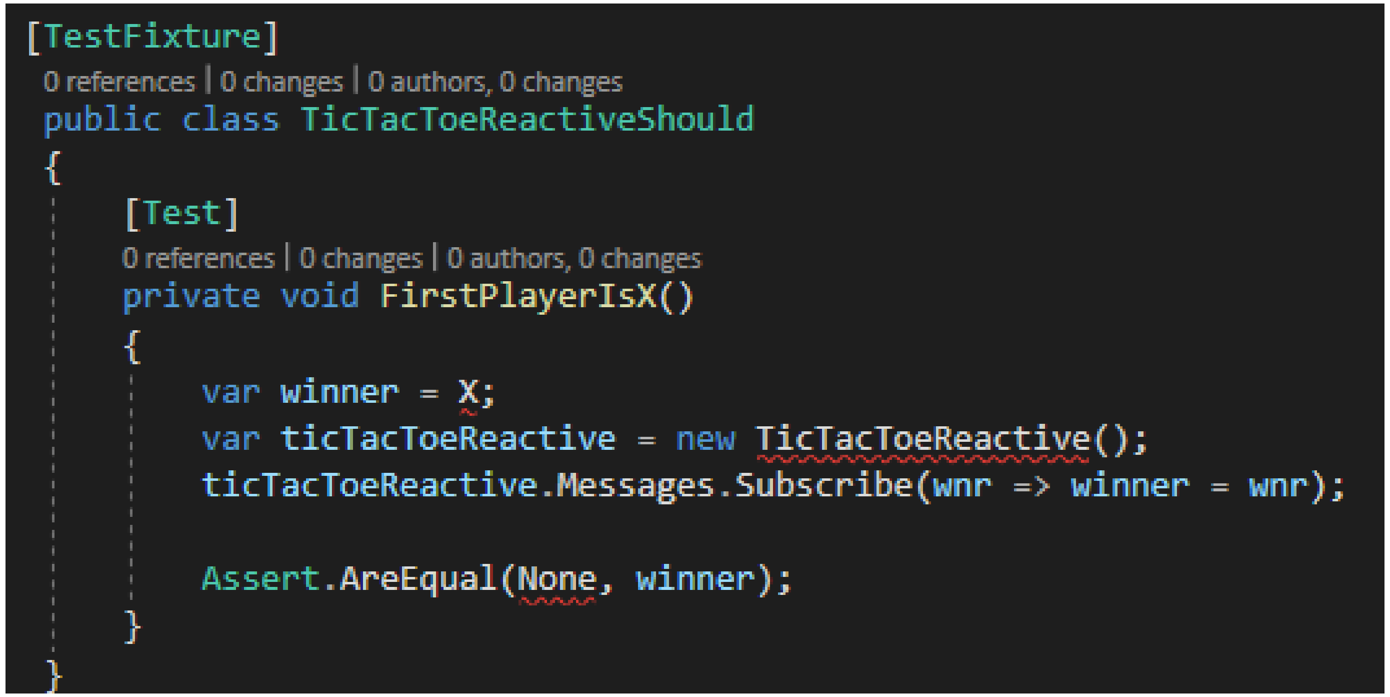
TDD
TicTacToe kata
using
Reactive Extensions

Henning Torsteinsen

What’s this?

- **Applying TDD and object calisthenics to the Tic Tac Toe kata.**
- **Secret ingredient:**
  Using Reactive Extensions to only expose a single public property.
- **This is an experiment to completely hide the implementation details, and reduce dependencies as much as possible.**

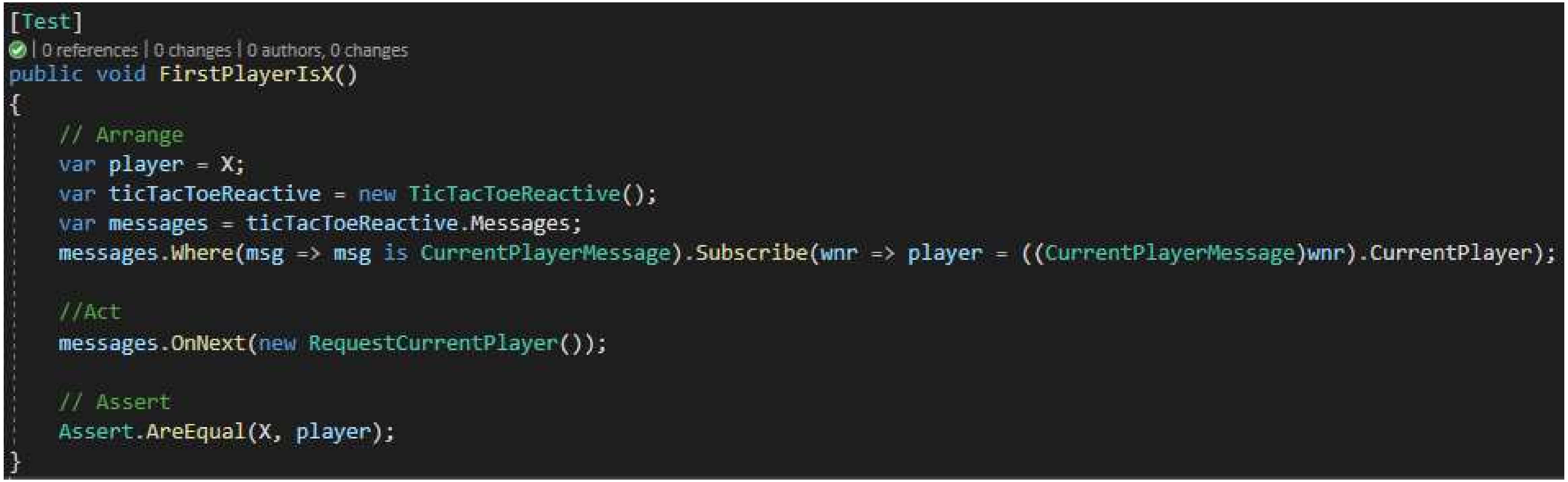
Start with at failing test...



```
[TestFixture]
 0 references | 0 changes | 0 authors, 0 changes
 public class TicTacToeReactiveShould
 {
     [Test]
     0 references | 0 changes | 0 authors, 0 changes
     private void FirstPlayerIsX()
     {
         var winner = X;
         var ticTacToeReactive = new TicTacToeReactive();
         ticTacToeReactive.Messages.Subscribe(wnr => winner = wnr);

         Assert.AreEqual(None, winner);
     }
 }
```

