CMP1902M Object Oriented Programming 2023/24

Assignment 2: Report

[*Expand the sections as necessary*]

Name: Callum Tomkins

Student ID: 28070874

Code repository URL: https://github.com/CallumCPP/OOP2

Video URL:

Application:

1. **Reflection on the OO features within your code.** (~400 words)
   Page 4

2. **Reflection on your handling of error conditions in your code.** (~200 words)
   Page 5

3. **Reflection on your testing activities: What did you test, and how did you do it?** (~400 words)
   Eg: I tested the application against ….

   Page 6

4. **Include evidence of the tests *(screenshots are OK)***

   Page 7

Reflection and Feedback

5. **What was the most important thing you learned from this assessment?** *(< 200 words)*
   Eg: I learned that If you don't think every day is a good day - try missing a few. You'll see.

   Page 8

6. **What was the most challenging aspect of this assessment and how did you approach it?** *(<20*
   words)
   Eg: I started painting as a hobby when I was little. I didn't know I had any talent. I believe talent is just a pursued interest.

   Page 8

7. **What would you particularly like to receive feedback on in this assessment?**

   Page 8

Assignment 2 Checklist

All of the elements in a section must be checked for it to be considered for that grade (this isn't guaranteed though). All previous elements must also be complete for a grade to be considered.

Pass standard:

The code compiles and runs.	Y
Die, Game and Testing classes are created.	Y
Object instantiation, method calls evident.	Y
Sevens Out game is created.	Y
The Testing class is used.	Y

2:2 standard:

The rules of the Sevens Out game, as specified, are implemented.	Y
Application repeats or quits the game gracefully according to user choice.	Y
Method calls from 'Main' to methods in other classes	Y
Error handling is evident, some errors are captured, such as erroneous input being made.	Y
Class definitions show **encapsulation.**	Y
The Testing class checks the dice sum is correct and that a total of 7 is detected.	Y
A Statistics class is used	Y

2:1 standard:

Sevens Out and Three Or More games are implemented.	Y
Inheritance is implemented, showing a class hierarchy	Y
`public/private` access control in classes	Y
Generic collections (such as List<>) are used.	Y
Exception handling is used	Y
Testing class uses verification methods in code (such as `debug.assert`()) to check code.	Y

First standard:

Interfaces and LINQ are used	Y
Static and/or Dynamic **polymorphism** are evident	Y
Use of `virtual/abstract` methods	Y
`protected` access control is used in class hierarchy	Y
The Testing class implements a way to record testing data (through a log file for example)	Y

1. Reflection on the object oriented features of my code

To demonstrate use of object oriented programming paradigm I have used multiple techniques including:
Objects:

- Die
- Game (abstract)
  - SevensOut : Game
  - ThreeOrMore : Game
- GameStats
  - Player
- Input (static)
- Program (static)
- Statistics (static)
- Testing (static)
  - ITestData (interface)
    - SevensOutData : ITestData
    - ThreeOrMoreData : ITestData

The SevensOut and ThreeOrMore classes inherit from the abstract class Game
These classes then override the abstract method _play() (which is called by the public method Play())
This demostrates inheritance and polymorphism.
The use of these techniques improve my code by allowing me to abstract more code and therefore have much easier to read and understand code with less duplicate code.

An example of this abstraction can be shown within the Game class, the public method Play() calls the protected _play() method. This method also determines wether to play multiplayer and gets a name to save the score, reducing duplicate code by preventing the need for it in the subsequent classes.
By applying the access modifier protected to this method, it is only accessible by itself and its children.
Access modifiers have aided in my programming by allowing me to define where variables should and should not be accessed from. This helps by preventing accidental setting of fields.

An example of inheritance and interfaces can be found with the ITestData interface and its children.
The children classes, SevensOutData and ThreeOrMoreData, inherit the definition of the method GetName(), they then impliment this method seperately to return the correct name.
Currently this use of interfaces gives little benefit currently however if I were to continue to develop this program structuring my code this way may help me.

The use of object oriented programming over procedural programming, in my project, was very beneficial to my code and code style, it helped me produce clean, easily understood, code. Rather than having many variable definitions and functions to produce the same effects.

