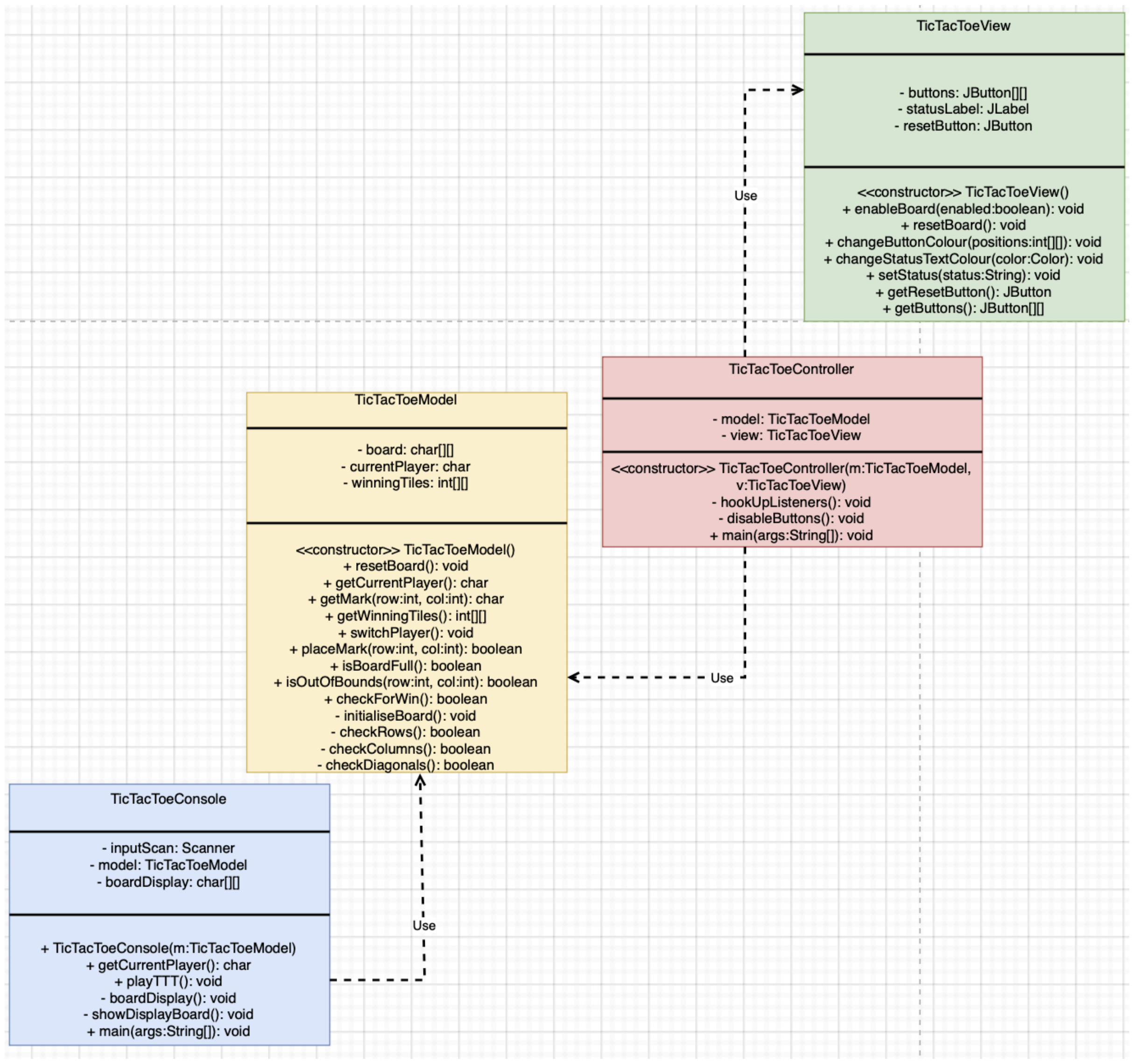
OOP TicTacToe Assignment Report

This report documents the design, implementation, and testing of a Java-based Tic-Tac-Toe (Noughts & Crosses) application, which I've developed for my first-year, second-semester assessment in Object-Oriented Programming (OOP). The project follows the Model-View-Controller (MVC) design to separate the game logic, user interface and control flow. Both a console and GUI version of the game were created, using the same model for functionality, and the core game logic (win/draw conditions) was tested to ensure the correct functionality.

Class and Method Descriptions + UML Diagram

Image below is a UML diagram for the application.


TicTacToeView
- buttons: JButton[][]
- statusLabel: JLabel
- resetButton: JButton
<<constructor>> TicTacToeView()
+ enableBoard(enabled:boolean): void
+ resetBoard(): void
+ changeButtonColour(positions:int[][]): void
+ changeStatusTextColour(color:Color): void
+ setStatus(status:String): void
+ getResetButton(): JButton
+ getButtons(): JButton[][]
Use
TicTacToeController
- model: TicTacToeModel
- view: TicTacToeView
<<constructor>> TicTacToeController(m:TicTacToeModel, v:TicTacToeView)
- hookUpListeners(): void
- disableButtons(): void
+ main(args:String[]): void
TicTacToeModel
- board: char[][]
- currentPlayer: char
- winningTiles: int[][]
<<constructor>> TicTacToeModel()
+ resetBoard(): void
+ getCurrentPlayer(): char
+ getMark(row:int, col:int): char
+ getWinningTiles(): int[][]
+ switchPlayer(): void
+ placeMark(row:int, col:int): boolean
+ isBoardFull(): boolean
+ isOutOfBounds(row:int, col:int): boolean
+ checkForWin(): boolean
- initialiseBoard(): void
- checkRows(): boolean
- checkColumns(): boolean
- checkDiagonals(): boolean
Use
TicTacToeConsole
- inputScan: Scanner
- model: TicTacToeModel
- boardDisplay: char[][]
+ TicTacToeConsole(m:TicTacToeModel)
+ getCurrentPlayer(): char
+ playTTT(): void
- boardDisplay(): void
- showDisplayBoard(): void
+ main(args:String[]): void
Use

TicTacToeModel

- *Fields:*
  - `char[][] board`: The 3x3 game board, storing 'X', 'O', or ' ' for each tile.
  - `char currPlayer`: Stores the current player ('X' or 'O')
  - `int[][] winningTiles`: Stores the positions of the winning line of tiles.
- *Key Methods:*
  - `TicTacToeModel()`: Constructor. Initializes the board and sets the current player to 'X'.
  - `initialiseBoard()`: Sets all board cells to empty and resets the current player (back to 'X').
  - `resetBoard()`: Calls `initialiseBoard()` to reset the game.
  - `getCurrentPlayer()`: Returns the current player.
  - `getMark(int row, int column)`: Returns the mark at the specified position, or a space if the position is found to be out of bounds.
  - `getWinningTiles()`: Returns the positions of the winning line, if there is any.
  - `switchPlayer()`: Switches the current player from 'X' to 'O' or vice versa.
  - `placeMark(int row, int column)`: Places the current player's mark if the cell is empty and not out of bounds.
  - `isBoardFull()`: Checks if the board is full (no empty spaces).
  - `isOutOfBounds(int row, int column)`: Checks if the given position is outside the board.
  - `checkForWin()`: Returns true if any row, column, or diagonal contains three identical marks (using private methods below).
  - `checkRows()`, `checkColumns()`, `checkDiagonals()`: Private methods to check for wins in rows, columns, and diagonals, respectively, and update winningTiles if a win is found.

TicTacToeConsole

- *Fields:*
  - `Scanner inputScan`: For reading user input.
  - `TicTacToeModel modelBoard`: The game model.
  - `char[][] boardDisplay`: Stores the current display state of the board.
- *Key Methods:*
  - `TicTacToeConsole(TicTacToeModel m)`: Constructor. Sets up the model and input scanner.
  - `getCurrentPlayer()`: Returns the current player from the model.
  - `playTTT()`: Main game loop for console play. Asks users for moves, updates the board, and checks for a win or a draw (board is full).
  - `boardDisplay()`: Updates the display board using the model.
  - `showDisplayBoard()`: Prints the current board to the console.
  - `main(String[] args)`: Entry point for running the console version.

TicTacToeView

- *Fields:*
  - `JButton[][] buttons`: The 3x3 grid of buttons for the board.
  - `JLabel statusLabel`: Displays the current status or turn.
  - `JButton resetButton`: Button to reset the game.
- *Key Methods:*
  - `TicTacToeView()`: Constructor. Sets up the GUI layout, buttons, and status label.
  - `enableBoard(boolean enabled)`: Enables or disables all board buttons.
  - `resetBoard()`: Clears the board and resets button colors and status label to original colours.
  - `changeButtonColour(int[][] positions)`: Changes colour of specified buttons.
  - `changeStatusTextColour(Color color)`: Changes the status label text color.
  - `getStatus(String status)`: Sets the status label text.
  - `getResetButton()`: Returns the reset button.
  - `getButtons()`: Returns the board buttons.

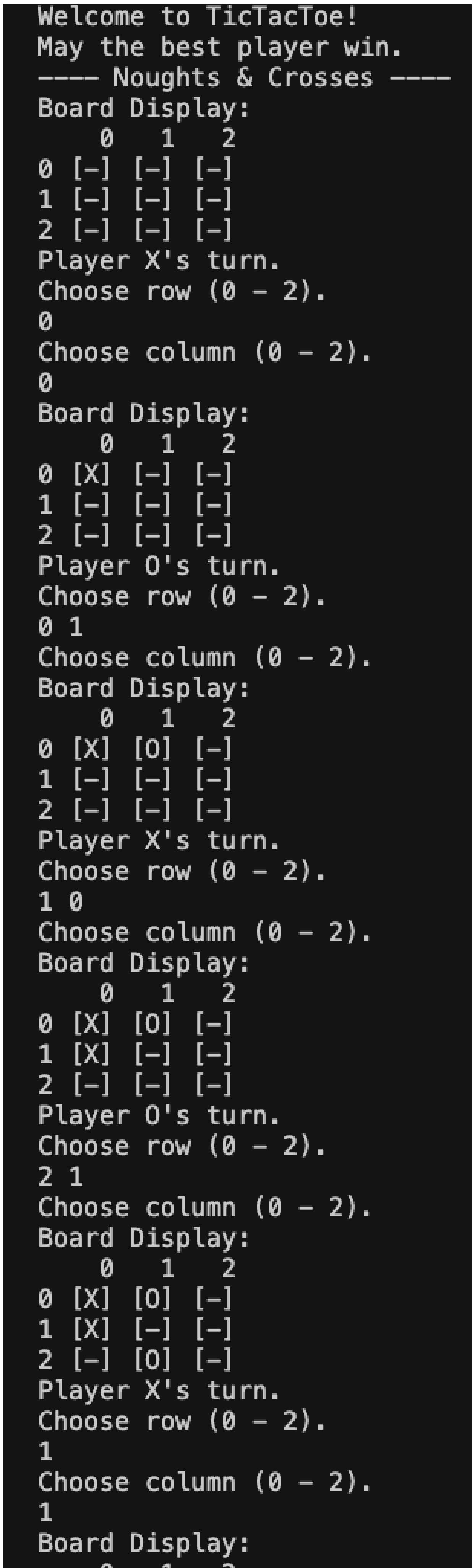
TicTacToeController

- *Fields:*
  - `TicTacToeModel model`: The game model.
  - `TicTacToeView view`: The GUI view.
- *Key Methods:*
  - `TicTacToeController(TicTacToeModel m, TicTacToeView v)`: Constructor. Sets up listeners and initializes the game.
  - `hookUpListeners()`: Adds action listeners to all buttons and the reset button; handles moves, detecting if a win or draw is reached, and updating the GUI (changing winning tiles and text to green, adding X / O buttons, etc).
  - `disableButtons()`: Disables all board buttons.
  - `main(String[] args)`: Entry point for running the GUI version.