... by pretending the class is already there.

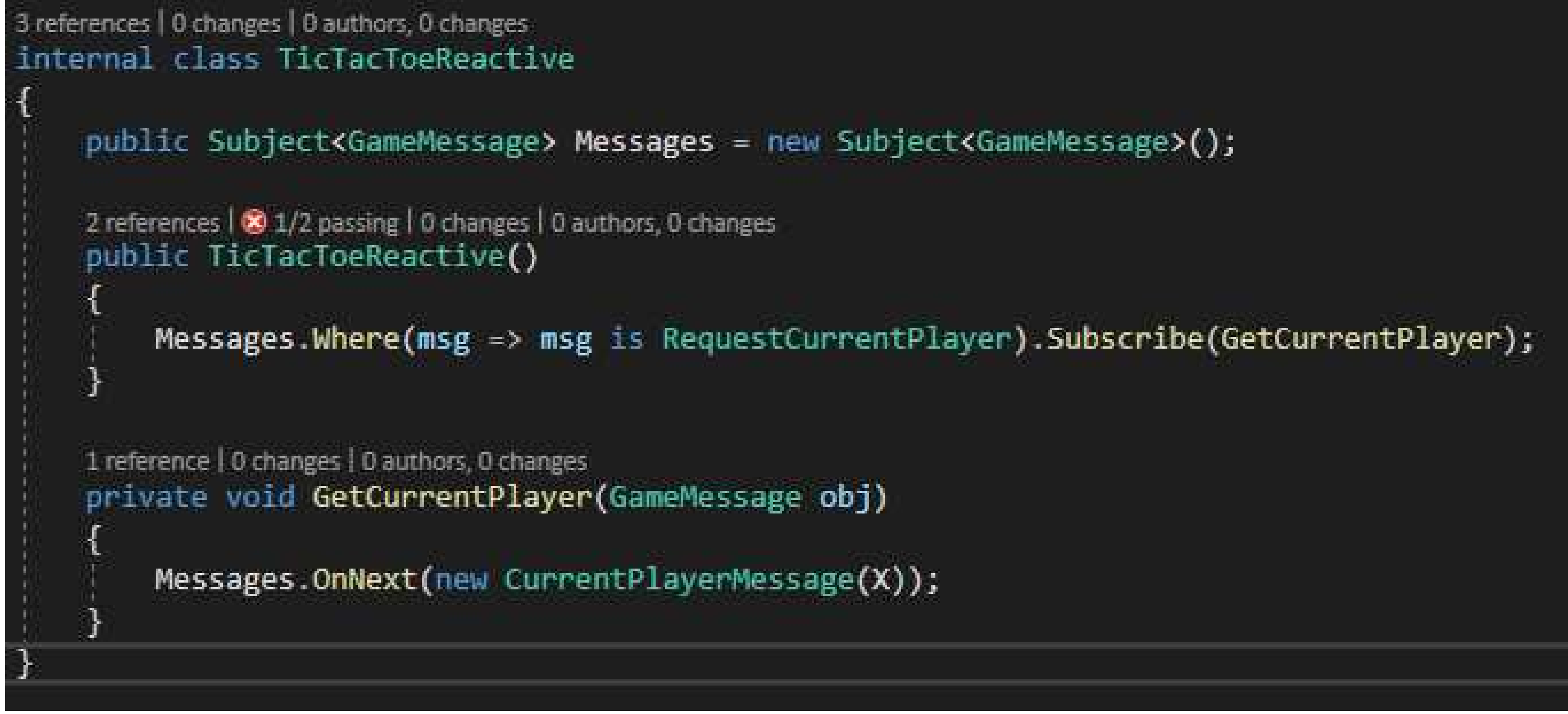
Fill in the blanks, and implement just enough to make the test pass.



