Testing

This document aims to describe the work done during week six, which consisted in some competitive testing. This involved completing the methods of an incomplete program, testing them, and then getting them cross-tested by other group's tests and viceversa.

Introduction

Before getting into the work done it is quite important to highlight that these tests were written using **pair programming** with one of my classmates. First, one of us was writing the code while the other one was thinking about improvements and reviewing the written lines, then we would swap roles and keep doing it until the work was completed.
We managed to write a total of 5 tests, 2 of which use the `InlineData` attribute to run the tests with multiple parameters sets. As required by the portfolio's instructions, only two of those five are shown in the next sections.

Test 1

The code's purpose

The first method to test was the `CreateNewChallenge()` method. This is called when the user selects one of the three difficulty levels that you can see below, as well as the method's implementation:

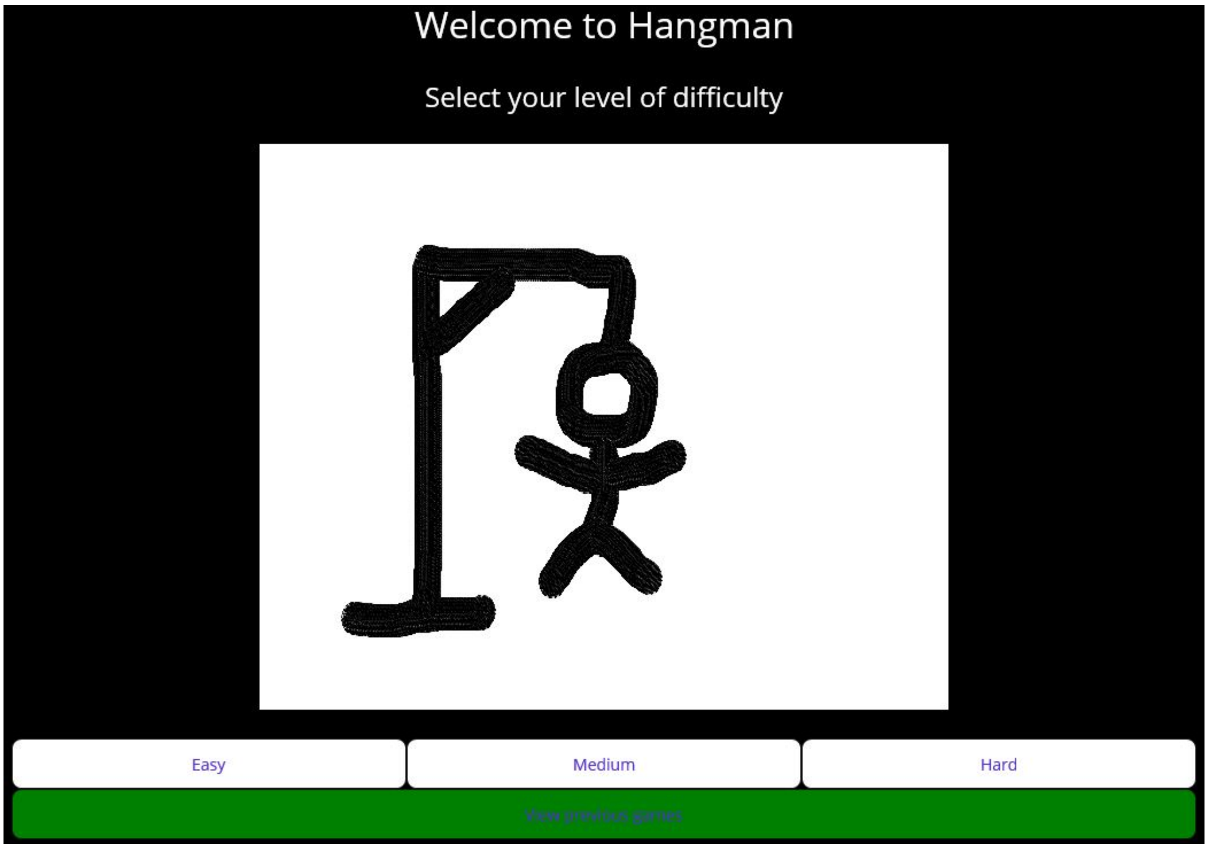


Fig.1 - Difficulty levels

```
private void CreateNewChallenge()
{
    Word = SelectWord(GameType);
    ResetDisplay(Word);
}
```

The first thing this method does is to call another method called `SelectWord()`, its function is to pick a word from a list of words which will be the one that the user is meant to guess, and it is assigned to a variable called `Word`. After that, one last method is called, `ResetDisplay()`, which resets the display to the initial image plus the appropriate number of visible labels.

