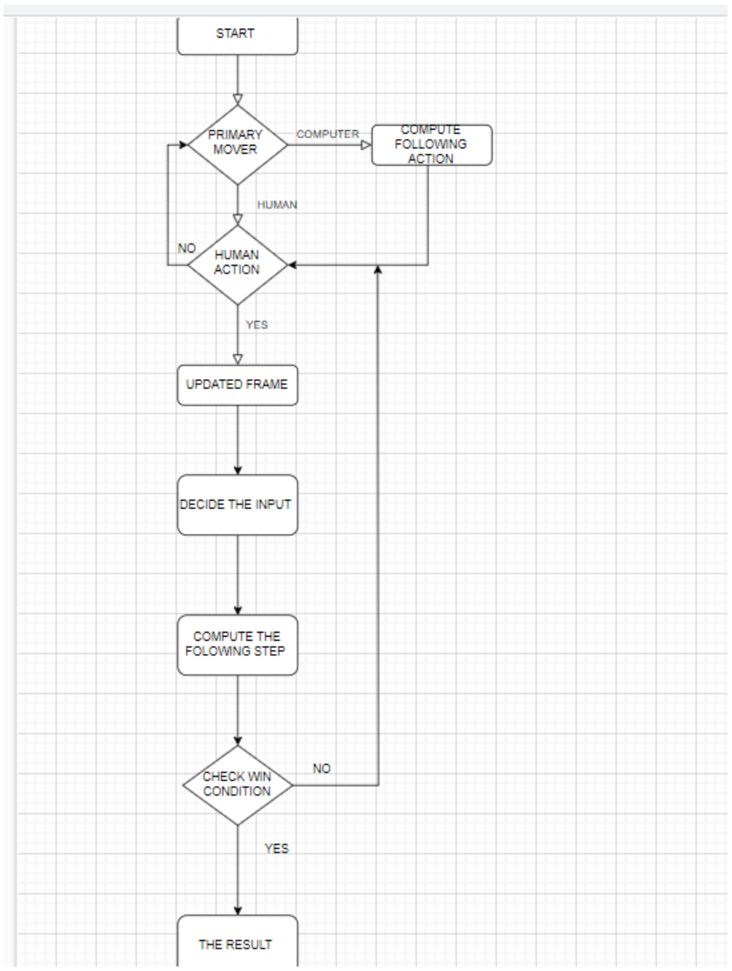
Algorithm Used

- The algorithm we employed is Minimax and alpha-beta pruning.
- Within the project, an advanced Minimax algorithm, intricately combined with alpha-beta pruning, is integrated to elevate the artificial intelligence capabilities of the opponent.
- This strategic algorithmic approach not only presents players with a dynamic and challenging gaming experience but also demonstrates the seamless integration of traditional gameplay with cutting-edge AI techniques.