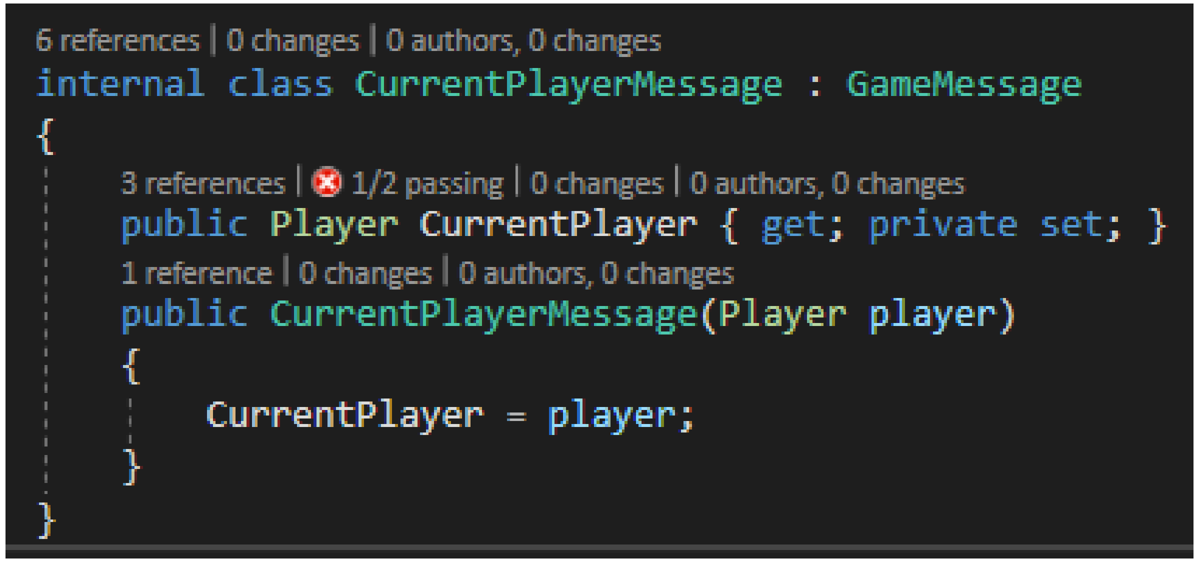
```
[Test]
| 0 references | 0 changes | 0 authors, 0 changes
public void FirstPlayerIsX()
{
    // Arrange
    var player = X;
    var ticTacToeReactive = new TicTacToeReactive();
    var messages = ticTacToeReactive.Messages;
    messages.Where(msg => msg is CurrentPlayerMessage).Subscribe(wnr => player = ((CurrentPlayerMessage)wnr).CurrentPlayer);

    //Act
    messages.OnNext(new RequestCurrentPlayer());

    // Assert
    Assert.AreEqual(X, player);
}
```

Sidenote…

- **More code needed to get first test passing when using Reactive Extensions**



```
3 references | 0 changes | 0 authors, 0 changes
internal class TicTacToeReactive
{
    public Subject<GameMessage> Messages = new Subject<GameMessage>();

    2 references | 1/2 passing | 0 changes | 0 authors, 0 changes
    public TicTacToeReactive()
    {
        Messages.Where(msg => msg is RequestCurrentPlayer).Subscribe(GetCurrentPlayer);
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private void GetCurrentPlayer(GameMessage obj)
    {
        Messages.OnNext(new CurrentPlayerMessage(X));
    }
}
```

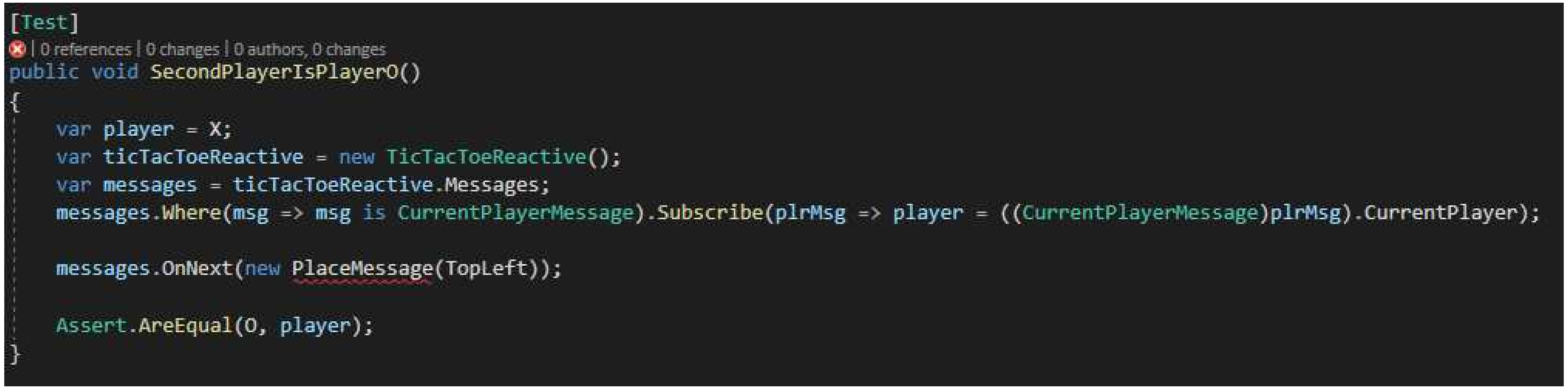


```
6 references | 0 changes | 0 authors, 0 changes
internal class CurrentPlayerMessage : GameMessage
{
    3 references | 1/2 passing | 0 changes | 0 authors, 0 changes
    public Player CurrentPlayer { get; private set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public CurrentPlayerMessage(Player player)
    {
        CurrentPlayer = player;
    }
}
```