My test implementation

We created a total of three tests, one for each difficulty, and here follows one of their implementations:

```
public void MediumGameIsCreated()
{
    GamePage game = new GamePage("Medium");

    game.CreateNewChallenge();
    Assert.True(game.Word.Length >= 7 & game.Word.Length < 10);
}
```

As you can deduce from the method's name, what this test does is to make sure that a game of medium difficulty is created when we input the string `"Medium"` to the constructor of the GamePage object. To do so we create a GamePage object passing `"Medium"` as parameter, run the `CreateNewChallenge()` method and check whether the word that was generated satisfies the word length criteria that we were given.
It is really important to test this because it helps us to find out if the word generation is working fine or not. A malfunctioning word generator would ruin the purpose of the user interface shown in figure 1, for example if it was taking only words that are shorter than 7 characters (easy words), or if it was taking random words without any criteria. Both of these cases imply that the choice of game difficulty would be completely vane.

Limitations

Being a relatively simple test method, it might feel safe to say that `MediumGameIsCreated()` does not have any limitations, in fact it doesn't, but our test class does. If you look closely, the `CreateNewChallenge()` metohd does not only generate a word, which is the only feature we tested as mentioned just before the code snippet, but it also resets the display. For the tests to be complete it would be essential to check, for example, whether the image was updated successfully, or whether the placeholders for the word to guess were displayed, etc.

Test 2

The code's purpose

The second method to test was the `OnAttemptSubmitted()` method. This is called after the user has clicked on the "Go" button, possibly after having written something in the text box. Below the UI and the method's implementation are shown:

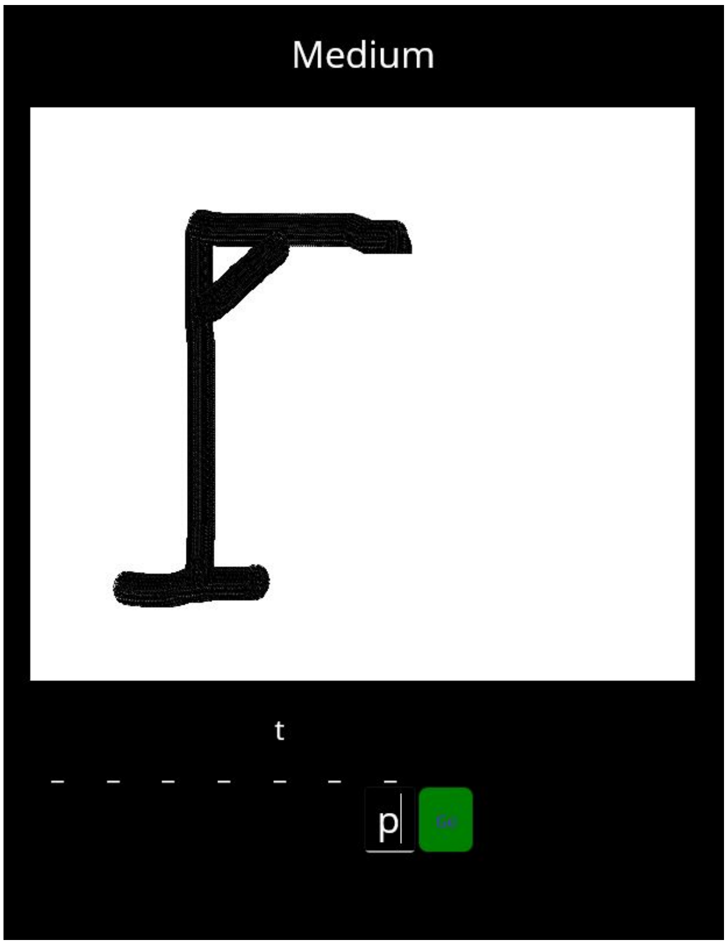


Fig.2 - Submit letter

```
public void OnAttemptSubmitted(object sender, EventArgs e)
{
    char answer = userInputProvider.GetUserInput();

    if (EntryIsValid(answer))
    {
        bool isCorrect = CheckLetterInWord(Word, answer);

        if (!isCorrect)
            remainingAttempts--;
        else
            UserAnswer.Add(answer);

        UpdateDisplay(isCorrect, Word, answer, remainingAttempts);
    }

    ClearEntry();
```

```
        CheckGameOver();
    }
```

