Assignment 3

Software Development 2022

Department of Computer Science
University of Copenhagen

Aditya Fadhillah <hjg708@alumni.ku.dk>

Friday, February 25, 14:30

Introduction

In the assignment that has been given to me I am required to go through testing and implementing the missing functionality of the game TicTacToe. For this assignment I have used Visual Studio Code on the .NET Core version. The missing functionalities is located in the TicTacToe file where the actual game is run and in the TicTacToeTest file where the various methods of the main file are tested. This assignment focuses on a test-driven approach, that means I need to write tests for the missing methods before actually implementing the methods themselves. For my first task I need to extend the code for the movements of the cursor, the task consists of five functions and their tests. For my second task I need to extend the code for the ending condition of the game.

User's Guide

Assuming that the user received the file as a .zip file, they would first need to extract the contents of the .zip file. After extracting the files, the user can run the game TicTacToe by go to the TicTacToe file directory in their terminal, and then type in: 'dotnet run'. If all goes well the user will then see a 3 x 3 board. This board is where the game will be played. The cross player will always start first. For the cross player to choose a location they will be using: 'w','a','s' and 'd'. For the naught player it is: 'i','j','k' and 'l'. And to confirm a location and ending a move press: 'spacebar'. The game will end when a player have made a line with three of their symbols. The line can be horizontal, vertical and diagonal. When a player have achieved this the screen will display: '⟨ player symbol ⟩ Wins!'. When the board is full be there are still no winner the screen will display: 'The game was a tie!'.

Implementation

Implementing codes for tests functions in CursorTest.cs

For the first task I decided to start with implementing codes for the tests as this assignment focuses on the testing aspect of coding.

```
1 [Test]
2 public void MoveUpTest() {
3     Setup();
4     cursor.MoveUp();
5     var curPos = cursor.position;
6     cursor.MoveUp();
7     Assert.True(cursor.position.X == curPos.X && cursor.position.Y == curPos.Y);
8 }
```

MoveUpTest() will be an example of my implementation of a test function, this function starts by moving the cursor to the middle of the board (1,1). It will then run the MoveUp() method. And by using a variable that tracks the position of the cursor. After another MoveUp() call I can then compare the position of the cursor with the variable. This test function test whether or not the position of the cursor has changed when the MoveUp() method is called.

Implementing codes for methods in Cursor.cs

```
public void MoveUp() {
    if(Y > min) {
        Y -= 1;
    }
}
```

MoveUp() will be an example of the implementation of methods. The MoveUp() method works very simple. It checks if the Y value of the cursor is larger than the minimum value of the board. If it is true the Y value of the cursor is reduced by one. The other move methods will function similarly to MoveUp(). The other part is to implement MoveCursor(Input input)-function. I solved this task by using a Switch-statement. I made it so the InputType.PerformMove case returns False and all other cases return True.

Implementing codes for tests functions in BoardCheckerTest.cs

```
[Test]
public void DiagonalWinTest() {
    Setup();
    board.TryInsert(0, 0, PlayerIdentifier.Naught);
    board.TryInsert(1, 1, PlayerIdentifier.Naught);
    board.TryInsert(2, 2, PlayerIdentifier.Naught);
    Assert.AreEqual(boardChecker.CheckBoardState(board), BoardState.Winner);
}
```

To see how I implement the codes for my tests I choose DiagonalWinTest() as an example. This test function insert three positions with the naught identifier. This will make a diagonal line through the board. Assert.AreEqual will then checks if CheckBoardState(board) has the same value as BoardState.Winner.

Implementing codes for methods in BoardChecker.cs

```
private bool IsDiagWin(Board board) {
    if(board.Get(0,0) == PlayerIdentifier.Naught
    && board.Get(1,1) == PlayerIdentifier.Naught
    && board.Get(2,2) == PlayerIdentifier.Naught) {
        return true;
    }
    else if(board.Get(2,0) == PlayerIdentifier.Naught
    && board.Get(1,1) == PlayerIdentifier.Naught
    && board.Get(0,2) == PlayerIdentifier.Naught) {
        return true;
    }
```

```
12      else if(board.Get(0,0) == PlayerIdentifier.Cross
13      ---------------------||---------------------
14      else {
15          return false;
16      }
17  }
```

The implementation of the IsDiagWin() method works by exhausting the possible position the symbols can be placed diagonally so that all three places have the same symbols. The method will return true if that is the case. To save space I have not shown the whole code, but it would be the same as the first 9 lines but the PlayerIdentifier is changed to Cross. The other part of the task is to implement the CheckBoardState(Board board) method. This method changes the BoardState to Winner if a win condition if fulfilled. It changes to Tied if the board is full without a winner, otherwise it will return BoardState.Inconclusive.

Evaluation

Other than the nine obligatory tests that I need to implement in the assignment, I also add four new test for the BoardCheckerTest. The first three tests is for the opposite PlayerIdentifier winning condition, this is to test that both player type can win the game. The last new test is to test that a game will have a winner even when the board is full. I have only taken into consideration that the board is 3 x 3 in size and that it is not changeable. A normal game of TicTacToe is 3 x 3, and that is why I decided that the board size cannot be changed. That is why my method for BoardChecker simply check the possible position for a win condition in a 3 x 3 board. Of course if the size of the board is indeed changeable, I would need to make a completely new method for BoardChecker. This new method will be using variables to keep track on when a line has ended so that the code can check the next line.

Conclusion

With the implementation I have applied to the files TicTacToe and TicTacToeTest, I have made it so that I can run and end the game without issues and according to the specification that have been given to me. I have also explained my considerations during the implementation process,