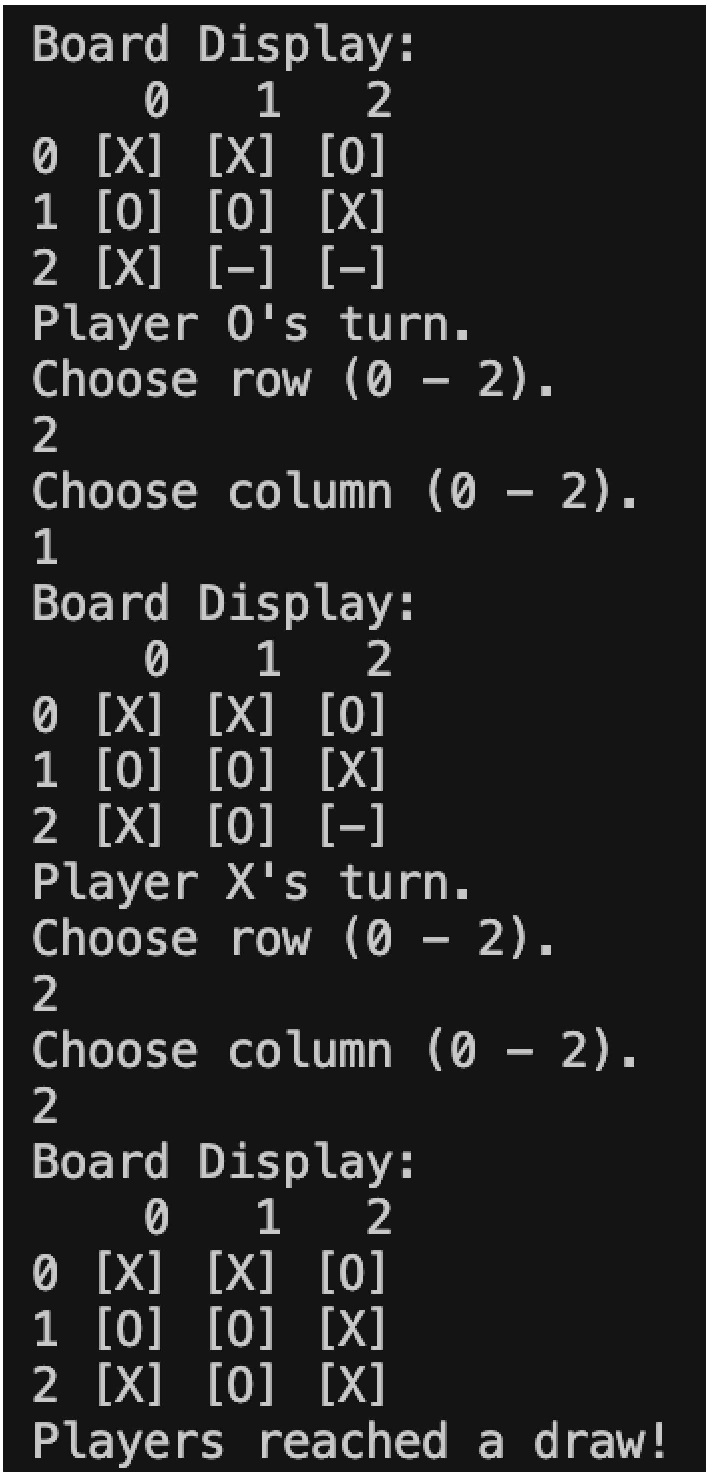
Console Program Game Play

The TicTacToeConsole class provides a text-based interface allowing users to play TicTacToe in the terminal. When launched, it displays a welcome message and an empty 3x3 board, alongside rows and column numbers to guide user input. Users take turns entering their moves, which are validated to ensure they are within bounds and not already occupied. After a valid move, the board updates and the program checks for a win or draw condition, swapping players if none are found. The game ends with a message if a player wins or if the board is full. The console component interacts with the model to manage game state and uses the same model as the GUI component.

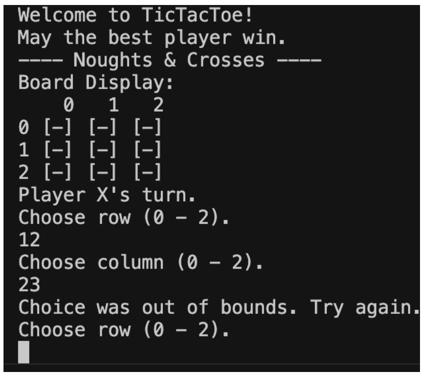
Images: The first is a full game where Player X wins, and the second is the last three Board Displays for a game ending in a draw.



```
Welcome to TicTacToe!
May the best player win.
---- Noughts & Crosses ----
Board Display:
    0   1   2
0 [-] [-] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player X's turn.
Choose row (0 - 2).
0
Choose column (0 - 2).
0
Board Display:
    0   1   2
0 [X] [-] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player O's turn.
Choose row (0 - 2).
0 1
Choose column (0 - 2).
Board Display:
    0   1   2
0 [X] [O] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player X's turn.
Choose row (0 - 2).
1 0
Choose column (0 - 2).
Board Display:
    0   1   2
0 [X] [O] [-]
1 [X] [-] [-]
2 [-] [-] [-]
Player O's turn.
Choose row (0 - 2).
2 1
Choose column (0 - 2).
Board Display:
    0   1   2
0 [X] [O] [-]
1 [X] [-] [-]
2 [-] [O] [-]
Player X's turn.
Choose row (0 - 2).
1
Choose column (0 - 2).
1
Board Display:
```

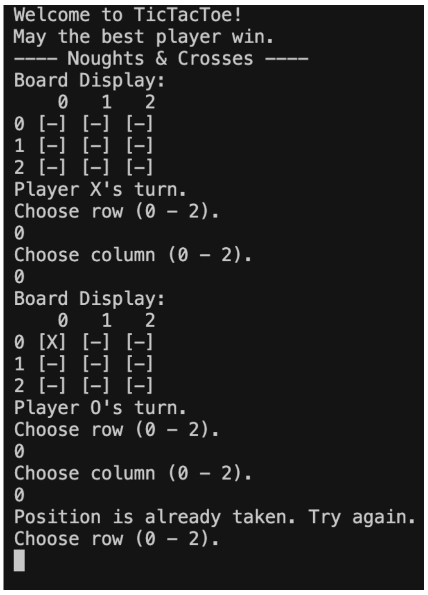


```
Board Display:
    0   1   2
0 [X] [X] [O]
1 [O] [O] [X]
2 [X] [-] [-]
Player O's turn.
Choose row (0 - 2).
2
Choose column (0 - 2).
1
Board Display:
    0   1   2
0 [X] [X] [O]
1 [O] [O] [X]
2 [X] [O] [-]
Player X's turn.
Choose row (0 - 2).
2
Choose column (0 - 2).
2
Board Display:
    0   1   2
0 [X] [X] [O]
1 [O] [O] [X]
2 [X] [O] [X]
Players reached a draw!
```



```
Welcome to TicTacToe!
May the best player win.
---- Noughts & Crosses ----
Board Display:
    0   1   2
0 [-] [-] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player X's turn.
Choose row (0 - 2).
12
Choose column (0 - 2).
23
Choice was out of bounds. Try again.
Choose row (0 - 2).
```

Image shows what happens if you attempt to place out of bounds

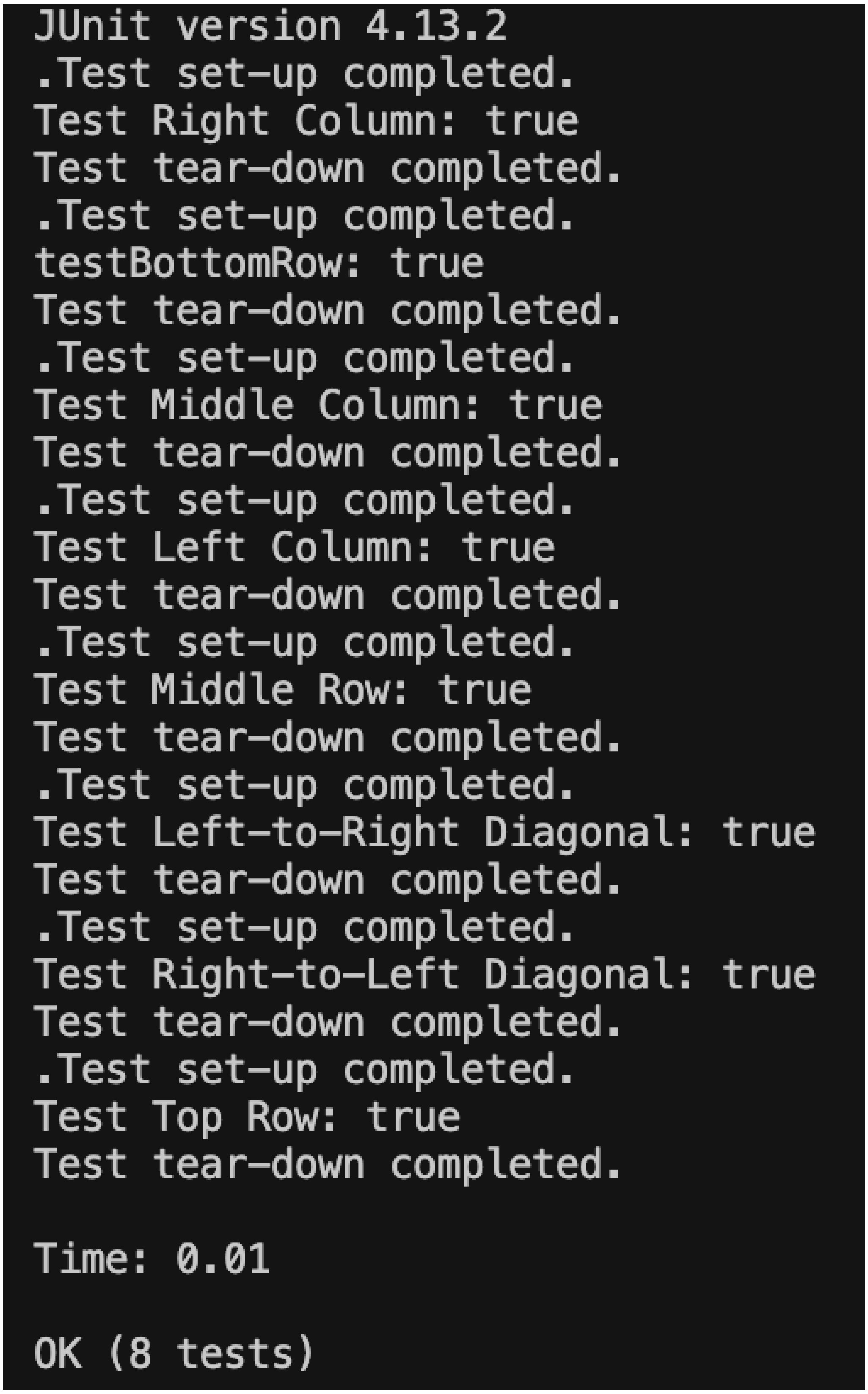


```
Welcome to TicTacToe!
May the best player win.
---- Noughts & Crosses ----
Board Display:
    0   1   2
0 [-] [-] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player X's turn.
Choose row (0 - 2).
0
Choose column (0 - 2).
0
Board Display:
    0   1   2
0 [X] [-] [-]
1 [-] [-] [-]
2 [-] [-] [-]
Player O's turn.
Choose row (0 - 2).
0
Choose column (0 - 2).
0
Position is already taken. Try again.
Choose row (0 - 2).
```

Image shows what happens if you attempt placing on an already filled tile.