The first thing this method does it to assign the first character of the answer that the user has written to a variable called `answer`. If this answer is an invalid value the if statement is skipped, otherwise we set another variable called `isCorrect` by using `CheckLetterInWord()`, a method which checks whether the letter appears or not in the word. The `remainingAttempts` variable and the guessed letters are updated accordingly, and then `UpdateDisplay()` is called, which changes the image shown on the page and updates the visibility of the labels representing the letters in the word. Finally `CheckGameOver()` checks whether `remainingAttemps` has reached 0 or the user has guessed all the letters, in either case `GameOver()` is called, its function is to reset all game variables and display the final result.

My test implementation

We wrote a total of two tests which check that the remaining attempts decrease when hitting "Go" and that invalid inputs are ignored, here follows the implementation of the second one:

```
    [Theory]
    [InlineData(null, 7)]
    [InlineData('=', 7)]
    public void OnAttemptSubmitted_InvalidInput_IsIgnored(char expectedInput, int
    expectedRemainingAttempts)
    {
        mockInputProvider.Setup(x => x.GetUserInput()).Returns(expectedInput);

        var gamePage = new GamePage("Easy");
        gamePage.userInputProvider = mockInputProvider.Object;

        gamePage.Word = "apple";

        gamePage.OnAttemptSubmitted(this, new EventArgs());

        Assert.Equal(expectedRemainingAttempts, gamePage.remainingAttempts);
    }
```

This test uses the Mock class to simulate a user input, which is `null` first, and then an equal sign. After that, a game mode is set, and a word too (Apple in this case). The attempt is submitted and then an assertion checks that the number of remaining attemps is equal to 7, basically they should not have decreased as `null` should have been ignored. The same thing happens with `"="`, the remaining attempts shoul remain the same.
It is important to test this because what we do not want in a game is for it to crash. If our game was not able to handle invalid data, like symbols, empty spaces or null, it means that whenever those values are encountered our program would throw an error and crash. While if we handle these values correctly, the user does not have to be worried about submitting wrong data by accident, the attempt will simply be ignored.

Limitations

Once again the test method is very concise and only tests one thing, therefore it has almost no limitations. Why almost? For example we could say that not testing some other special character would be a limitation,

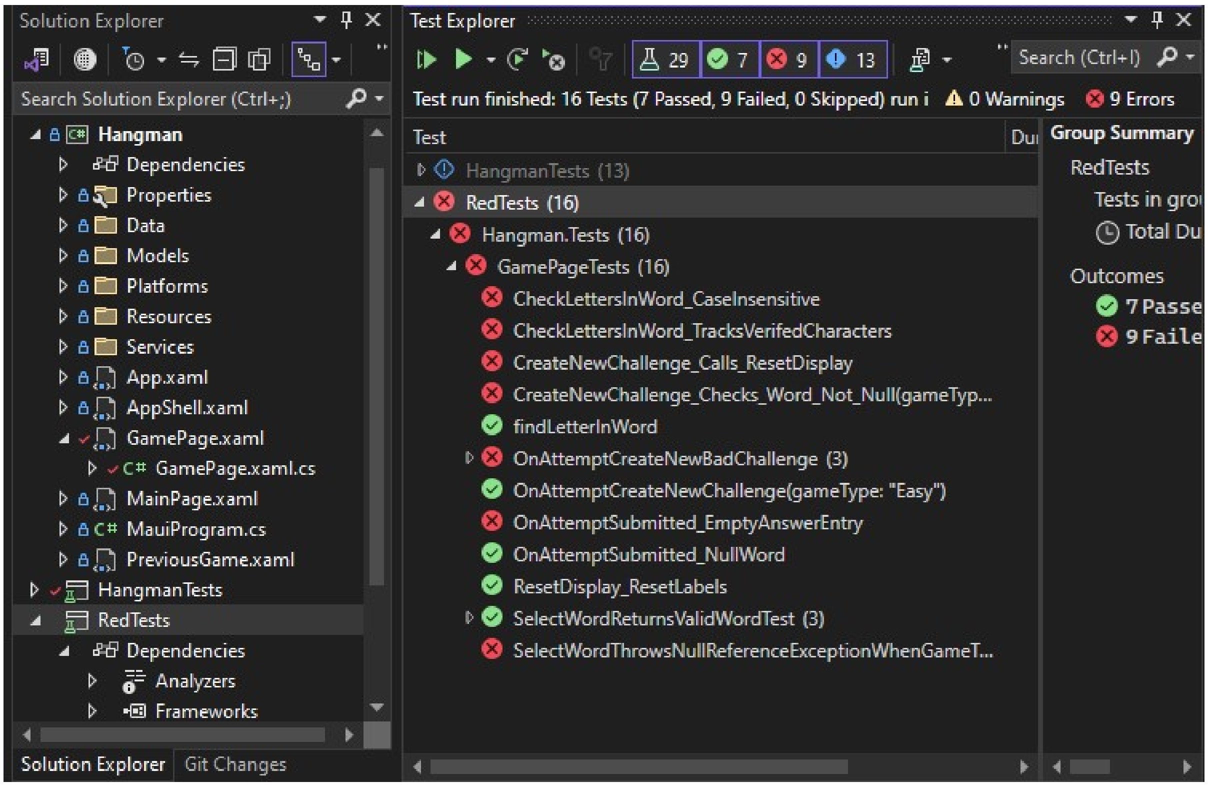
but personally I would not consider it as such as you should not test every possible character. But this time the metohd to test was a bit longer and was using a fair amount of functionalities, consequently our test class should have covered more test cases.
It is possible to see that a method that updates the display and one that clears the input field are called. Therefore to consider our test class as complete we should check, for example, that the image stays the same when no error is made and that it changes when a mistake is made or check that the input field is cleared after a submission etc.