Approach

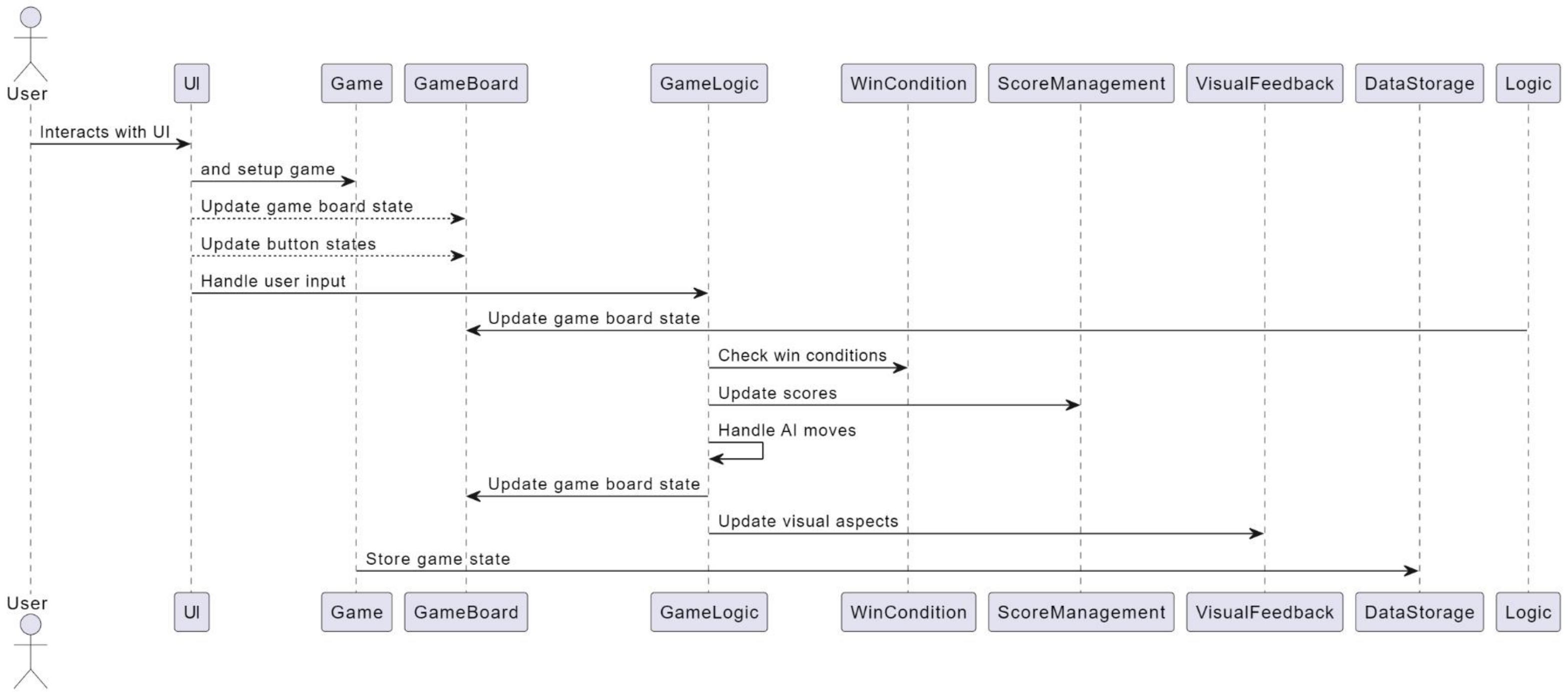
- In order to select moves based on optimal score, the Minimax algorithm takes an iterative exchange between two players. After that, each general movement shall be assessed on the basis of its lowest score in relation to all available measures.
- Each turn, the score of each opponent's move is checked by the player and cascaded from tree to tree. The introduction of alpha-beta pruning modifies the original MINIMAX algorithm, serving as an optimization technique.
- With so many games, the difficulty of MINIMAX's search algorithm is increasing on a grand scale and requires deep examination of trees. The reduction of exponents is feasible, but the total elimination cannot be achieved.
- By strategically pruning the gaming tree, rather than evaluating all nodes individually, it will be possible to calculate optimum MINIMAX decisions. Two thresholds, Alpha and Beta, are introduced by the algorithm's alpha beta Pruning mechanism which will be used to determine the trajectory of growth over time.
- Alpha-beta pruning stands as a adapted iteration of the minimax algorithm, characterized by two parameters: Alpha and Beta. Alpha represents the optimal (highest value) choice along the Maximizer's path, while Beta signifies the most favorable (lowest value) option along the Minimizer's path. Initially, the beta value is set at $+\infty$, emphasizing its openness to exploration and potential updates as the algorithm progresses.
- Alpha-beta pruning operates equivalently to its application in a standard minimax algorithm. Its particular feature is that it allows removal of nodes contributing only to the computation overhead and does not interfere with the algorithm's end result. By cutting out redundant branches, this process makes the algorithm more efficient.
- The Alpha parameter shall represent a record for the best possible move allowed by the opponent, i.e. set at -650001 which is one percentage point below the potential action representing the opponent's victory.
- Knowing that a player can beat Alpha, the minim's movement is stopped promptly when it finds an option other than Alpha.
- Conversely, Beta, initially set to 600001, monitors the most unfavorable move the opponent can force upon the player (one greater than the value of a win).
- The algorithm routes the decision tree and effectively prunes branches that do not significantly influence the outcome, by strategically weighing Beta.