Then refactor…

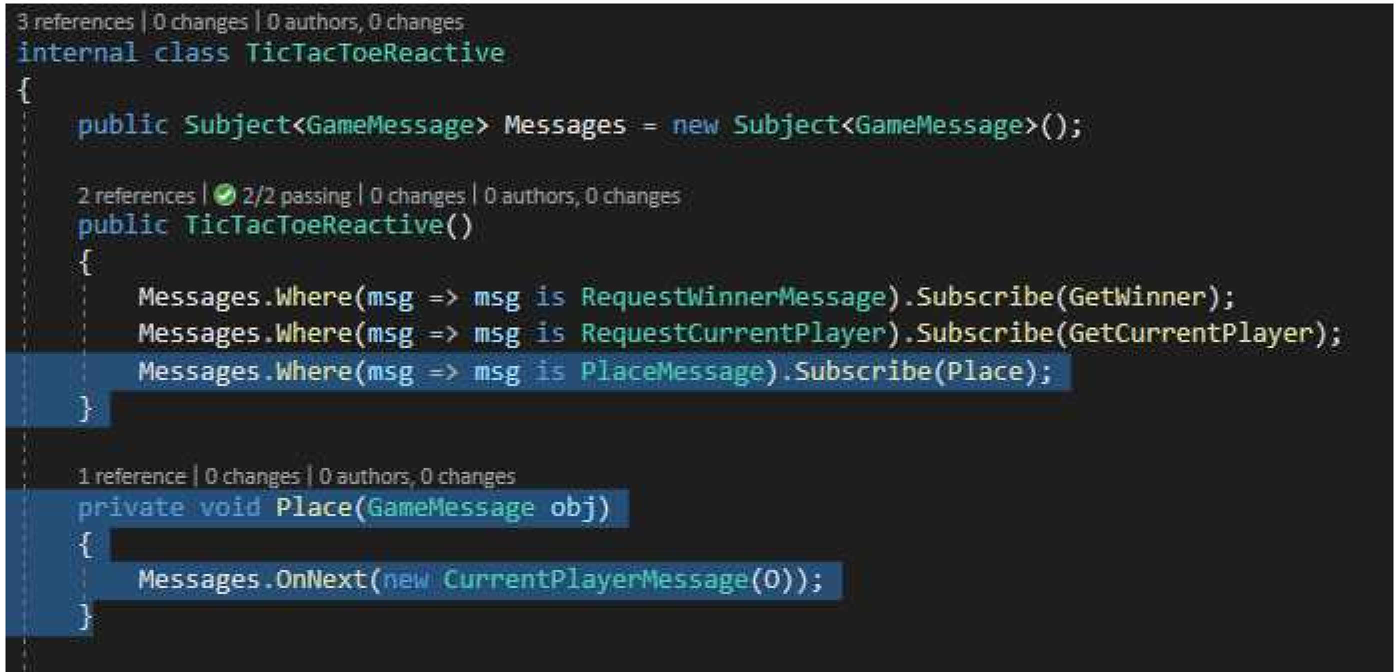
- **Not much to do yet but move all classes out into their own file in correct folder.**