Encapsulation was using in my code in many places but one example is the Game class.
This class contains 4 fields and 3 methods with varying accessibilty.
The public Play() method executes code which manipulates the protected fields and calls the protected method _play() to further maipulate the data stored by the class.

2. How I’ve handled error conditions in my code

An example of error handling is found in the Input class.
Whenever accepting a string from the console there are often many opportunities for errors, every invalid case must be handled.
For example, in the GetInt(string message, int min, int max) method multiple checks must be in place.

The first check in the while loop, the null check, ensures that Console.ReadLine() does not return null. This can happen if the user presses the key combination ctrl+z.

The second check, wether the string was parsed as an integer, is to ensure that the user has entered a valid integer.

The final two checks are to enure the value entered is within the specified range, the min and max parameters have a default value that is int.MinValue and int.MaxValue respectively.

All of these checks work together to ensure no erroneous input is possible.

Another example is in the Statistics class.
In the method Load(), the program first checks that the files exist before attempting to load and deserialize the stored json. If the file does not exist it will create it and populate it with an empty instance of the GameStats class.

If this was not done then when the program is loaded for the first time an error will occur, preventing the statistic files from ever being created.

A third example is in the GameStats class, in the AddEntry(string name, int score, bool tie) method.
If no scores have been set with the specified name the linq query witll return an empty enumerable, the .SingleOrDefault() method called on this enumerable returns null. If there was a score with that name set before then the method will return the instance of Player with that name.
This null case is handled by creating a new instance of Player and adding it to the Players list.

3. A reflection on my testing activities

To test the sevens out game I ensured rolls are added correctly using Debug.Assert().
The program loops over all of the rolls in the game and ensures each total was calculated correctly.
The condition for the Debug.Assert was roll.roll1 + roll.roll2 == roll.total.

To get the testing data out of the SevensOut class I created a class named SevensOutData that inherits from the ITestData interface.
I then passed an instance of SevensOutData to the optional constructor parameter testingData, by filling this parameter tesing is enabled.
The data is then collected throughout the automated play of the game and checked afterwards.

I decided to collect and test after the game has completed as testing during play would've either involved some back and forth between the static Testing class and SevensOut instance or testing code defined directly in the SevensOut class.

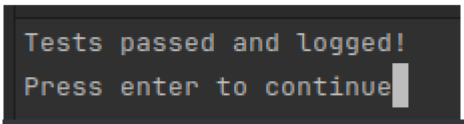
The final Sevens Out check is that the final score was 7.
The condition for this Assert was data.Rolls.Last().total == 7.

To test the Three Or More game I first ensure that the winners penultimate score is less than 20 and their final score is greater than 20.
This is done to ensure that the game ends when a player reaches 20.

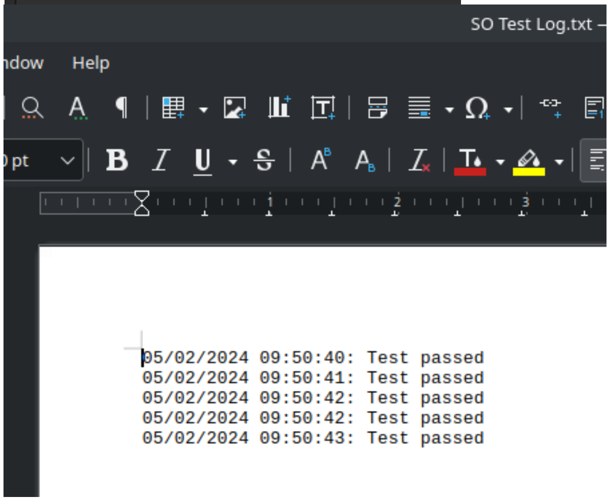
Secondly I test to make sure that the right score is added depending on the number of same die.
This is done by a switch statement switching the number of same die.
In the case that it is 3, assert that the change in score was 3.
In the case that it is 4, assert that the change in score was 6.
In the case that it is 5, assert that the change in score was 12.

This testing helped to ensure my code worked as intended throughout the development process.