SYSTEM FLOW DIAGRAM



The flowchart represents a game's decision-making process. It starts with determining the primary mover. If it's the computer, it computes its action. If it's the human's turn, the game waits for the player's action. Upon the human's move, the game frame updates, and then it's time to decide the next input. The game computes the following step, which could be either the computer's or human's move depending on the game's state. After a move, the game checks for a win condition. If no win is detected, the cycle continues. If there is a win, the game ends and presents the result.

DATA FLOW DIAGRAM



The tictactoe_game Java program features a user interface with buttons and panels for player interaction. It initializes and sets up the game environment, including UI configuration. User inputs are processed through action listeners, handling game starts, symbol choices, and difficulty settings. Game board management involves updating the board and checking for wins, using a multi-dimensional array for state storage. The AI's behavior, dictated by several methods, adapts to the game's difficulty level. Winning conditions are checked, scores are managed, and the game state is updated accordingly. The designBoard class renders visual feedback, completing the game's interactive and dynamic environment.

INPUT DESIGN

The design of the input system is a crucial element in the overall architecture of the system. The primary goal of input data design is to facilitate effortless and error-free data entry.

The input design converts user interactions into a format understandable by the computer. When a user clicks on the board, the corresponding axis coordinates are transmitted to the backend, resulting in the display of either an X or O symbol on the Board screen.

Features like the 'New Game' Button, which resets the game, and 'Easy', 'Medium', and 'Hard' radio buttons for adjusting difficulty levels, are integral parts of the design. Additionally, players can choose who makes the first move (user or CPU) through 'CPU First' or 'Human First' radio buttons.

– The method 'Board_Panelting' constructs the Graphical User Interface (GUI). 'Listener_For_BoardPanel' is an ActionListener that establishes the initial player based on player input. It clears the board, sets the starting player, updates the title text, and assesses the difficulty level upon activation. If the computer is set to play first and the difficulty level is not high, it will make a random move for a more challenging game.

– If the CPU First radio button is selected, the 'NewButtonListener' ActionListener will take the first turn, clear the board, set the display text, and initiate a new game.

– The 'Listener_For_Movements' ActionListener adjusts variables for human and computer pieces based on user inputs, then resets the game and clears the board.

– Using 'Listener_for_Levels' ActionListener, users can adjust the computer's aggressiveness or intelligence by altering the difficulty setting. This class changes the global difficulty setting and starts a new game.

– The 'actionPerformed' method is invoked for each button click on the GUI. It processes user clicks, updates the internal and GUI boards, checks if the move was successful, and declares a winner if applicable, displaying the winning move or message. If the move isn’t successful, it triggers 'random_movement_by_cpu'.

– 'Score_BoardPanel()' updates the score panel with the correct score after a win.

– The 'movement_by_cpu()' method is used when the difficulty is set to easy or medium, and the computer goes first. This is to prevent the game from being overwhelmingly difficult for human players, as a first move by the computer using the minimax strategy can be almost unbeatable. The function places the first move randomly to keep the game engaging and competitive. This function is not used on hard difficulty, allowing for more aggressive play.

– The main method in the game's AI is 'random_movement_by_cpu()'. It evaluates all available moves on the board, uses 'lookAhead()' to build branches from those moves, predicts the player's response to each possible move, and selects the most strategically advantageous one.

- 'lookAhead()' generates every possible move in the open spaces based on the current board state in response to a potential computer move in 'random_movement_by_cpu()'. The method utilizes the 'heuristic ()' function to determine the heuristic value. The function employs the alpha-beta pruning technique, especially useful when playing on hard difficulty due to the vast search tree.