JUnit Model Testing

The JUnit testing component demonstrates that the core game logic works correctly. It detects all possible win conditions with a series of eight automated tests checking for every possible line of three same tiles. The screenshot below shows the results of running these tests, confirming checkForWin() condition behaves as expected.

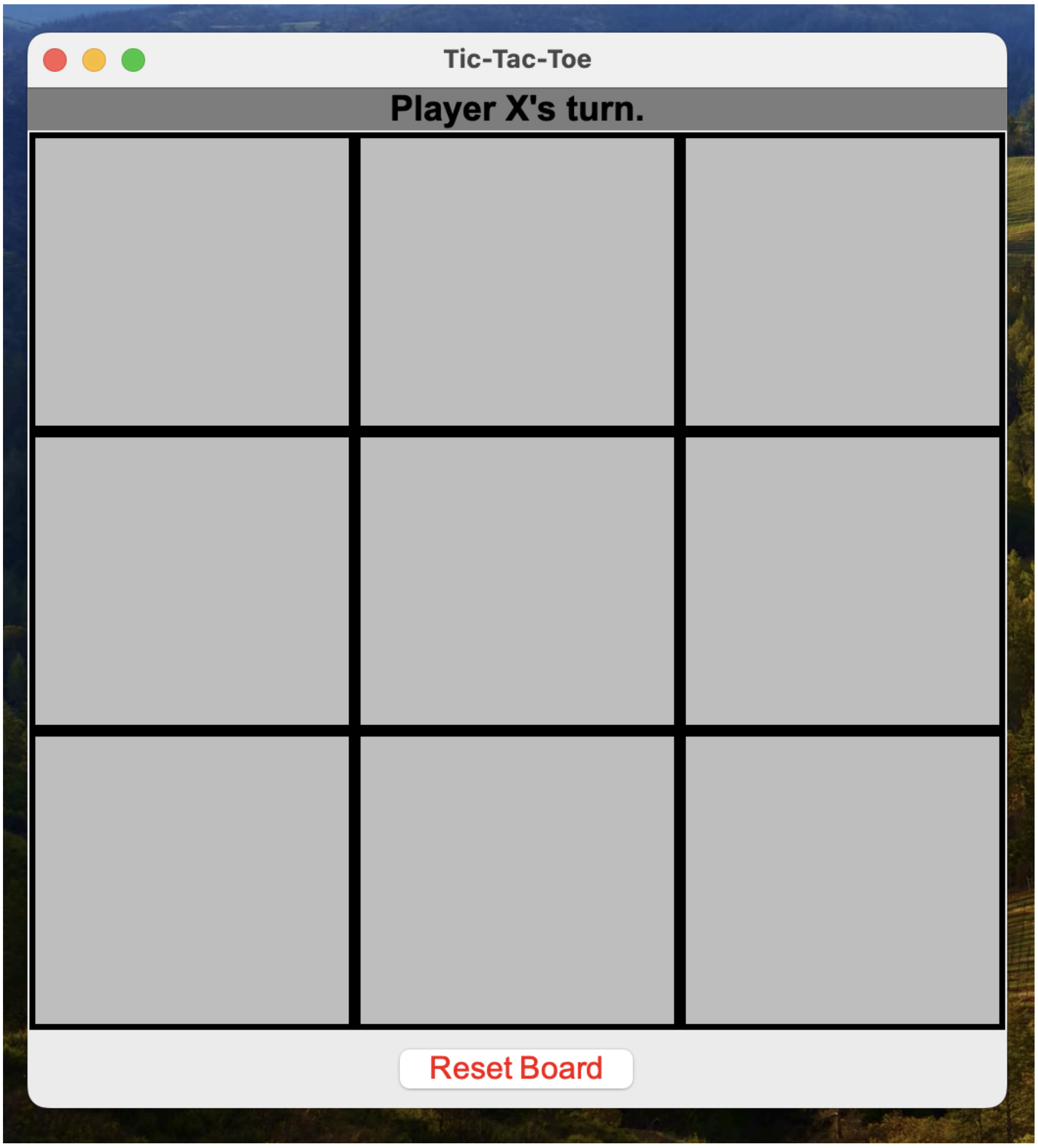

```
JUnit version 4.13.2
.Test set-up completed.
Test Right Column: true
Test tear-down completed.
.Test set-up completed.
testBottomRow: true
Test tear-down completed.
.Test set-up completed.
Test Middle Column: true
Test tear-down completed.
.Test set-up completed.
Test Left Column: true
Test tear-down completed.
.Test set-up completed.
Test Middle Row: true
Test tear-down completed.
.Test set-up completed.
Test Left-to-Right Diagonal: true
Test tear-down completed.
.Test set-up completed.
Test Right-to-Left Diagonal: true
Test tear-down completed.
.Test set-up completed.
Test Top Row: true
Test tear-down completed.

Time: 0.01

OK (8 tests)
```

GUI at the Start of Gameplay

The screenshot below shows the start of a new game in the GUI, prior to any inputs. The window displays an empty 3x3 grid of buttons for the board, a status label indicating it is Player X's turn, and a reset button at the bottom. A player can click on any empty square to make their move, and the interface (buttons and status label) will update accordingly.