Final evaluation

After the test battle we got to be on the podium, at third place, but out of four teams, so not the best result we could get. Below we can see how our code performed on other team's tests and how our tests performed on other team's code.

Our Code

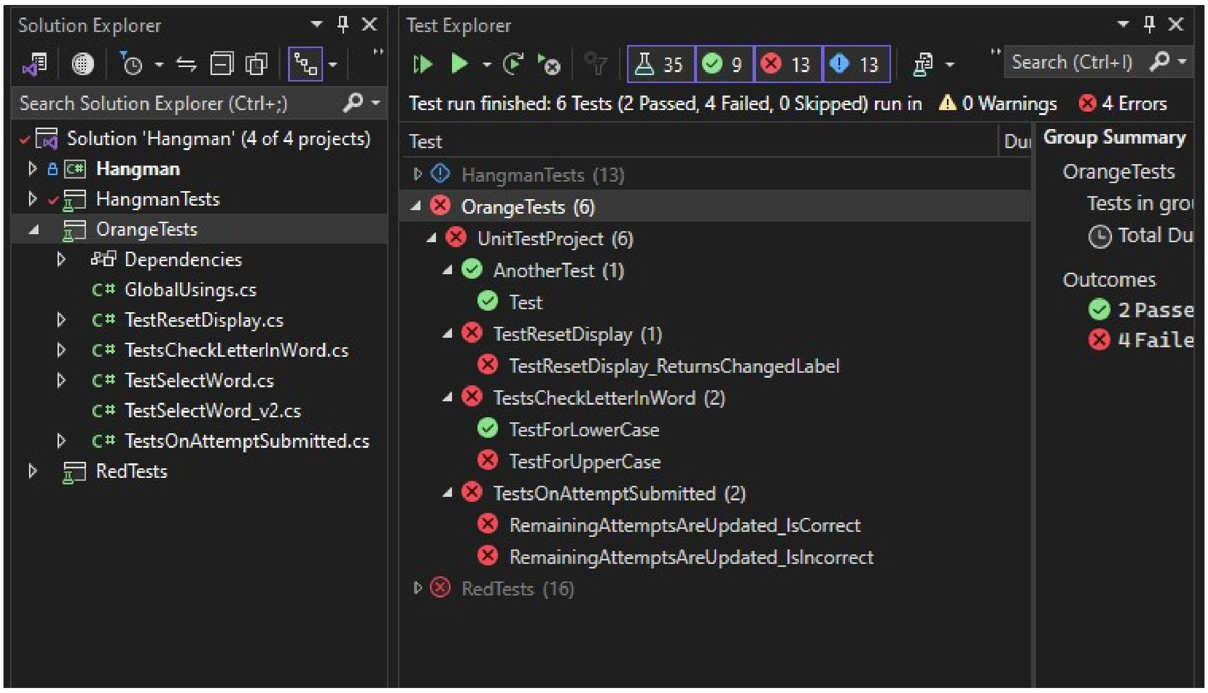
Starting we have the tests of the Red team (figure 3), from a glance at it I can say that most of these tests failed because the implementation of the Red team was different from ours, for example the case insensitive test writes the word `Word` and tries to find the `W` but our code simply saves it all lowercase so there is no capital w to find. We can also see that they also managed empty answers and exceptions differently, but despite these differences I like the idea of a case sensitive test and we probably should have written one too.


```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Hangman
    Dependencies
    Properties
    Data
    Models
    Platforms
    Resources
    Services
    App.xaml
    AppShell.xaml
    GamePage.xaml
        GamePage.xaml.cs
    MainPage.xaml
    MauiProgram.cs
    PreviousGame.xaml
HangmanTests
RedTests
    Dependencies
        Analyzers
        Frameworks
Solution Explorer   Git Changes
```

```
Test Explorer
29   7   9   13                Search (Ctrl+I)
Test run finished: 16 Tests (7 Passed, 9 Failed, 0 Skipped) run i   0 Warnings   9 Errors
Test                                                            Dui   Group Summary
HangmanTests (13)                                                      RedTests
RedTests (16)                                                            Tests in grou
  Hangman.Tests (16)                                                     Total Du
    GamePageTests (16)                                                 Outcomes
      CheckLettersInWord_CaseInsensitive                                 7 Passe
      CheckLettersInWord_TracksVerifedCharacters                         9 Faile
      CreateNewChallenge_Calls_ResetDisplay
      CreateNewChallenge_Checks_Word_Not_Null(gameTyp...
      findLetterInWord
      OnAttemptCreateNewBadChallenge (3)
      OnAttemptCreateNewChallenge(gameType: "Easy")
      OnAttemptSubmitted_EmptyAnswerEntry
      OnAttemptSubmitted_NullWord
      ResetDisplay_ResetLabels
      SelectWordReturnsValidWordTest (3)
      SelectWordThrowsNullReferenceExceptionWhenGameT...
```