Then add a new failing test…



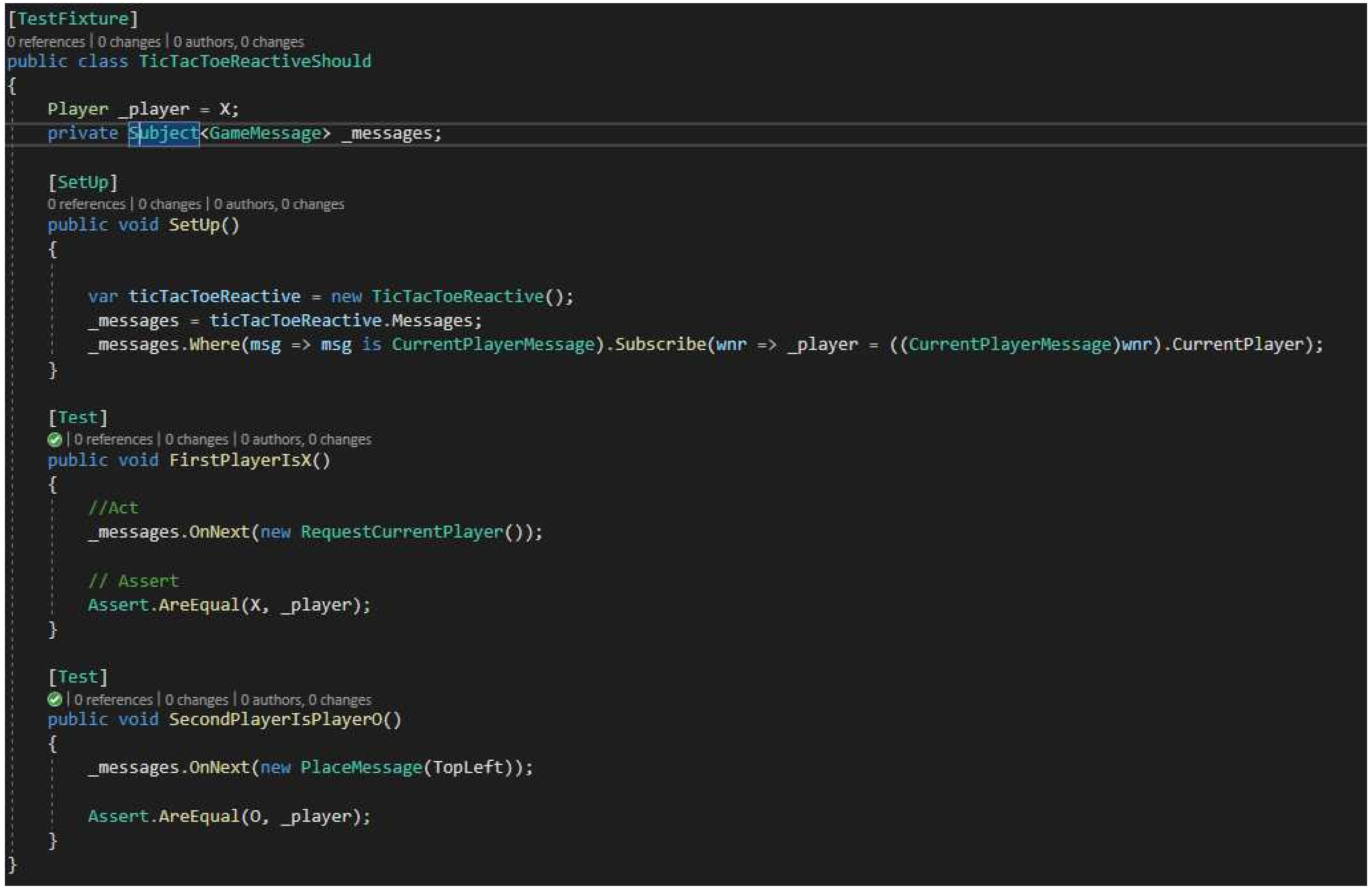
```
[Test]
| 0 references | 0 changes | 0 authors, 0 changes
public void SecondPlayerIsPlayerO()
{
    var player = X;
    var ticTacToeReactive = new TicTacToeReactive();
    var messages = ticTacToeReactive.Messages;
    messages.Where(msg => msg is CurrentPlayerMessage).Subscribe(plrMsg => player = ((CurrentPlayerMessage)plrMsg).CurrentPlayer);

    messages.OnNext(new PlaceMessage(TopLeft));

    Assert.AreEqual(O, player);
}
```

… and code to make it work



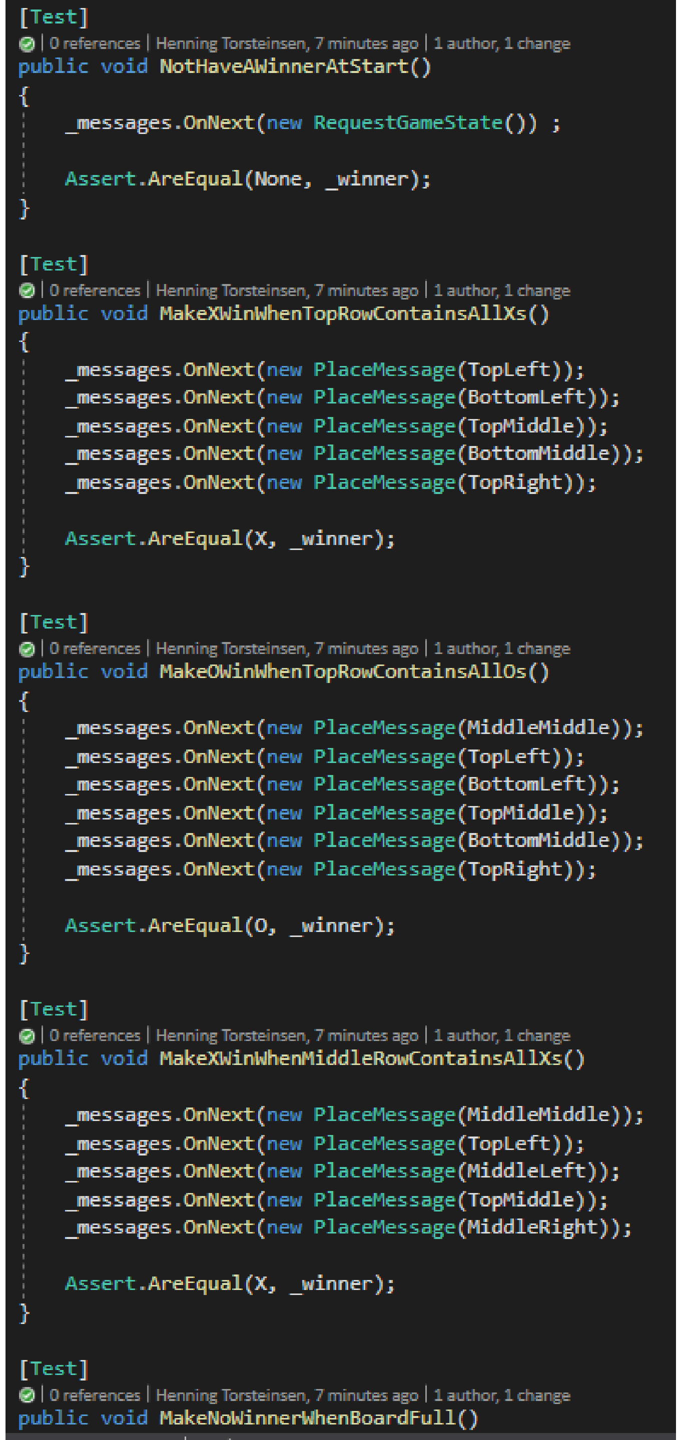
```
3 references | 0 changes | 0 authors, 0 changes
internal class TicTacToeReactive
{
    public Subject<GameMessage> Messages = new Subject<GameMessage>();

    2 references | 2/2 passing | 0 changes | 0 authors, 0 changes
    public TicTacToeReactive()
    {
        Messages.Where(msg => msg is RequestWinnerMessage).Subscribe(GetWinner);
        Messages.Where(msg => msg is RequestCurrentPlayer).Subscribe(GetCurrentPlayer);
        Messages.Where(msg => msg is PlaceMessage).Subscribe(Place);
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private void Place(GameMessage obj)
    {
        Messages.OnNext(new CurrentPlayerMessage(O));
    }
```

