Refactoring – How to do it quick and easy!

Or maybe how not to do it??

TicTacToe Kata

- How to refactor the TicTacToe Kata
- Guidelines....pffft....Best practice.....pffft....don't need that
- Start from the beginning and let the flow guide you

Logical First Step

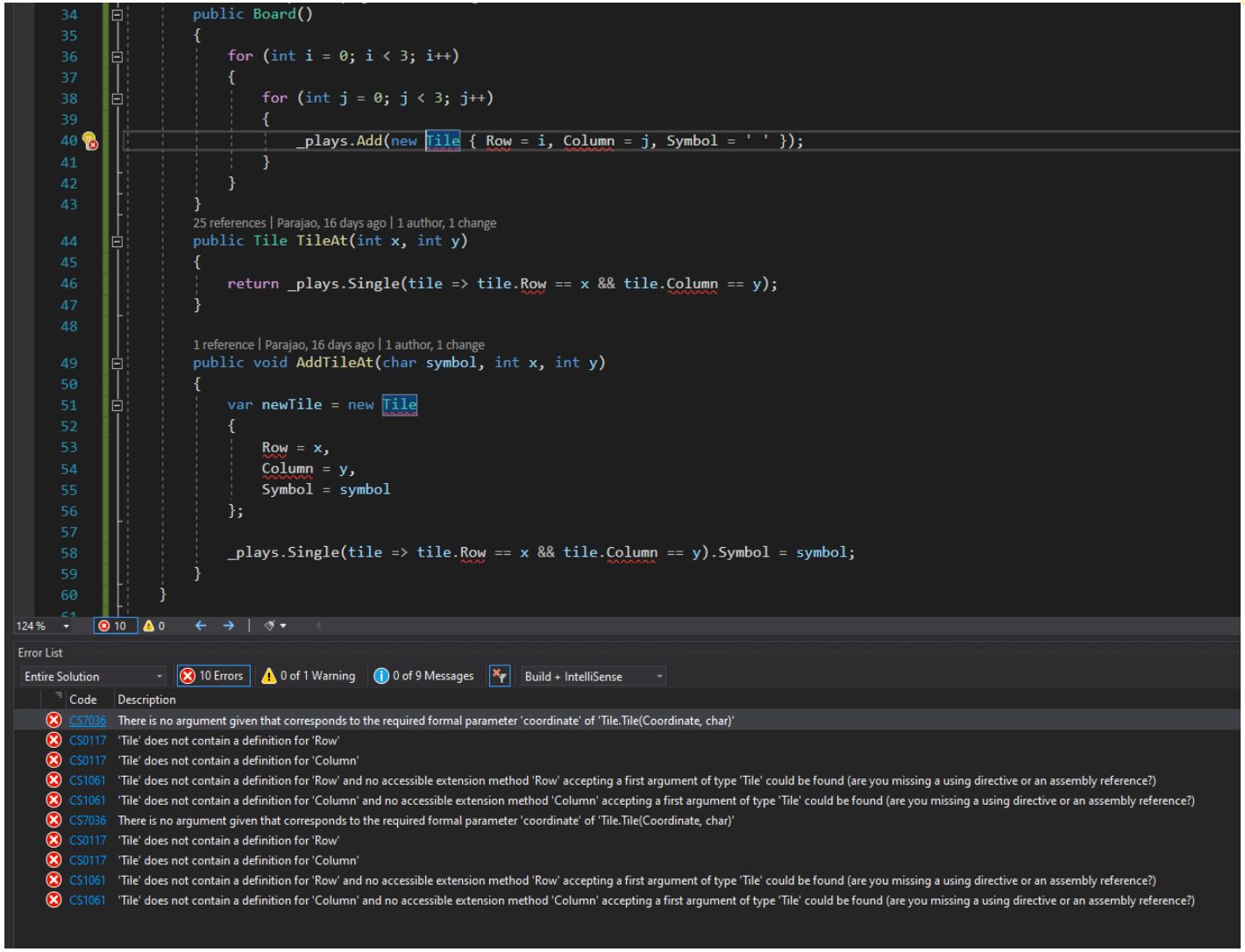
- Introduce new Coordinate class
- Slap on some constructors
- Rename x --> Row
- Rename y --> Column

```
public class Tile
{
    2 references | 0 changes | 0 authors, 0 changes
    public Tile(Coordinate coordinate, char symbol)
    {
        Coordinate = coordinate;
        Symbol = symbol;
    }
    28 references | Parajao, 16 days ago | 1 author, 1 change
    public char Symbol { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public Coordinate Coordinate { get; set; }
}

3 references | 0 changes | 0 authors, 0 changes
public class Coordinate
{
    0 references | 0 changes | 0 authors, 0 changes
    public Coordinate(int row, int column)
    {
        Row = row;
        Column = column;
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public int Row { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public int Column { get; set; }
}
```

Not that bad, I can fix this.....

- Only 10 errors
- Just have to change to using the constructor instead.
- And change to tile.Coordinate.Row/Column

- EasyPeasy



That was easy, next challenge please.

```
public Tile TileAt(int x, int y)
{
    return _plays.Single(tile => tile.Coordinate.Row == x && tile.Coordinate.Column == y);
}

1 reference | Parajao, 16 days ago | 1 author, 1 change
public void AddTileAt(char symbol, int x, int y)
{
    var newTile = new Tile(new Coordinate(x, y), symbol);
    _plays.Single(tile => tile.Coordinate.Row == x && tile.Coordinate.Column == y).Symbol = symbol;
}
```

```
26 references | Parajao, 16 days ago | 1 author, 1 change
public Tile TileAt(int x, int y)
{
    return _plays.Single(tile => tile.Coordinate.Row == x && tile.Coordinate.Column == y);
}

1 reference | Parajao, 16 days ago | 1 author, 1 change
public void AddTileAt(char symbol, int x, int y)
{
    var newTile = new Tile(new Coordinate(x, y), symbol);
    TileAt(newTile.Coordinate.Row, newTile.Coordinate.Column).Symbol = symbol;
}
```

- So next one is an obvious duplication
- Easy to fix

Soo what's left?