Fig.3 - Red team tests

A test which was really similar to the one mentioned above was written by the Orange team too and, consequently, it failed again. We can also see another test that checks whether the remaining attempts update correctly and it fails despite our code working perfectly, so it is again a case of different code implementation from the teams. You can see the tests below:

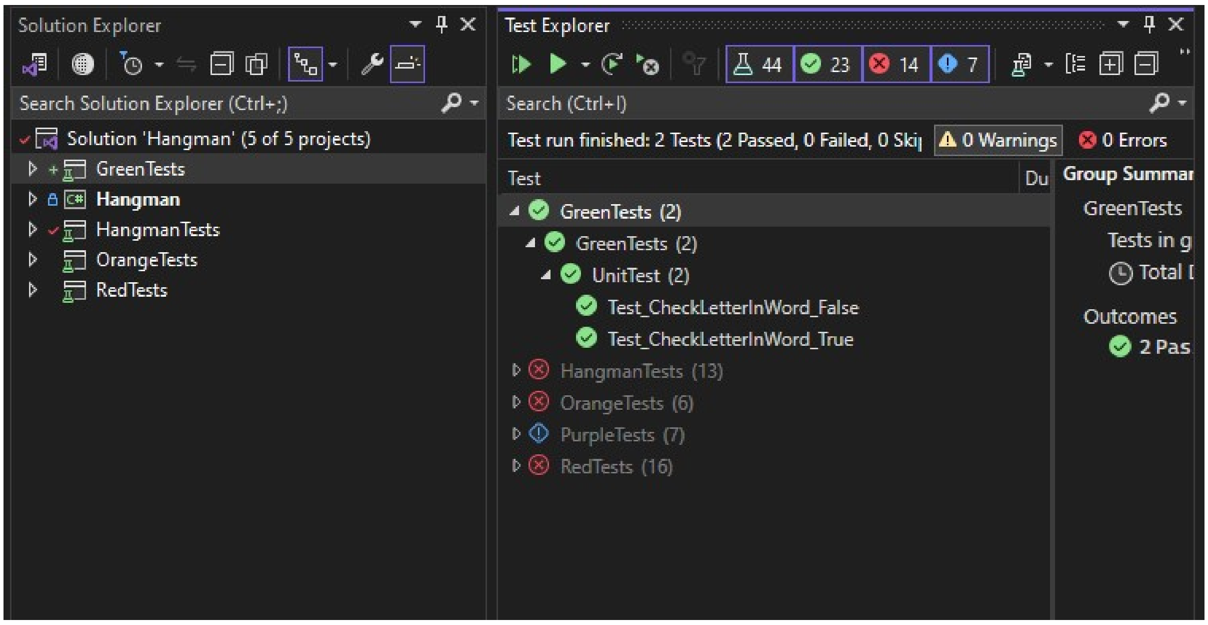


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Hangman' (4 of 4 projects)
Hangman
HangmanTests
OrangeTests
Dependencies
GlobalUsings.cs
TestResetDisplay.cs
TestsCheckLetterInWord.cs
TestSelectWord.cs
TestSelectWord_v2.cs
TestsOnAttemptSubmitted.cs
RedTests

Test Explorer
35 9 13 13
Search (Ctrl+I)
