Exercise POSE.22 – Tic Tac Toe

2024-04-25

Instructions

- Copy the starter project in a separate directory
- Complete all methods according to the instructions and XML comments
- Solve the following tasks and make sure that the provided tests are all passing
- Make sure to run `CleanSolutionDir.ps1` before you pack and hand in your solution

1 Tic Tac Toe

You are going to implement the game Tic Tac Toe.

To make it a little more interesting we are going to use the provided `Board` utility `class` to actually display the game state in its own window. Input will still be handled by the console, though. See fig. 1 for a sample run of the application.

Using the `Board`

Initializing the board as well as showing the window etc. is handled for you (this time). You'll have to make use of the following methods of the `Board`:

- `Board.SetCellContent`
  - Sets the text (value) for one specific cell of the board's grid
  - Takes four parameters:
    - `row`: Row index, 0 based
    - `col`: Column index, 0 based
    - `text`: The value to display – 'X', 'O' or a number in our case
    - `color`: The color in which to display the `text`[1]
- `Board.GetCellContent`
  - Get the text (value) from one specific cell of the board's grid
  - Take two parameters:
    - `row`: Row index, 0 based
    - `col`: Column index, 0 based

Methods

- `InitPositions`
  - Sets up the playing field in such a way, that each cell contains a number (no player 'stones' put down yet)
  - The numbers are arranged in the same way as on a num block of a keyboard (7,8,9,4,5,6,1,2,3)
  - The numbers are colored using 'LightGrey'
- `CheckPositionValid`
  - Ensures that the passed position (based on the numbers set in the `InitPositions` method) is valid

[1] Possible colors are `Brushes.Red`, `Brushes.Green`, `Brushes.Black` & `Brushes.LightGray`.

  - Valid means it is within the range of possible values [1,9]
- `CheckPosition`
  - Checks if a position is valid (= within range) and *still empty*
    - *Empty* means no player 'stone' has been placed so far
- `GetStonePosition`
  - Reads a position for placing a 'stone' from the console
  - A stone can only be placed on a valid and empty position
    - Repeat input until a valid position is entered
  - The players alternate in turn, picking positions
    - This logic does *not* have to be handled by this method
- `GetRowFromPosition`
  - Based on the given position (= number) the row is determined
  - For example: the number 8 is in row 0, the number 4 is in row 1
- `GetColFromPosition`
  - Based on the given position and (to make it easier) the given row the column is determined
  - For example: the number 6 is in column 2 and the number 2 is in column 1
- `GetPlayerIndexFromCell`
  - Determines which stone is located at the specified (via row & column) cell
  - If the cell is empty or the location invalid -1 is returned
  - For 'X' return 1 and for 'O' return 0
- `GetFieldValues`
  - Retrieves all (textual) values from the `Board`
- `CheckWinner`
  - Determines who (= player 0 or player 1) has won the game or if it ended in a draw (-1)
  - Possible, winning combinations are:
    - The two diagonales
    - A full row
    - A full column

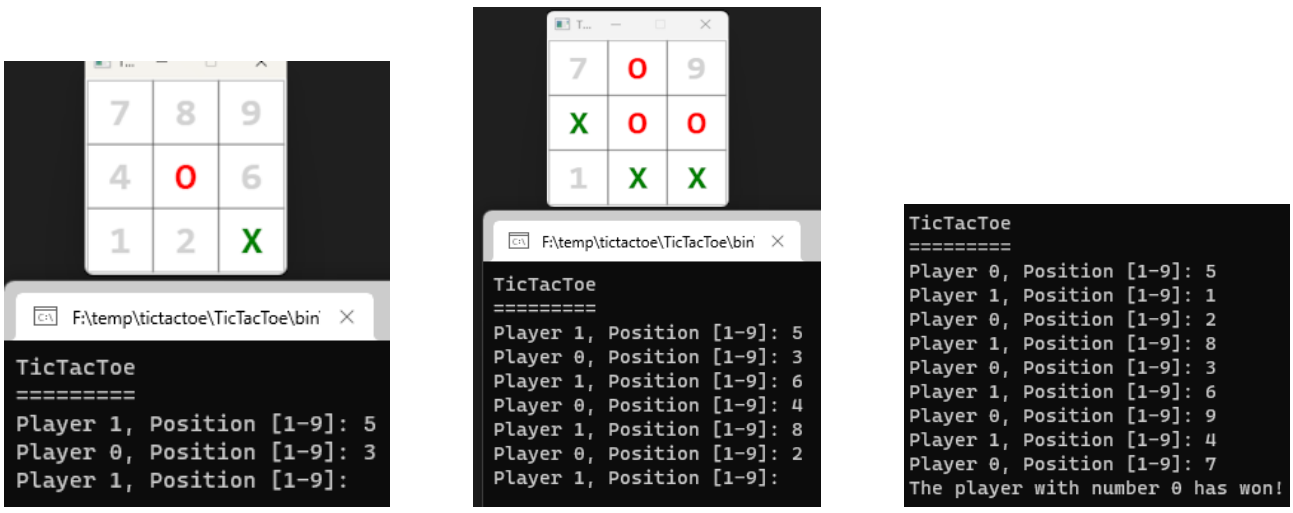

Figure 1: TicTacToe – Sample Run