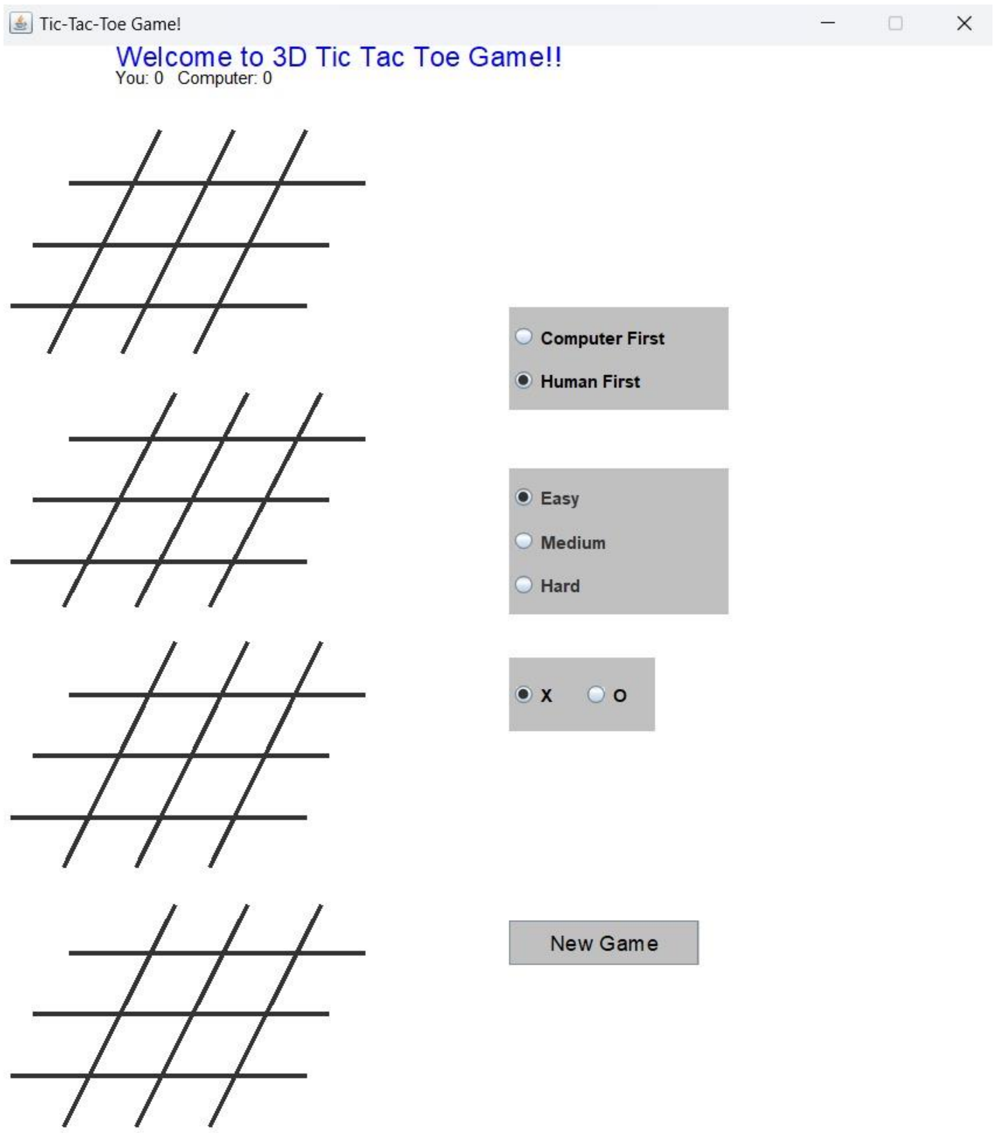
- The 'heuristic ()' function calculates its value by subtracting both the computer's and human's 'Positions_for_BoardPanel' method outcomes from the current board, with higher values being more favorable for the computer.

- 'Win_Meth()' takes a character to check for a win and a move to verify. It uses a 1-dimensional array to represent all board spaces and a 2-dimensional array to store every possible winning combination.