Test run finished: 6 Tests (2 Passed, 4 Failed, 0 Skipped) run in 0 Warnings 4 Errors
Test Dur Group Summary
HangmanTests (13)
OrangeTests (6)
UnitTestProject (6)
AnotherTest (1)
Test
TestResetDisplay (1)
TestResetDisplay_ReturnsChangedLabel
TestsCheckLetterInWord (2)
TestForLowerCase
TestForUpperCase
TestsOnAttemptSubmitted (2)
RemainingAttemptsAreUpdated_IsCorrect
RemainingAttemptsAreUpdated_IsIncorrect
RedTests (16)
OrangeTests
Tests in gro
Total Du
Outcomes
2 Passe
4 Faile

Fig.4 - Orange team tests

Finally the tests of the last team, Green team, were only two and really simple so we managed to pass both.



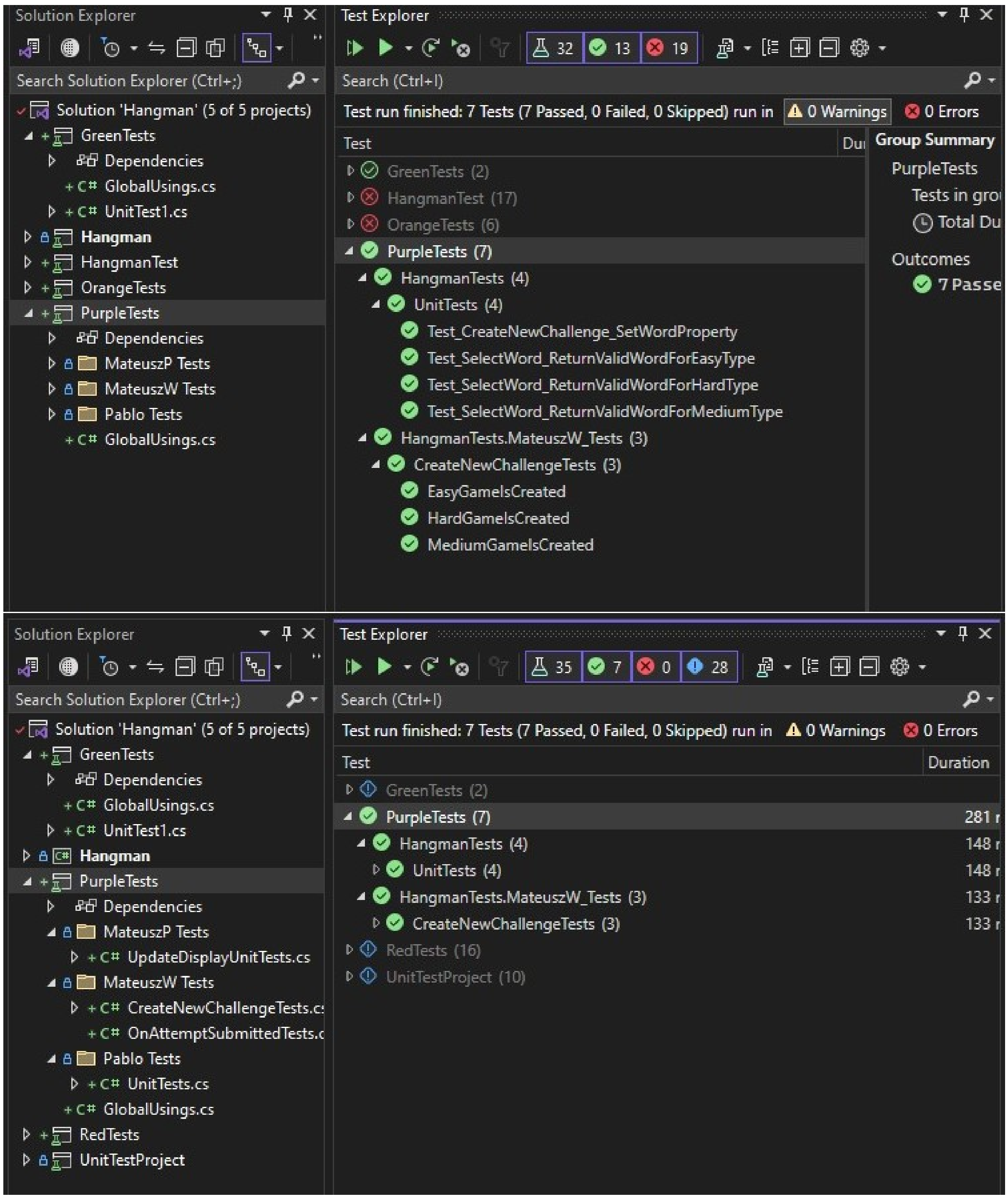
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Hangman' (5 of 5 projects)
GreenTests
Hangman
HangmanTests
OrangeTests
RedTests