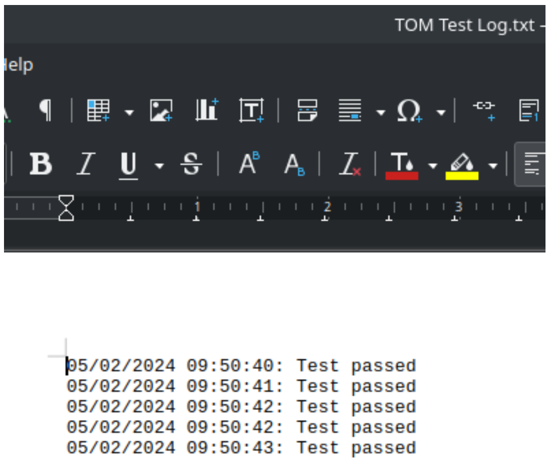
Test Evidence



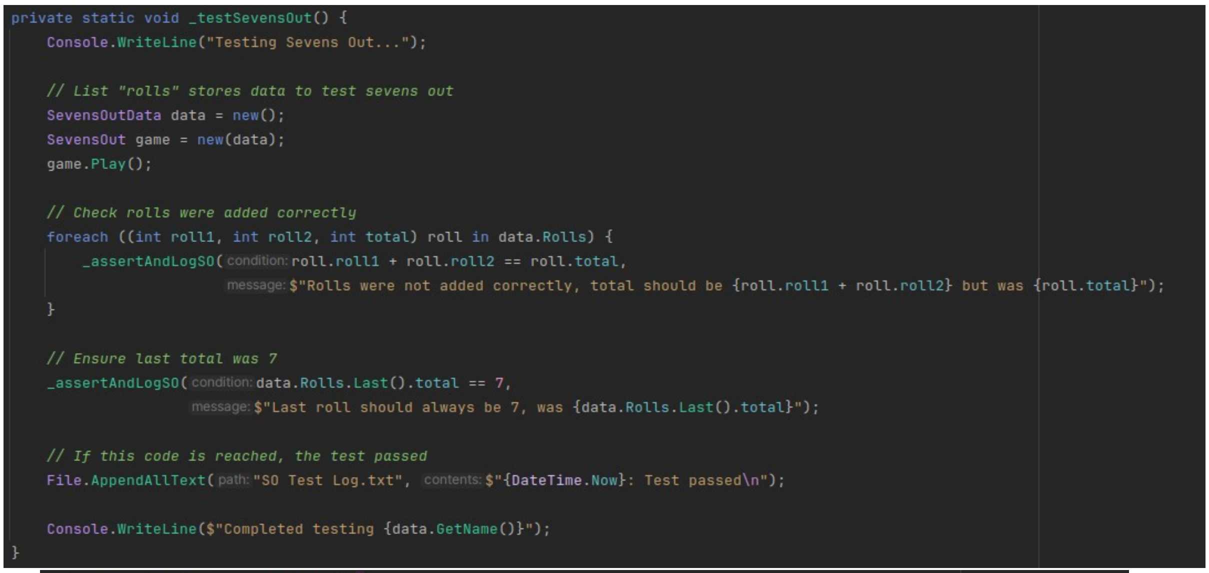
```
Tests passed and logged!
Press enter to continue
```