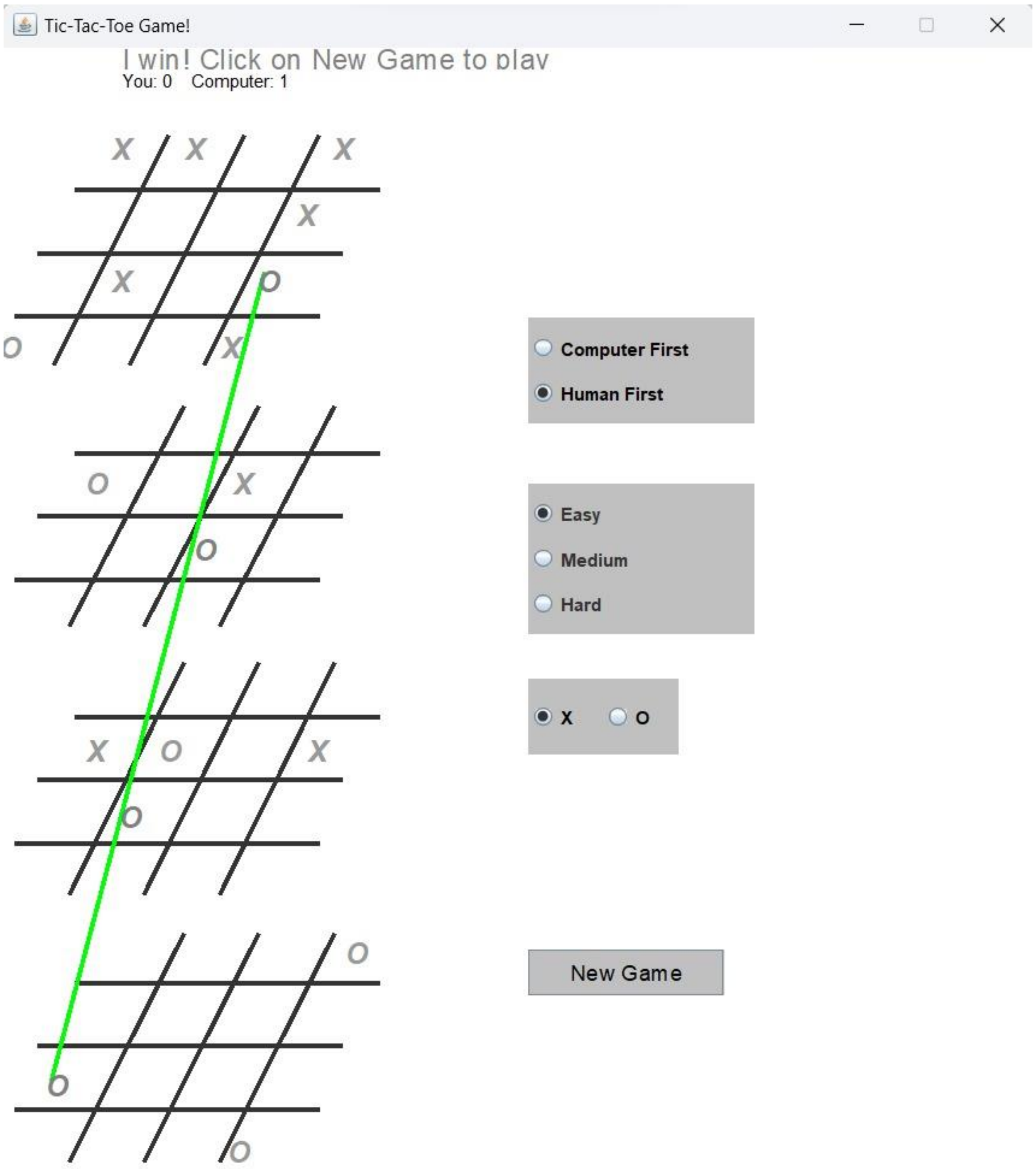
- Finally, 'Positions_for_BoardPanel' is similar to 'Win_Meth()' but returns an integer indicating the number of potential wins for the input character ('X' or 'O'), instead of a Boolean for a win.

OUTPUT DESIGN

This s the UI of our application



This is an instance of computer win case


Tic-Tac-Toe Game!