Test Explorer
44 23 14 7
Search (Ctrl+I)
Test run finished: 2 Tests (2 Passed, 0 Failed, 0 Ski 0 Warnings 0 Errors
Test Du Group Summar
GreenTests (2)
GreenTests (2)
UnitTest (2)
Test_CheckLetterInWord_False
Test_CheckLetterInWord_True
HangmanTests (13)
OrangeTests (6)
PurpleTests (7)
RedTests (16)
GreenTests
Tests in g
Total I
Outcomes
2 Pas

Fig.5 - Green team tests

Our tests

As it is possible to see below, our tests were used on the code of Red and Orange team and the outcome was successful, not even one test failed, which means that both team worked on enough features for all our tests to pass:

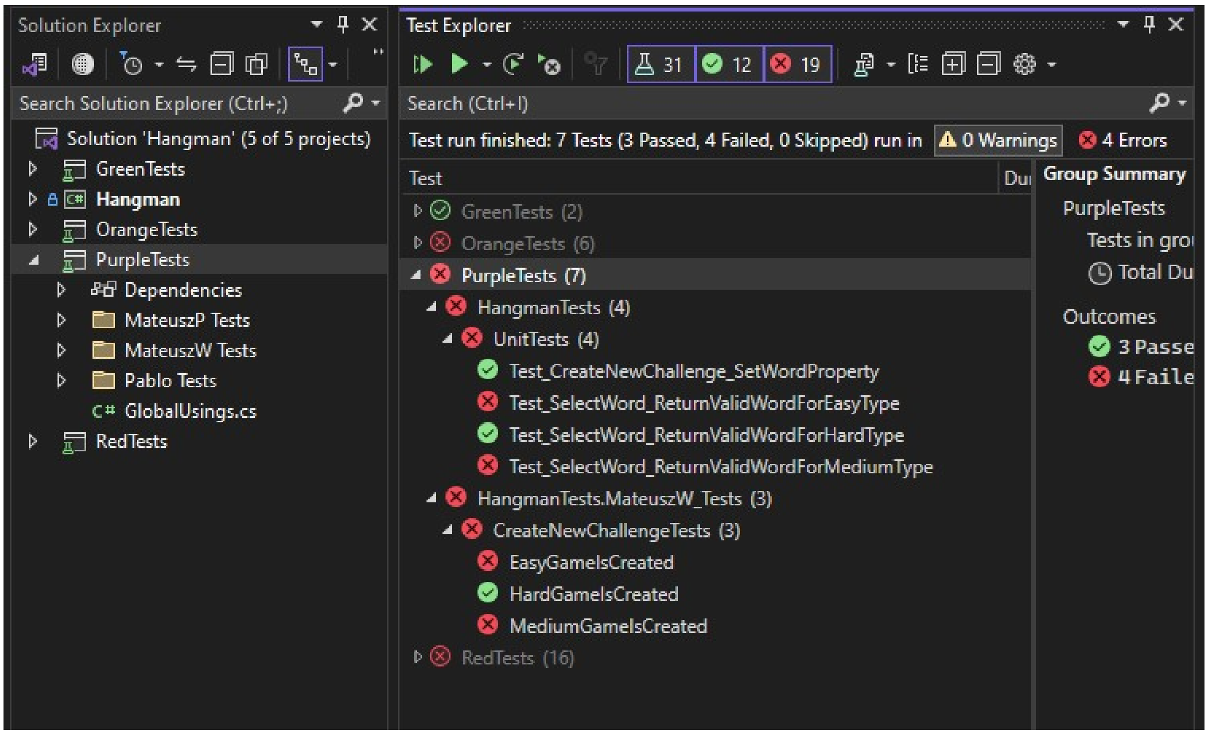


```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Hangman' (5 of 5 projects)
  GreenTests
    Dependencies
    GlobalUsings.cs
    UnitTest1.cs
  Hangman
  HangmanTest
  OrangeTests
  PurpleTests
    Dependencies
    MateuszP Tests
    MateuszW Tests
    Pablo Tests
    GlobalUsings.cs

Test Explorer
32  13  19
Search (Ctrl+I)
Test run finished: 7 Tests (7 Passed, 0 Failed, 0 Skipped) run in   0 Warnings   0 Errors
Test                                                    Dur   Group Summary
GreenTests (2)                                                PurpleTests
HangmanTest (17)                                                Tests in gro
OrangeTests (6)                                                 Total Du
PurpleTests (7)                                               Outcomes
  HangmanTests (4)                                              7 Passe
    UnitTests (4)
      Test_CreateNewChallenge_SetWordProperty
      Test_SelectWord_ReturnValidWordForEasyType
      Test_SelectWord_ReturnValidWordForHardType
      Test_SelectWord_ReturnValidWordForMediumType
  HangmanTests.MateuszW_Tests (3)
    CreateNewChallengeTests (3)
      EasyGameIsCreated
      HardGameIsCreated
      MediumGameIsCreated
```

```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Hangman' (5 of 5 projects)
  GreenTests
    Dependencies
    GlobalUsings.cs
    UnitTest1.cs
  Hangman
  PurpleTests
    Dependencies