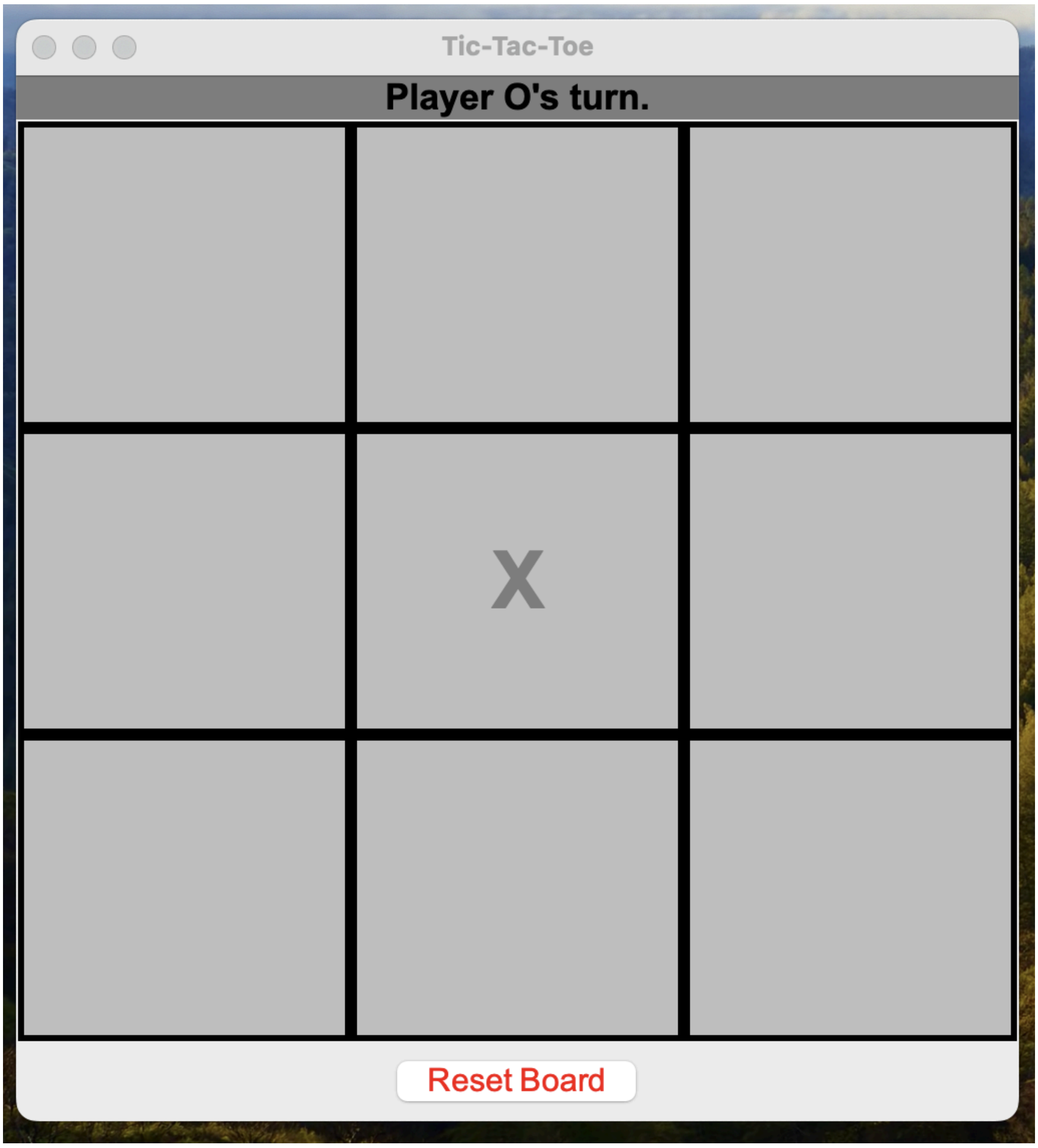


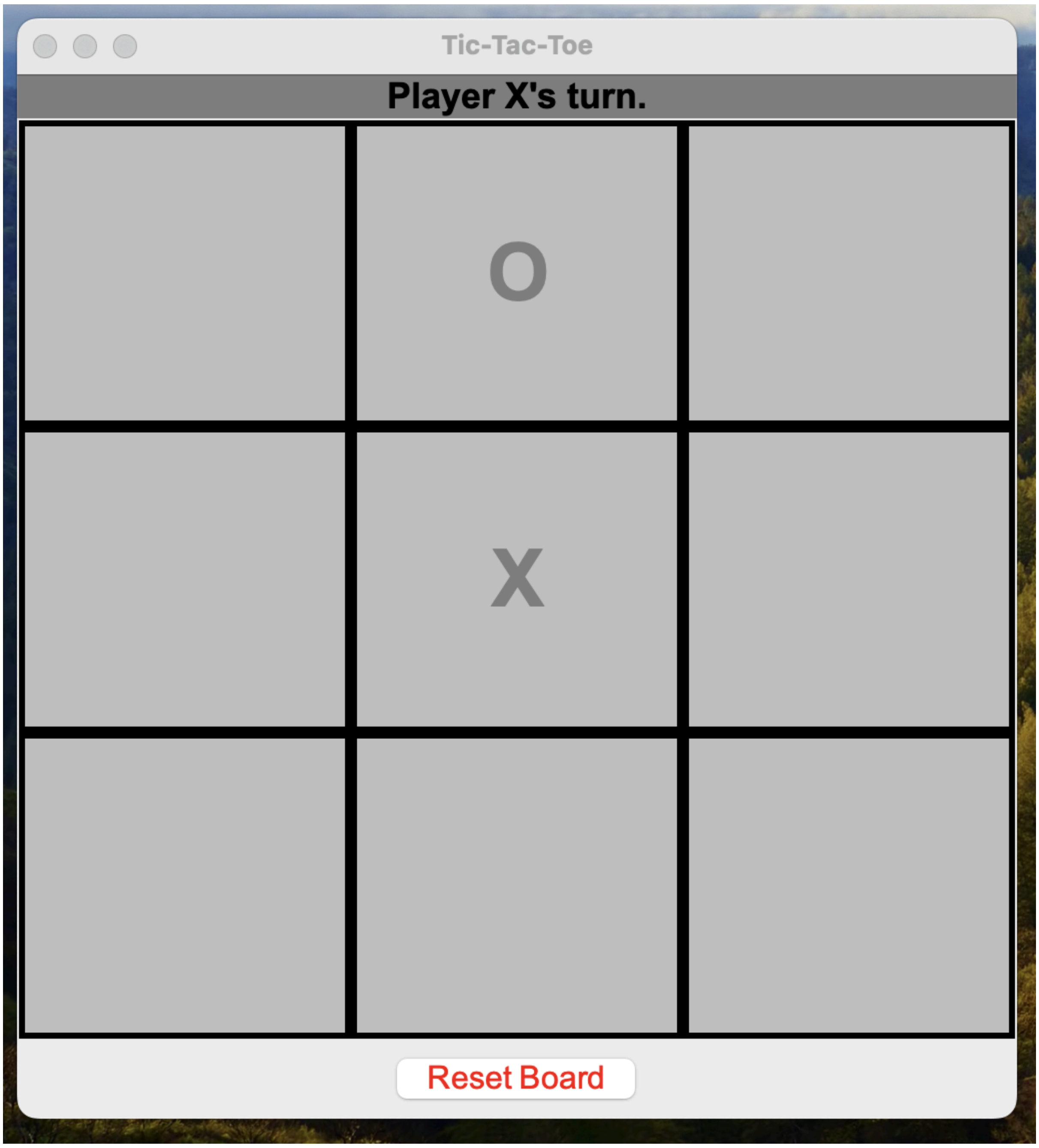
GUI-Based Gameplay

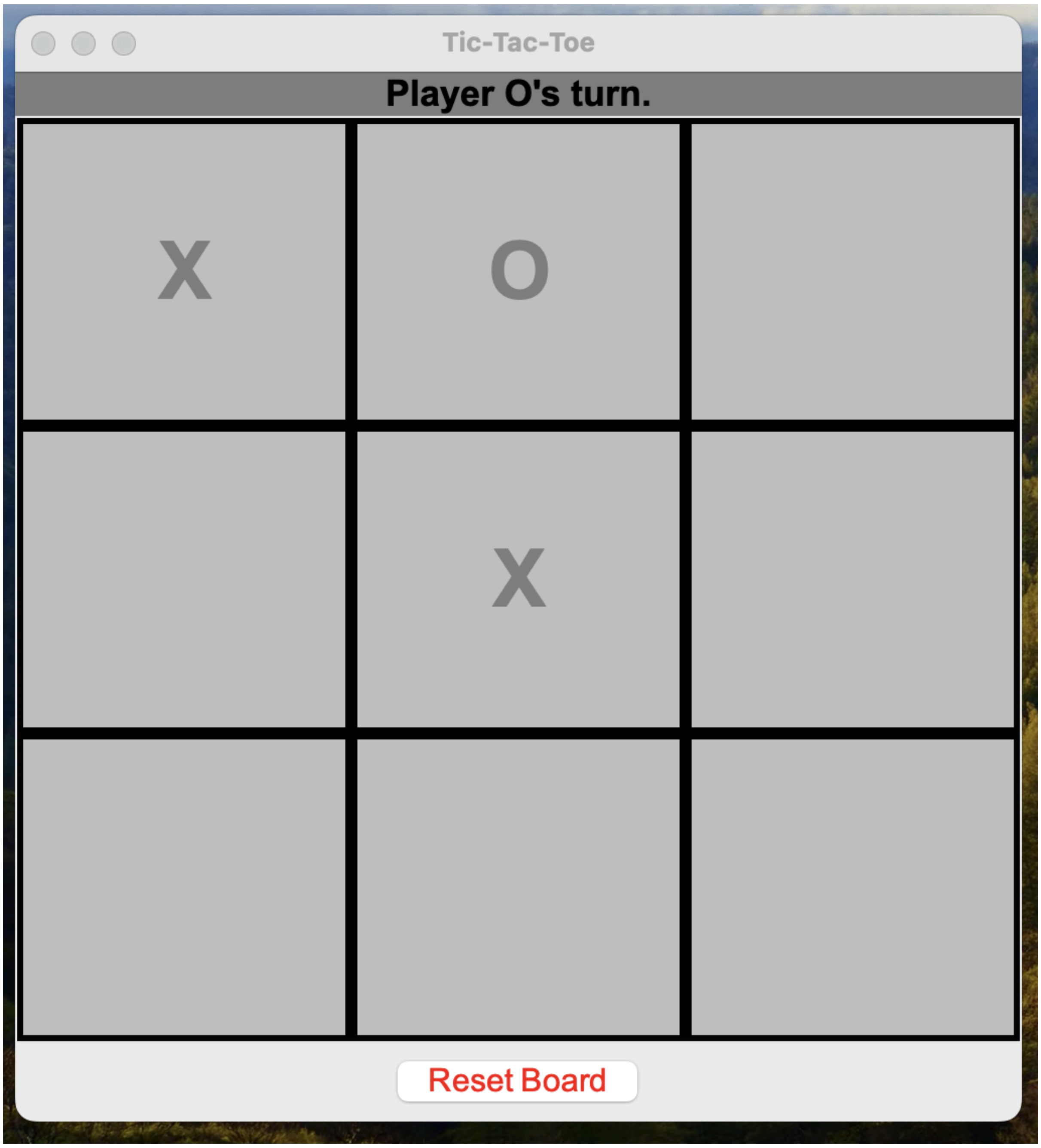
The following sequence of screenshots below show the GUI-based gameplay as a simple game progresses. Each screenshot shows the state of the GUI after a player's move (the updated board, current player's turn displayed in the status label, and any win or draw messages). This demonstrates how the GUI responds to user inputs throughout the game.