Refactor common Arrange code



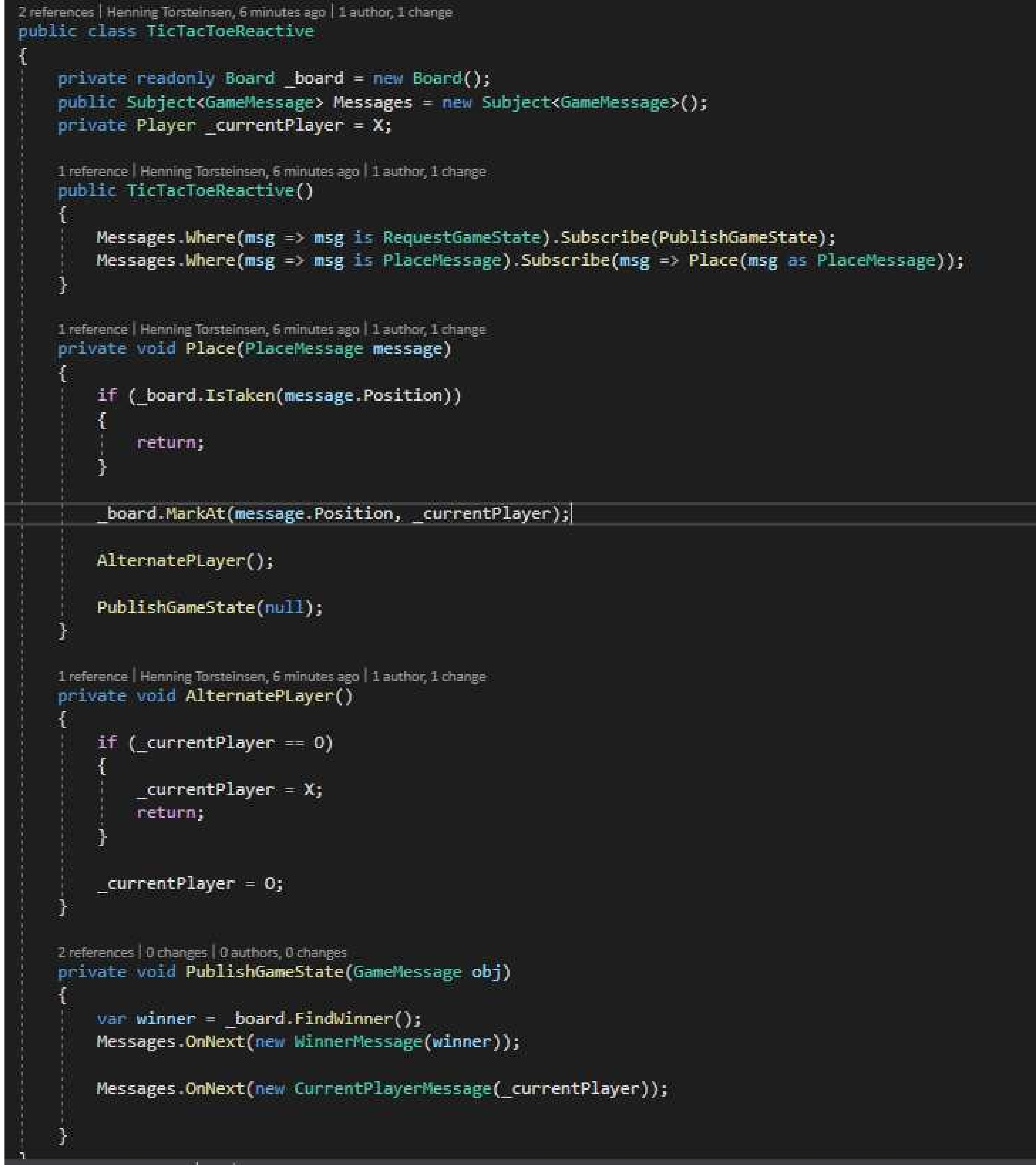
```
[TestFixture]
0 references | 0 changes | 0 authors, 0 changes
public class TicTacToeReactiveShould
{
    Player _player = X;
    private Subject<GameMessage> _messages;

    [SetUp]
    0 references | 0 changes | 0 authors, 0 changes
    public void SetUp()
    {

        var ticTacToeReactive = new TicTacToeReactive();
        _messages = ticTacToeReactive.Messages;
        _messages.Where(msg => msg is CurrentPlayerMessage).Subscribe(wnr => _player = ((CurrentPlayerMessage)wnr).CurrentPlayer);
    }

    [Test]
    | 0 references | 0 changes | 0 authors, 0 changes
    public void FirstPlayerIsX()
    {
        //Act
        _messages.OnNext(new RequestCurrentPlayer());

        // Assert
        Assert.AreEqual(X, _player);
    }

    [Test]
    | 0 references | 0 changes | 0 authors, 0 changes
    public void SecondPlayerIsPlayerO()
    {
        _messages.OnNext(new PlaceMessage(TopLeft));

        Assert.AreEqual(O, _player);
    }
}
```