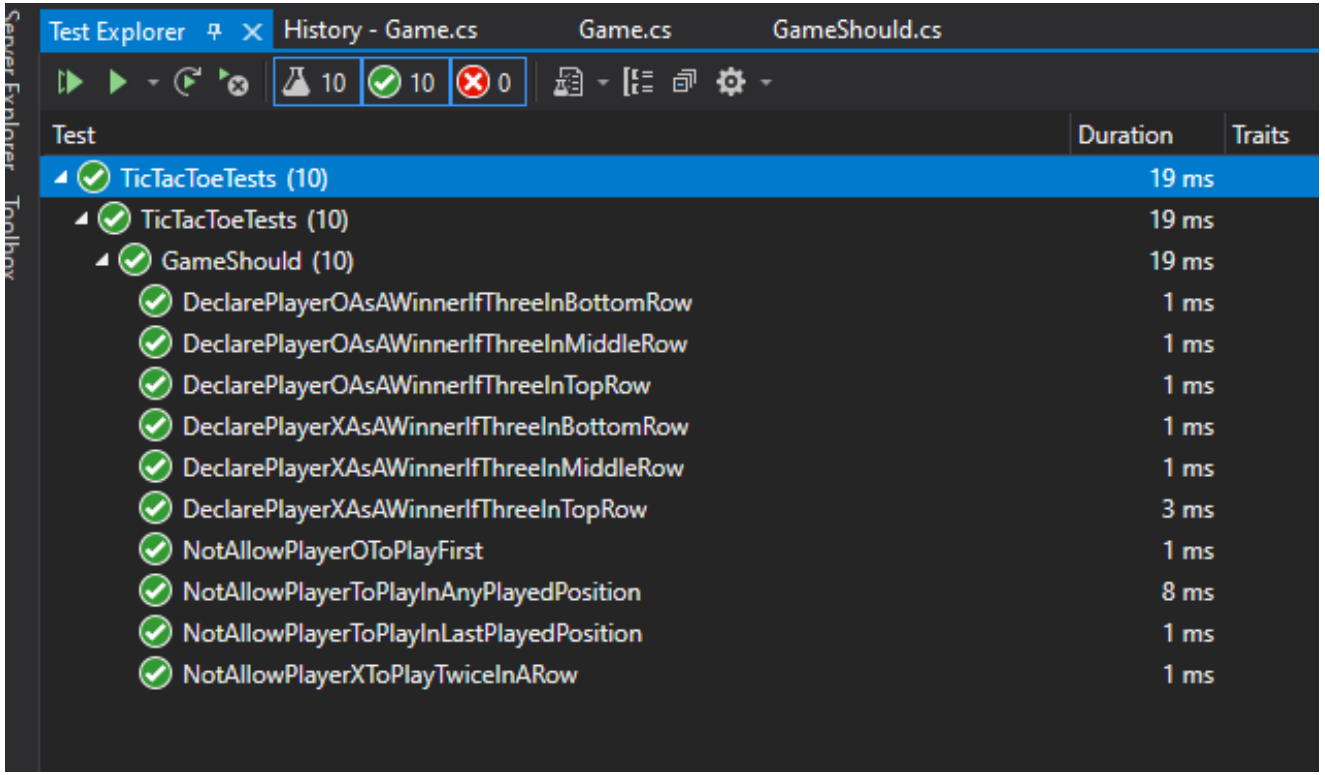
- Only thing left is to fix up the Winner method
- Just extract a couple of helper methods and put them in the Board class

```
public char Winner()
{   //if the positions in first row are taken

    if (_board.IsColumnTaken(0) && _board.HasSameSymbol(0))
        return _board.TileAt(0, 0).Symbol;

    //if the positions in first row are taken
    if (_board.IsColumnTaken(1) && _board.HasSameSymbol(1))
        return _board.TileAt(1, 0).Symbol;

    //if the positions in first row are taken
    if (_board.IsColumnTaken(2) && _board.HasSameSymbol(2))
        return _board.TileAt(2, 0).Symbol;

    return ' ';
}
```

```
3 references | 0 changes | 0 authors, 0 changes
internal bool IsColumnTaken(int column)
{
    return TileAt(column, 0).Symbol != ' ' &&
           TileAt(column, 1).Symbol != ' ' &&
           TileAt(column, 2).Symbol != ' ';
}

3 references | 0 changes | 0 authors, 0 changes
internal bool HasSameSymbol(int column)
{
    return TileAt(column, 0).Symbol ==
           TileAt(column, 1).Symbol &&
           TileAt(column, 2).Symbol ==
           TileAt(column, 1).Symbol;
}
```

Job done

- All tests are green



What went wrong?

- Started refactoring right away from the top
- Got distracted by implementation details
- Only saw the obvious code smells, and not even all of them
- Even managed to introduce new ones

The right way

- Go fast by going slow (First step to solve a problem, is to know the problem)
  - Get an overview
  - Add comments where you see code smells, or suspect there may be an issue
- Divide and Conquer (Can you split a big problem into smaller problems?)
  - Reduce complexity
  - Extract methods
- Find a natural home for the code
  - Redistribute the behavior
  - Use feature envy to determine where a block of code belongs
- Make new fancy code
  - When all that is done it's easier to introduce new abstractions

Thank you!

Ronny Navelsaker
ronny.navelsaker@bouvet.no