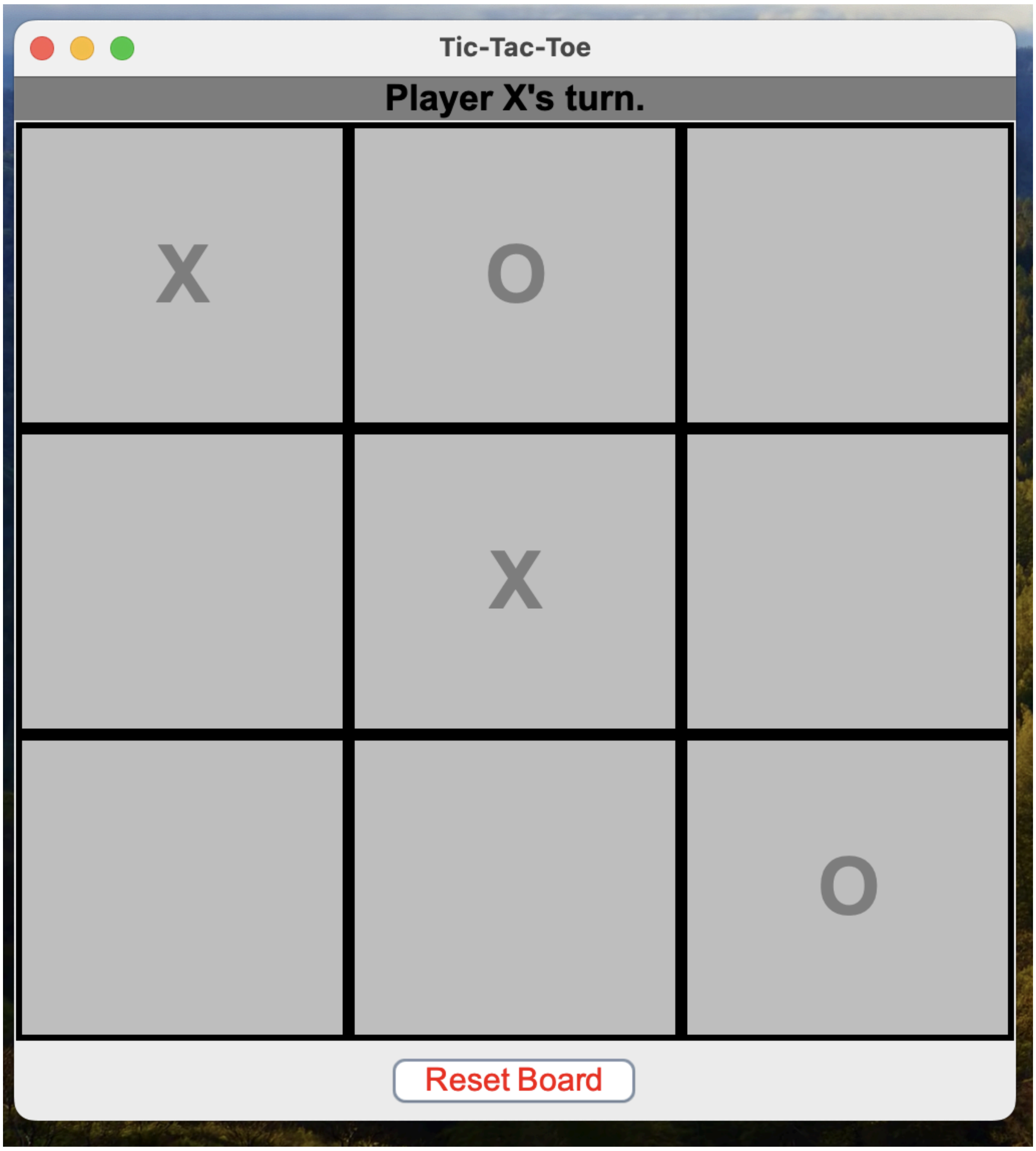
Resetting the board

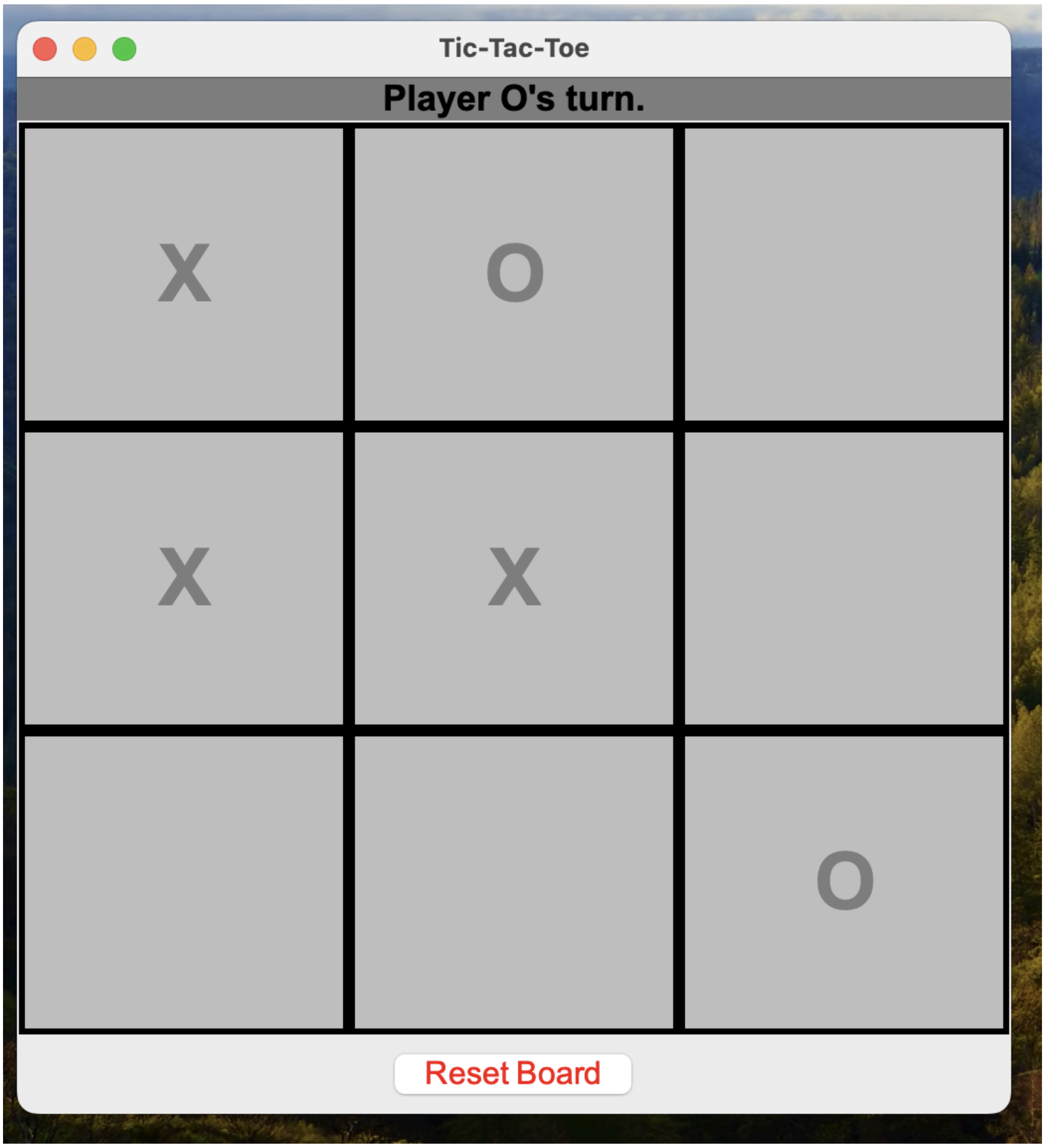
Game resulting in player X win:

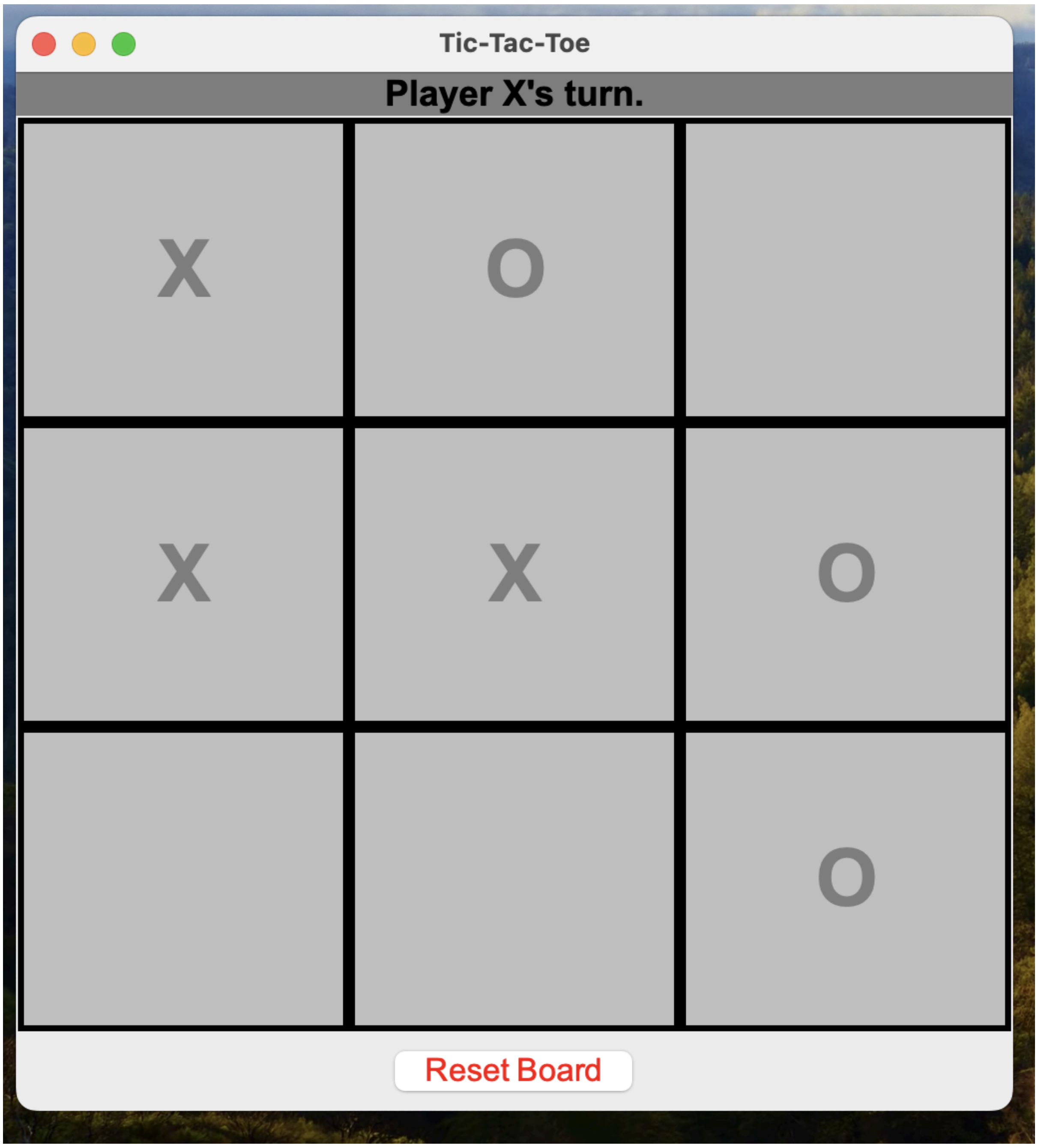


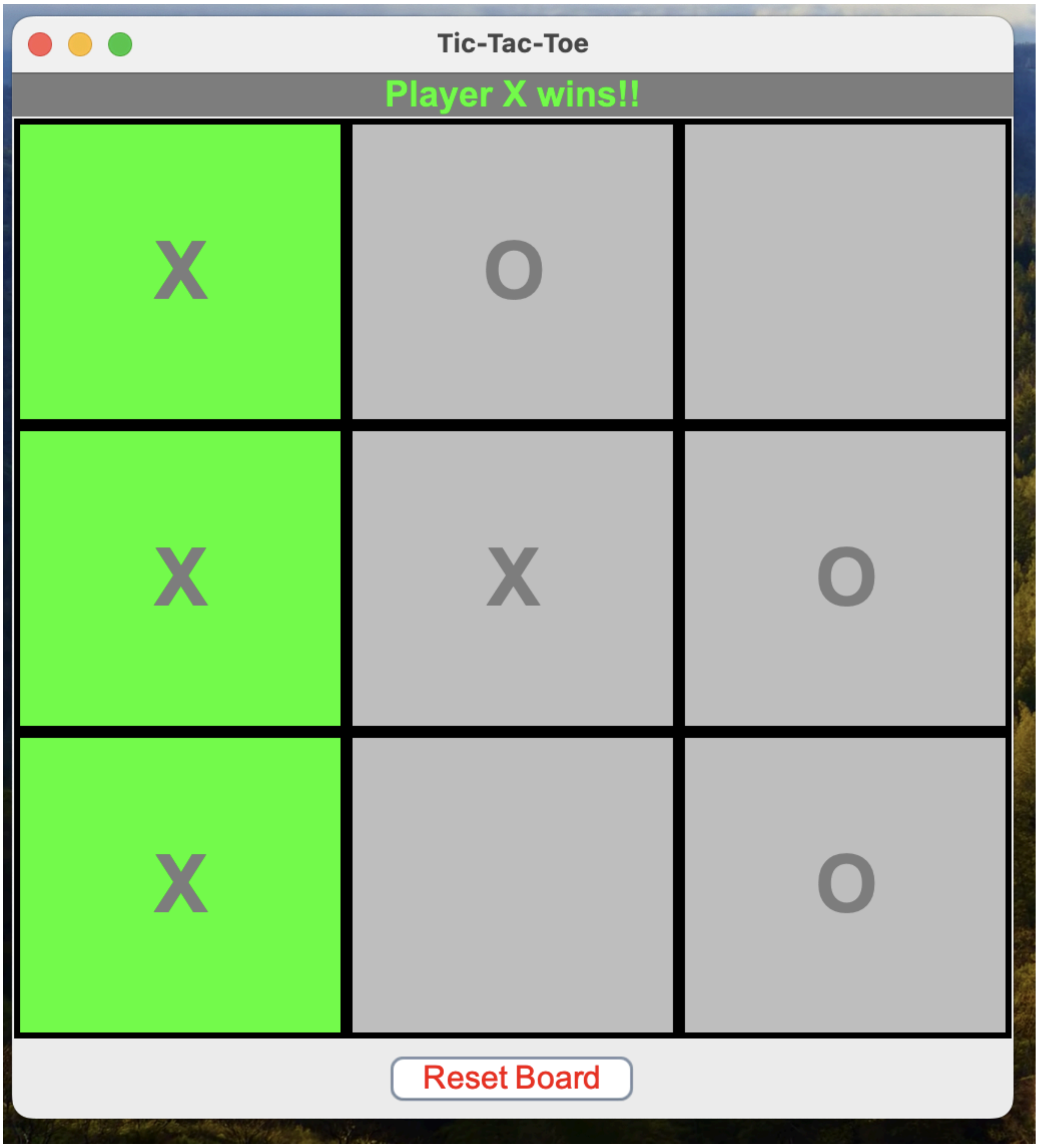

Tic-Tac-Toe
Player X's turn.
O
X
Reset Board


Tic-Tac-Toe
Player O's turn.
X
O
X
Reset Board

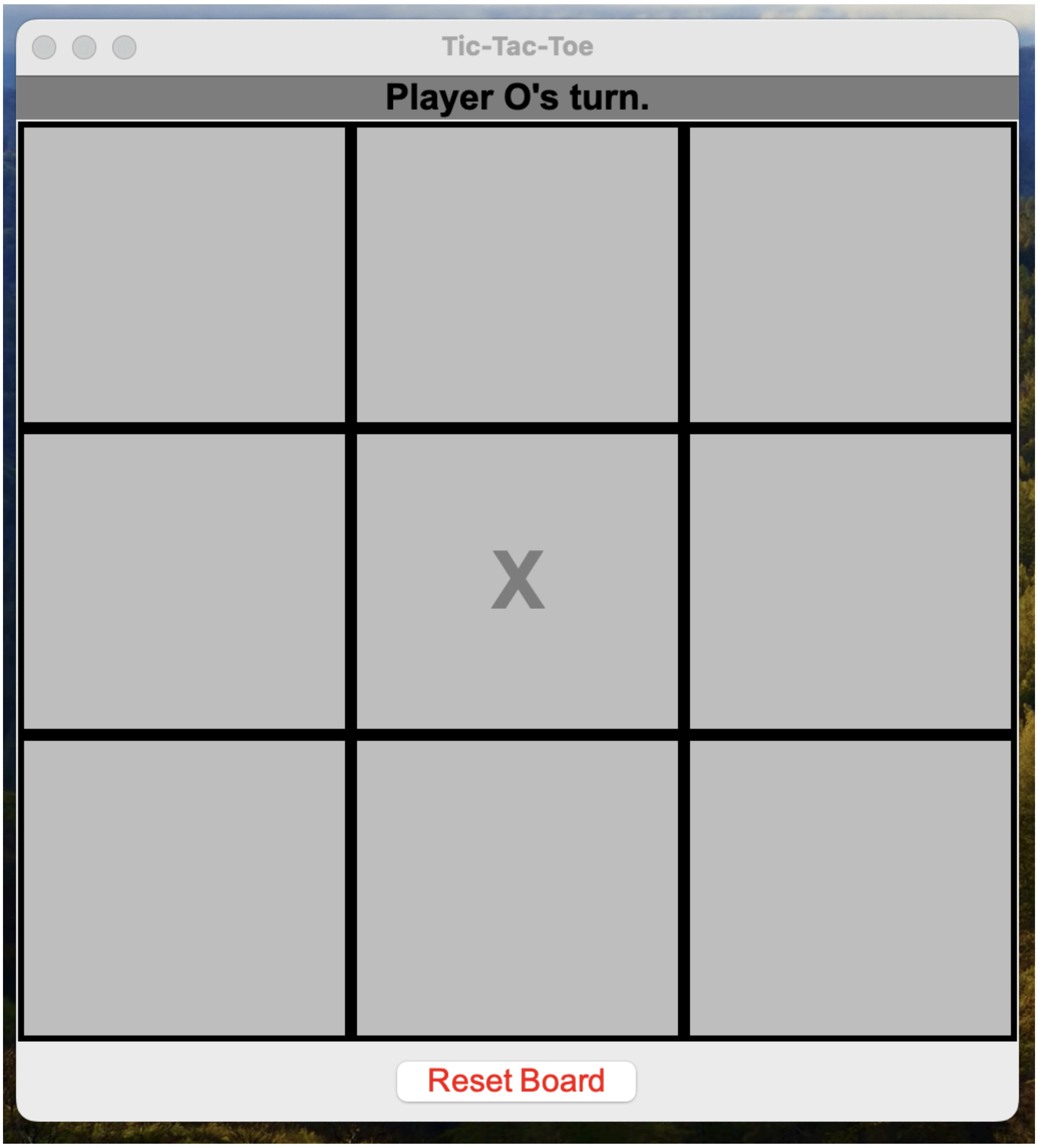

Tic-Tac-Toe
Player X's turn.
X
O
X
O
Reset Board