Then add new test and implementation… until all tests done and game is complete



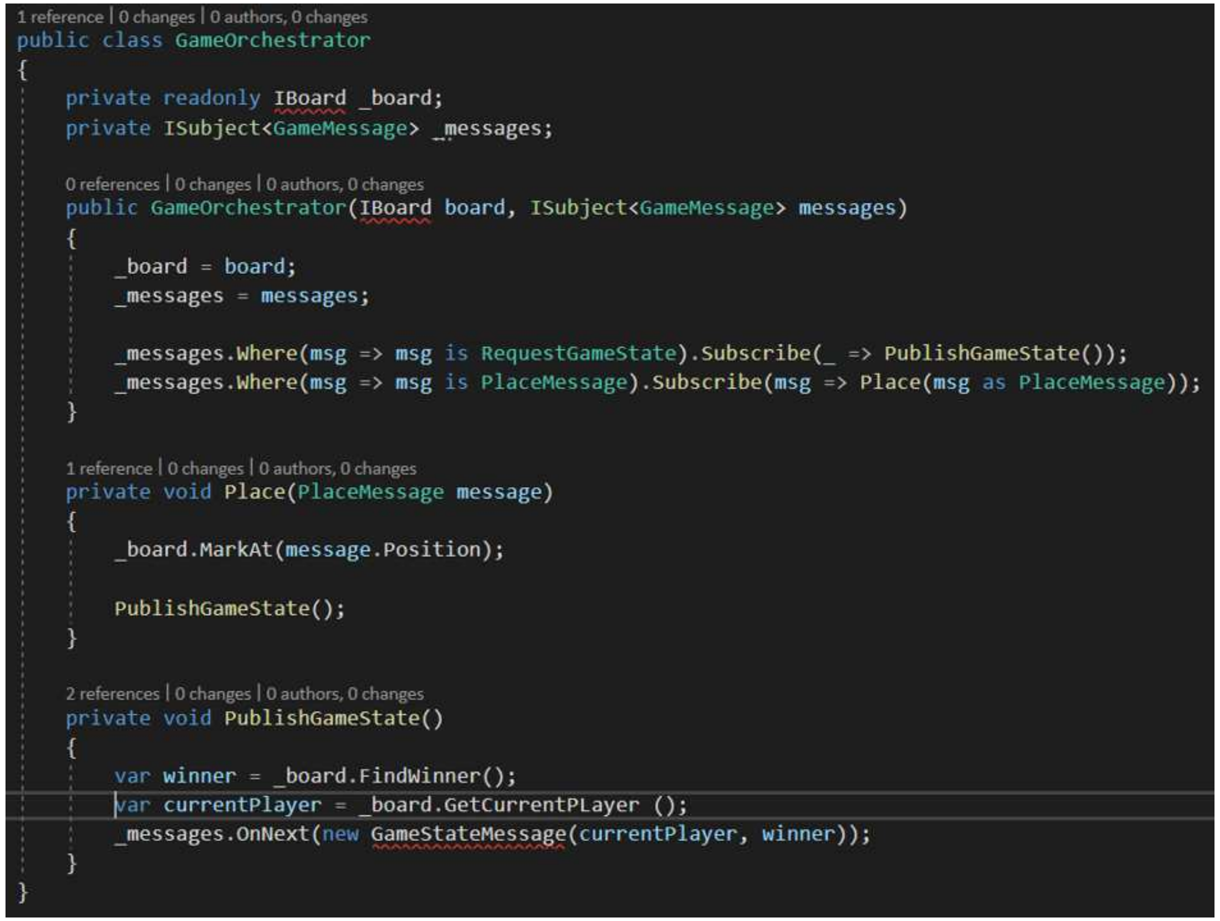
```
[Test]
| 0 references | Henning Torsteinsen, 7 minutes ago | 1 author, 1 change
public void NotHaveAWinnerAtStart()
{
    _messages.OnNext(new RequestGameState()) ;

    Assert.AreEqual(None, _winner);
}

[Test]
| 0 references | Henning Torsteinsen, 7 minutes ago | 1 author, 1 change
public void MakeXWinWhenTopRowContainsAllXs()
{
    _messages.OnNext(new PlaceMessage(TopLeft));
    _messages.OnNext(new PlaceMessage(BottomLeft));
    _messages.OnNext(new PlaceMessage(TopMiddle));
    _messages.OnNext(new PlaceMessage(BottomMiddle));
    _messages.OnNext(new PlaceMessage(TopRight));

    Assert.AreEqual(X, _winner);
}

[Test]
| 0 references | Henning Torsteinsen, 7 minutes ago | 1 author, 1 change
public void MakeOWinWhenTopRowContainsAllOs()
{
    _messages.OnNext(new PlaceMessage(MiddleMiddle));
    _messages.OnNext(new PlaceMessage(TopLeft));
    _messages.OnNext(new PlaceMessage(BottomLeft));
    _messages.OnNext(new PlaceMessage(TopMiddle));
    _messages.OnNext(new PlaceMessage(BottomMiddle));
    _messages.OnNext(new PlaceMessage(TopRight));

    Assert.AreEqual(O, _winner);
}

[Test]
| 0 references | Henning Torsteinsen, 7 minutes ago | 1 author, 1 change
public void MakeXWinWhenMiddleRowContainsAllXs()
{
    _messages.OnNext(new PlaceMessage(MiddleMiddle));
    _messages.OnNext(new PlaceMessage(TopLeft));
    _messages.OnNext(new PlaceMessage(MiddleLeft));
    _messages.OnNext(new PlaceMessage(TopMiddle));
    _messages.OnNext(new PlaceMessage(MiddleRight));

    Assert.AreEqual(X, _winner);
}

[Test]
| 0 references | Henning Torsteinsen, 7 minutes ago | 1 author, 1 change
public void MakeNoWinnerWhenBoardFull()
```