I win! Click on New Game to play

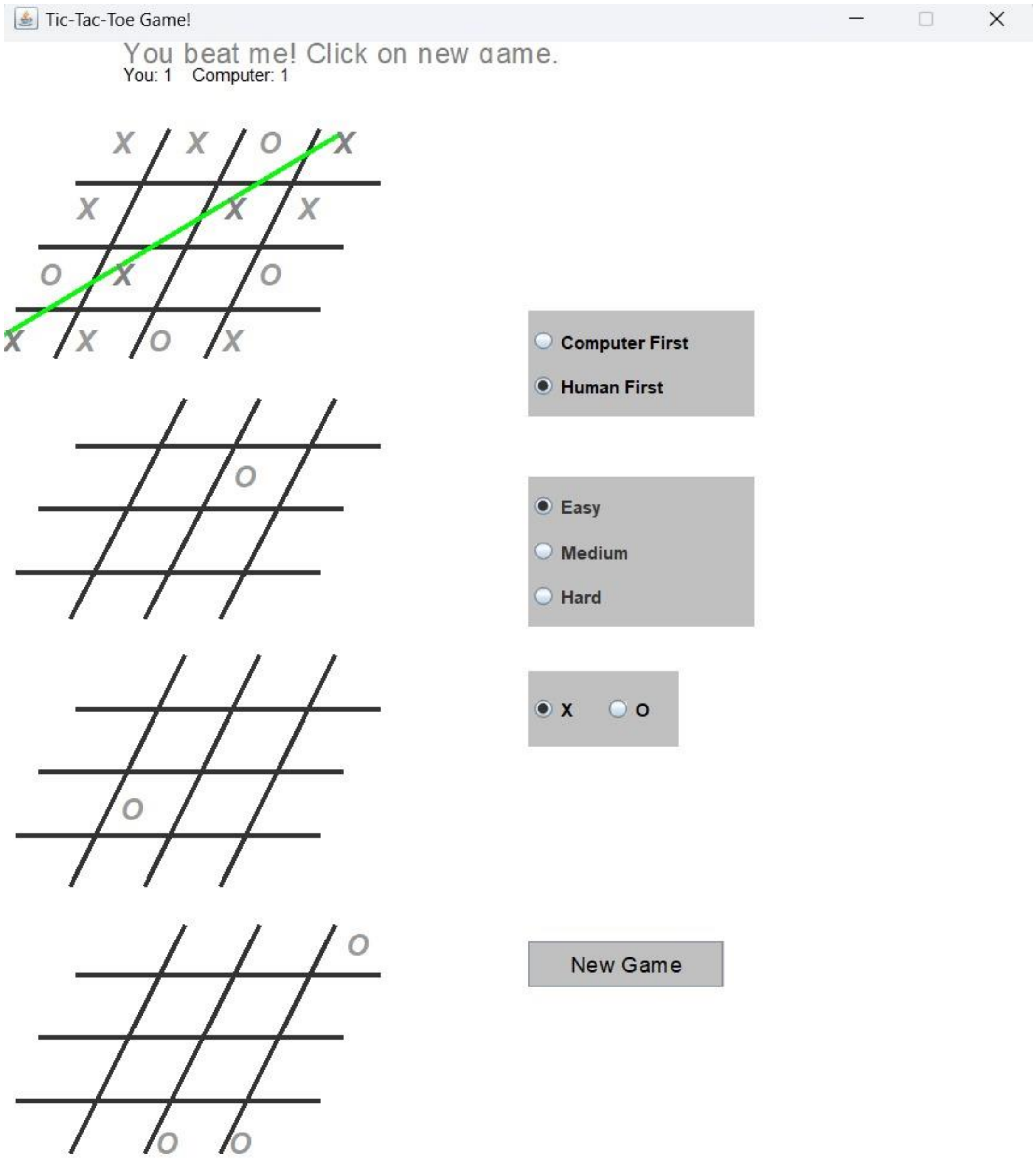
You: 0 Computer: 1

Computer First [✓ Human First]

[✓ Easy] Medium Hard

[✓ X] O

This is an instance of human win case



Tic-Tac-Toe Game!

You beat me! Click on new game.

You: 1   Computer: 1

( ) Computer First

(●) Human First

(●) Easy

( ) Medium

( ) Hard

(●) X   ( ) O