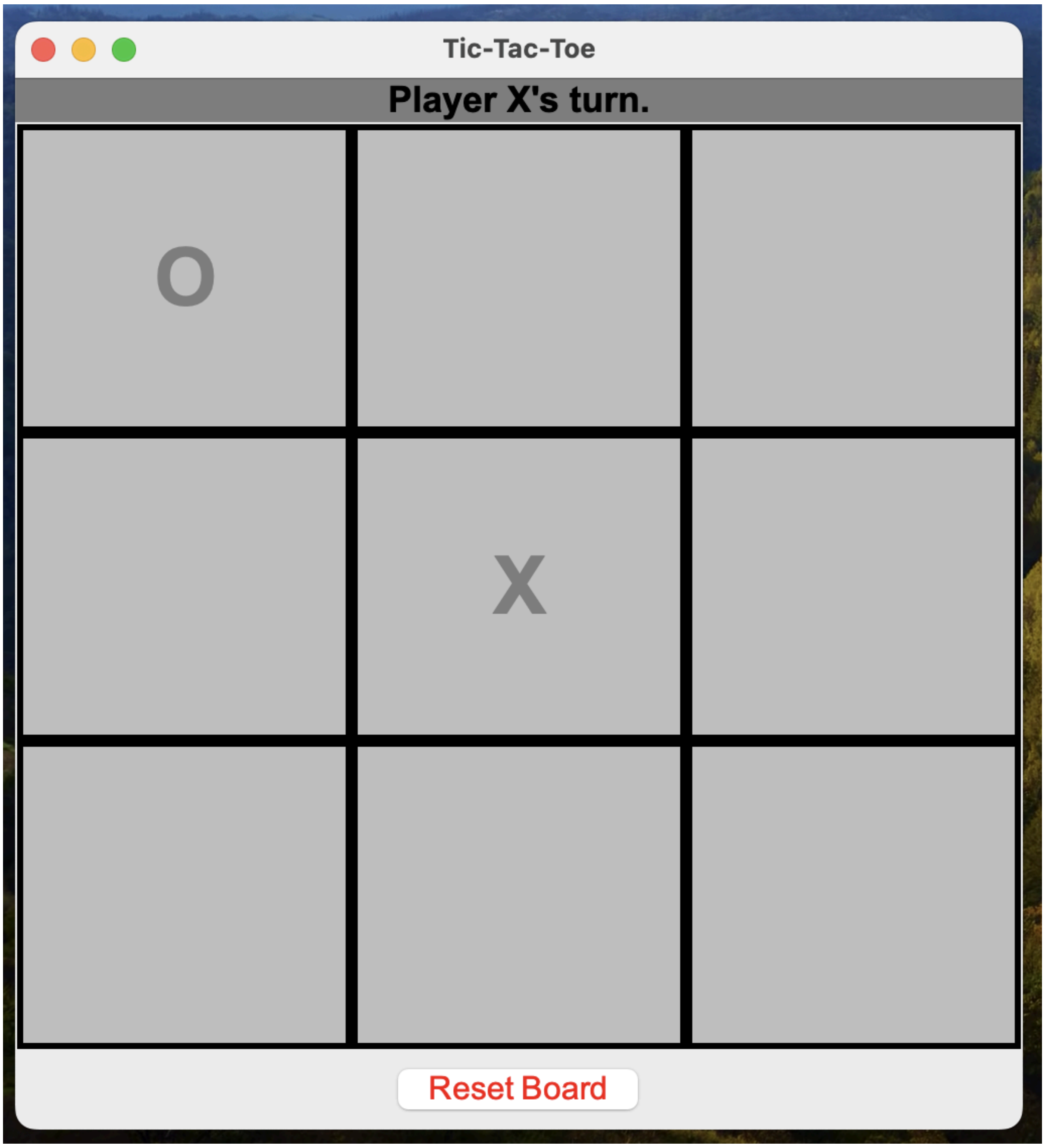

Tic-Tac-Toe
Player O's turn.
X
O
X
X
O
Reset Board

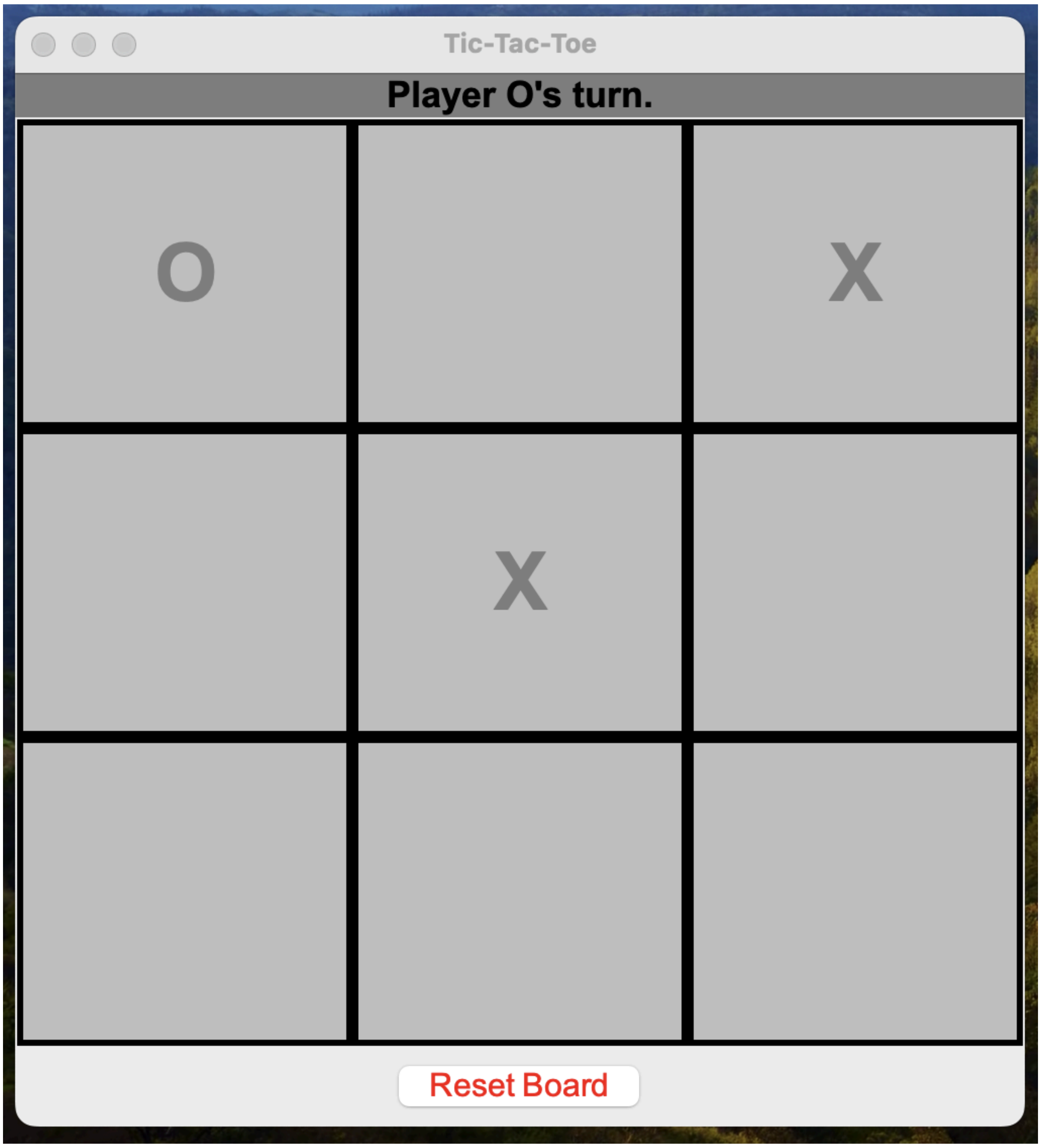

Tic-Tac-Toe
Player X's turn.
X
O
X
X
O
O
Reset Board



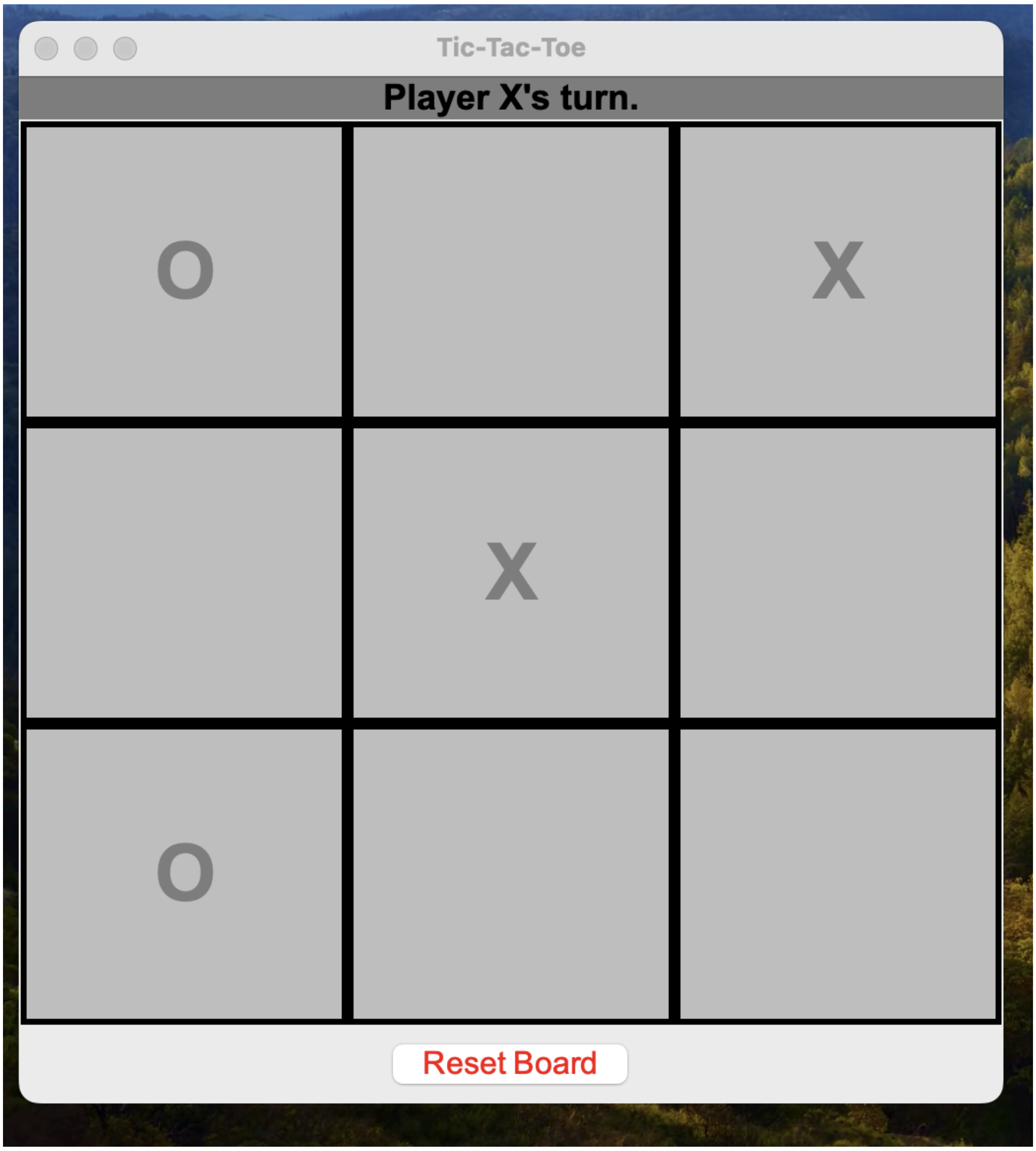
Game resulting in a draw:

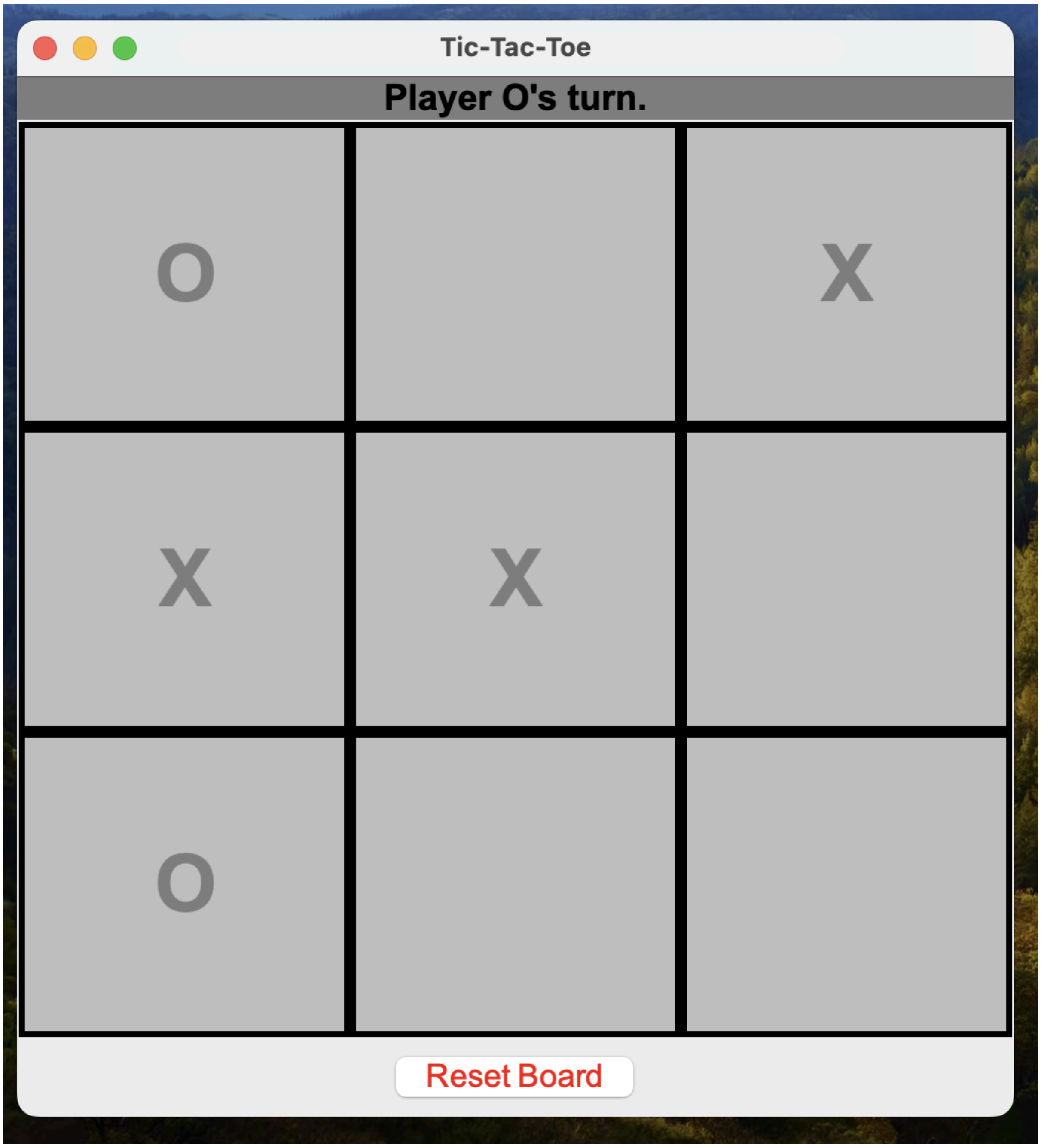

Tic-Tac-Toe
Player O's turn.
X
Reset Board

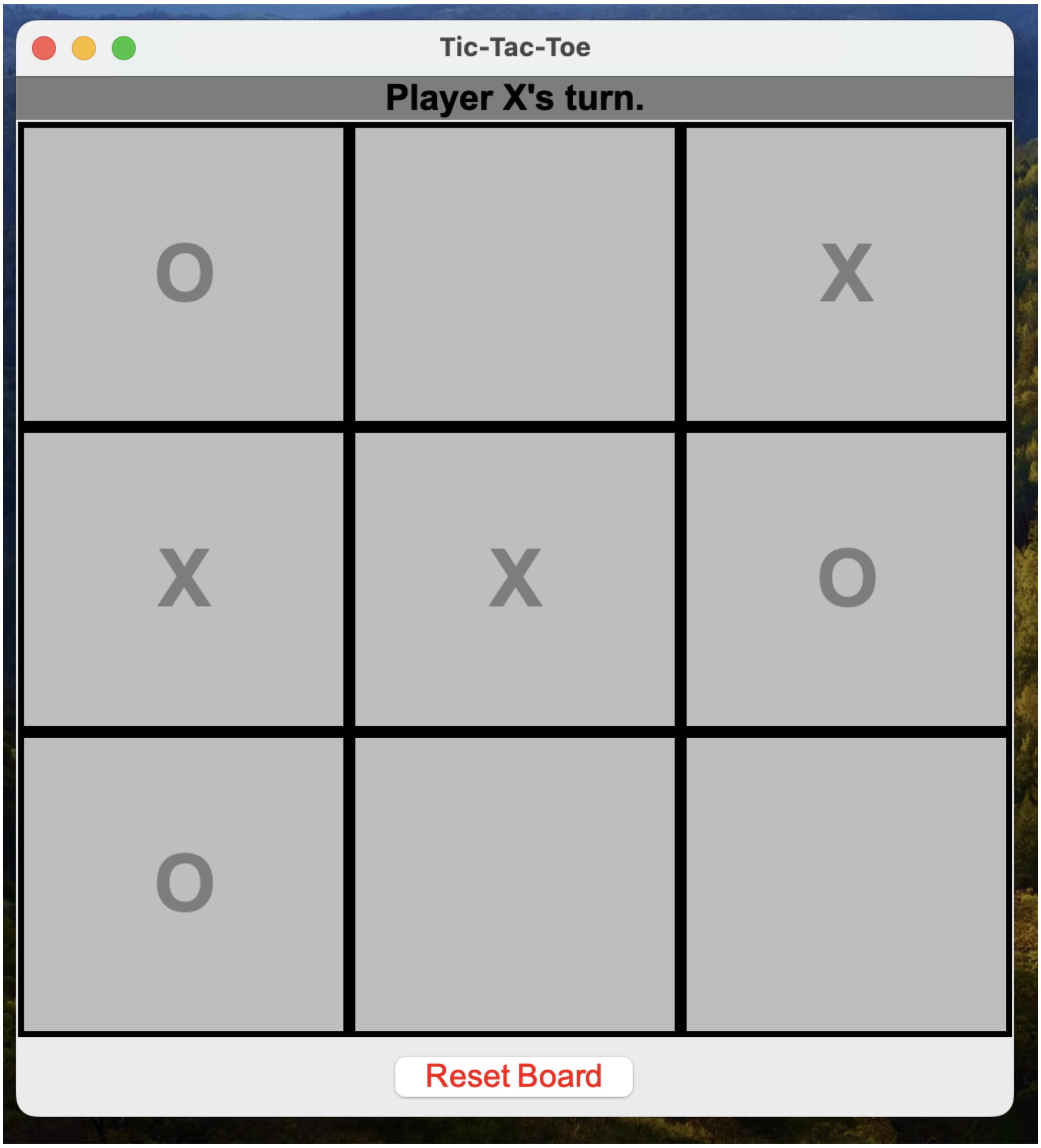

Tic-Tac-Toe
Player X's turn.
O
X
Reset Board

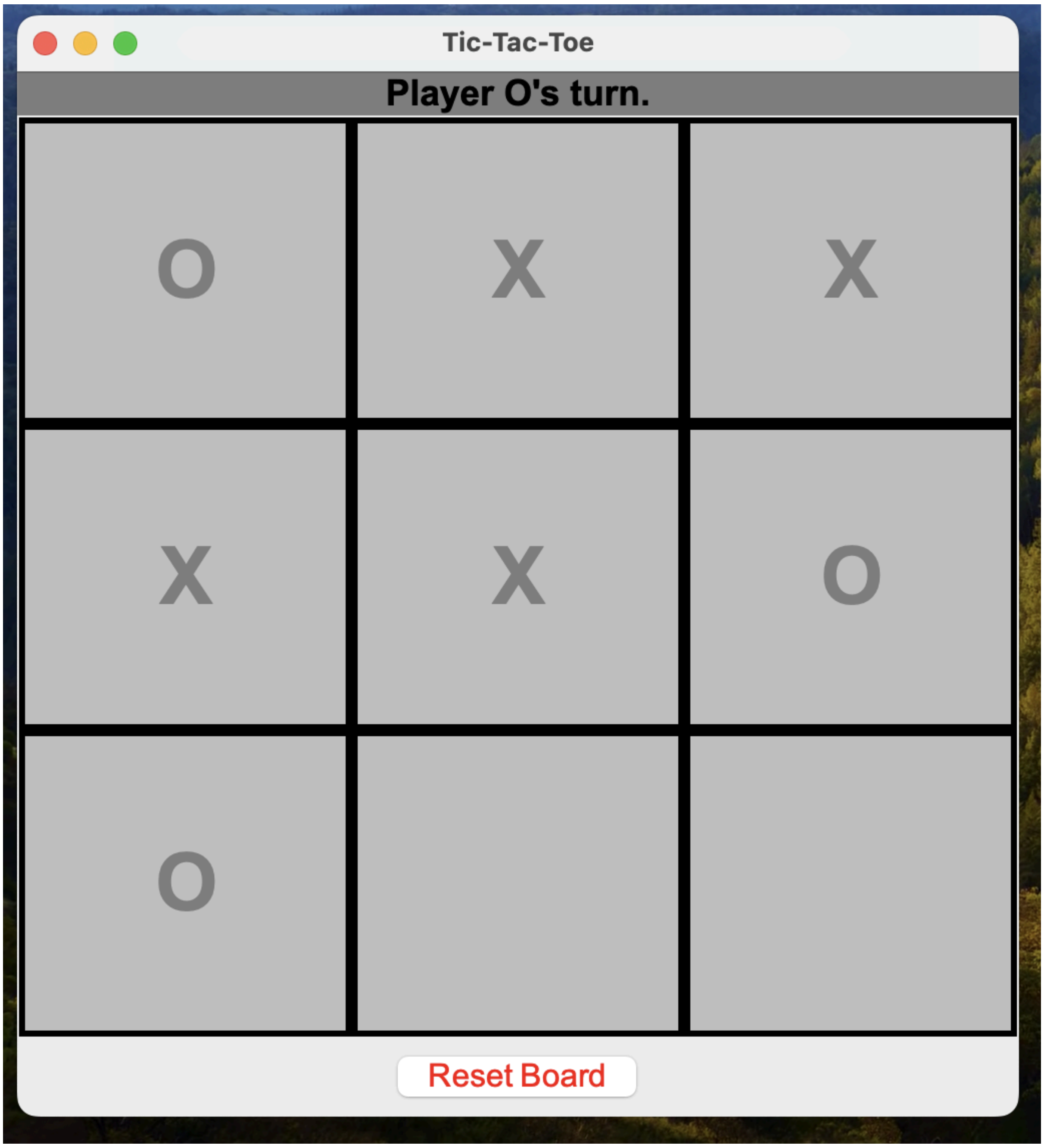


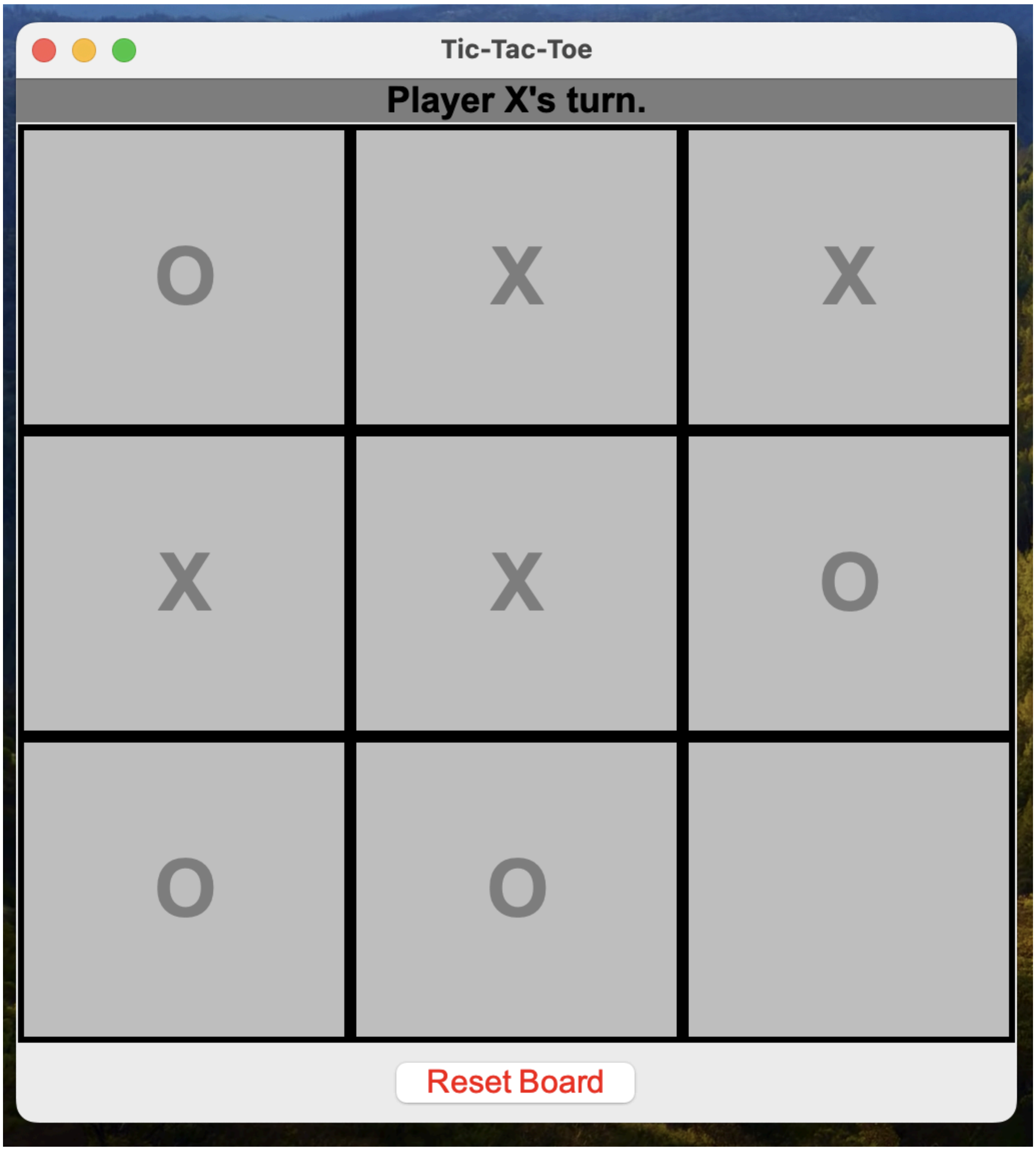
Player O's turn.

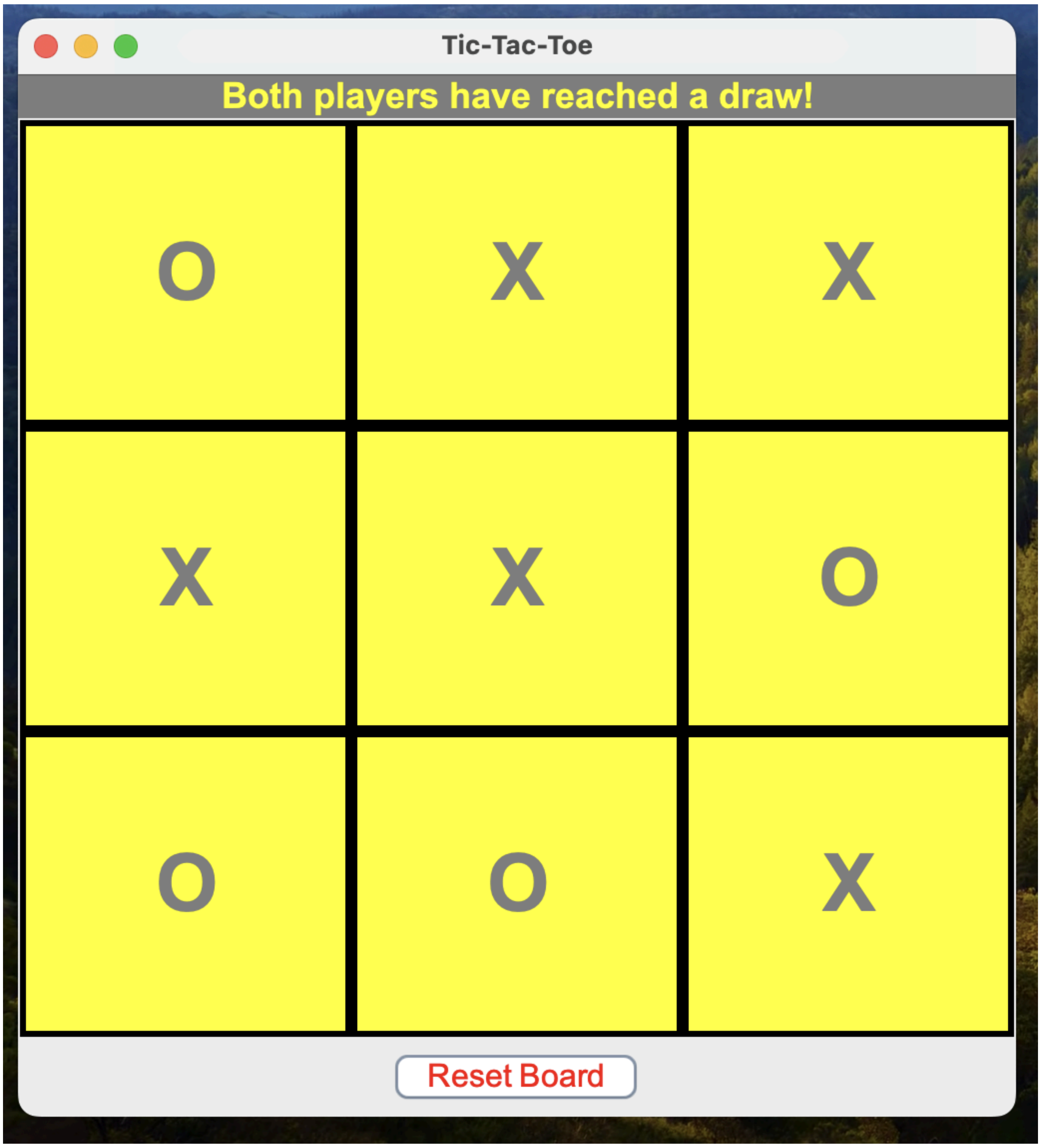

Tic-Tac-Toe
Player X's turn.
O
X
X
O
Reset Board


Tic-Tac-Toe
Player O's turn.
O
X
X
X
O
Reset Board


Tic-Tac-Toe
Player X's turn.
O
X
X
X
O
O
Reset Board


Tic-Tac-Toe
Player O's turn.
O
X
X
X
X
O
O
Reset Board


Tic-Tac-Toe
Player X's turn.
O
X
X
X
X
O
O
O
Reset Board


Tic-Tac-Toe

Both players have reached a draw!

O	X	X
X	X	O
O	O	X