    MateuszP Tests
      UpdateDisplayUnitTests.cs
    MateuszW Tests
      CreateNewChallengeTests.c
      OnAttemptSubmittedTests.c
    Pablo Tests
      UnitTests.cs
    GlobalUsings.cs
  RedTests
  UnitTestProject

Test Explorer
35  7  0  28
Search (Ctrl+I)
Test run finished: 7 Tests (7 Passed, 0 Failed, 0 Skipped) run in   0 Warnings   0 Errors
Test                                                    Duration
GreenTests (2)
PurpleTests (7)                                         281 r
  HangmanTests (4)                                      148 r
    UnitTests (4)                                       148 r
  HangmanTests.MateuszW_Tests (3)                       133 r
    CreateNewChallengeTests (3)                         133 r
RedTests (16)
UnitTestProject (10)
```

Fig.6 - Our test on Red and Orange team

Not the same can be said about the Green team, which clearly had not implemented enough features to get our tests to run properly, therefore almost all of them failed except for the three that you can see in the image below


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'Hangman' (5 of 5 projects)
- GreenTests
- Hangman
- OrangeTests
- PurpleTests
  - Dependencies
  - MateuszP Tests
  - MateuszW Tests
  - Pablo Tests
  - GlobalUsings.cs
- RedTests

Test Explorer

31 12 19

Search (Ctrl+I)

Test run finished: 7 Tests (3 Passed, 4 Failed, 0 Skipped) run in 0 Warnings 4 Errors

```
Test                                              Du
✓ GreenTests (2)
✗ OrangeTests (6)
✗ PurpleTests (7)
  ✗ HangmanTests (4)
    ✗ UnitTests (4)
      ✓ Test_CreateNewChallenge_SetWordProperty
      ✗ Test_SelectWord_ReturnValidWordForEasyType
      ✓ Test_SelectWord_ReturnValidWordForHardType
      ✗ Test_SelectWord_ReturnValidWordForMediumType
  ✗ HangmanTests.MateuszW_Tests (3)
    ✗ CreateNewChallengeTests (3)
      ✗ EasyGameIsCreated
      ✓ HardGameIsCreated
      ✗ MediumGameIsCreated
✗ RedTests (16)
```

Group Summary

PurpleTests

Tests in gro

Total Du

Outcomes

✓ 3 Passe

✗ 4 Faile

Fig.7 - Our tests on Green team