<Demonstrate refactor here…>



```
2 references | Henning Torsteinsen, 6 minutes ago | 1 author, 1 change
public class TicTacToeReactive
{
    private readonly Board _board = new Board();
    public Subject<GameMessage> Messages = new Subject<GameMessage>();
    private Player _currentPlayer = X;

    1 reference | Henning Torsteinsen, 6 minutes ago | 1 author, 1 change
    public TicTacToeReactive()
    {
        Messages.Where(msg => msg is RequestGameState).Subscribe(PublishGameState);
        Messages.Where(msg => msg is PlaceMessage).Subscribe(msg => Place(msg as PlaceMessage));
    }

    1 reference | Henning Torsteinsen, 6 minutes ago | 1 author, 1 change
    private void Place(PlaceMessage message)
    {
        if (_board.IsTaken(message.Position))
        {
            return;
        }

        _board.MarkAt(message.Position, _currentPlayer);

        AlternatePLayer();

        PublishGameState(null);
    }

    1 reference | Henning Torsteinsen, 6 minutes ago | 1 author, 1 change
    private void AlternatePLayer()
    {
        if (_currentPlayer == O)
        {
            _currentPlayer = X;
            return;
        }

        _currentPlayer = O;
    }

    2 references | 0 changes | 0 authors, 0 changes
    private void PublishGameState(GameMessage obj)
    {
        var winner = _board.FindWinner();
        Messages.OnNext(new WinnerMessage(winner));

        Messages.OnNext(new CurrentPlayerMessage(_currentPlayer));

    }
```

Refactor result



```
1 reference | 0 changes | 0 authors, 0 changes
public class GameOrchestrator
{
    private readonly IBoard _board;
    private ISubject<GameMessage> _messages;

    0 references | 0 changes | 0 authors, 0 changes
    public GameOrchestrator(IBoard board, ISubject<GameMessage> messages)
    {
        _board = board;
        _messages = messages;

        _messages.Where(msg => msg is RequestGameState).Subscribe(_ => PublishGameState());
        _messages.Where(msg => msg is PlaceMessage).Subscribe(msg => Place(msg as PlaceMessage));
    }

    1 reference | 0 changes | 0 authors, 0 changes
    private void Place(PlaceMessage message)
    {
        _board.MarkAt(message.Position);

        PublishGameState();
    }

    2 references | 0 changes | 0 authors, 0 changes
    private void PublishGameState()
    {
        var winner = _board.FindWinner();
        var currentPlayer = _board.GetCurrentPLayer ();
        _messages.OnNext(new GameStateMessage(currentPlayer, winner));
    }
}
```

What did I learn?

- Many responsibilities to single responsibility
- Subscriptions reads as a table of contents for the class
- Can make orchestrator generic, using interfaces for each game type.

<End of presentation>

THANK YOU FOR YOUR TIME!

LET’S KEEP IN TOUCH:
GITHUB.COM/HENNINGNT
WWW.LINKEDIN.COM/IN/HENNINGNT/