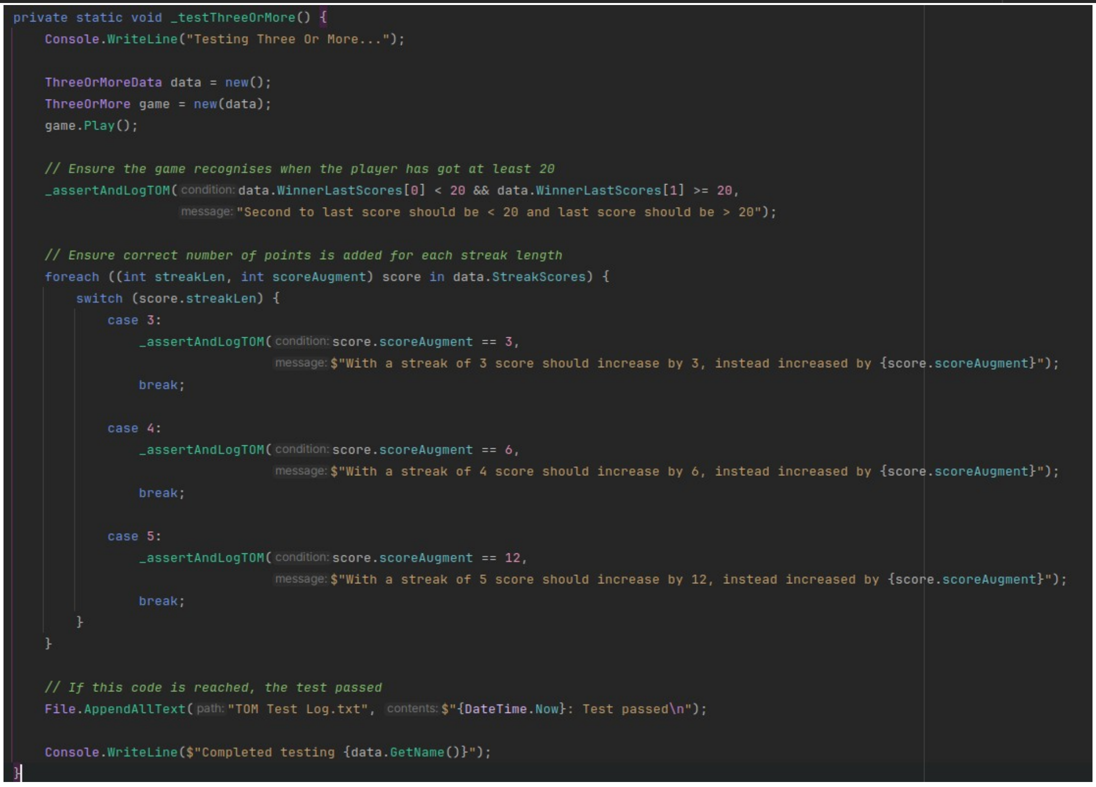
```
05/02/2024 09:50:40: Test passed
05/02/2024 09:50:41: Test passed
05/02/2024 09:50:42: Test passed
05/02/2024 09:50:42: Test passed
05/02/2024 09:50:43: Test passed
```



```
05/02/2024 09:50:40: Test passed
05/02/2024 09:50:41: Test passed
05/02/2024 09:50:42: Test passed
05/02/2024 09:50:42: Test passed
05/02/2024 09:50:43: Test passed
```



```
private static void _testSevensOut() {
    Console.WriteLine("Testing Sevens Out...");

    // List "rolls" stores data to test sevens out
    SevensOutData data = new();
    SevensOut game = new(data);
    game.Play();

    // Check rolls were added correctly
    foreach ((int roll1, int roll2, int total) roll in data.Rolls) {
        _assertAndLogSO( condition: roll.roll1 + roll.roll2 == roll.total,
                         message: $"Rolls were not added correctly, total should be {roll.roll1 + roll.roll2} but was {roll.total}");
    }

    // Ensure last total was 7
    _assertAndLogSO( condition: data.Rolls.Last().total == 7,
                     message: $"Last roll should always be 7, was {data.Rolls.Last().total}");

    // If this code is reached, the test passed
    File.AppendAllText( path: "SO Test Log.txt",  contents: $"{DateTime.Now}: Test passed\n");

    Console.WriteLine($"Completed testing {data.GetName()}");
}
```



```
private static void _testThreeOrMore() {
    Console.WriteLine("Testing Three Or More...");

    ThreeOrMoreData data = new();
    ThreeOrMore game = new(data);
    game.Play();

    // Ensure the game recognises when the player has got at least 20
    _assertAndLogTOM( condition: data.WinnerLastScores[0] < 20 && data.WinnerLastScores[1] >= 20,
                      message: "Second to last score should be < 20 and last score should be > 20");

    // Ensure correct number of points is added for each streak length
    foreach ((int streakLen, int scoreAugment) score in data.StreakScores) {
        switch (score.streakLen) {
            case 3:
                _assertAndLogTOM( condition: score.scoreAugment == 3,
                                  message: $"With a streak of 3 score should increase by 3, instead increased by {score.scoreAugment}");
                break;

            case 4:
                _assertAndLogTOM( condition: score.scoreAugment == 6,
                                  message: $"With a streak of 4 score should increase by 6, instead increased by {score.scoreAugment}");
                break;

            case 5:
                _assertAndLogTOM( condition: score.scoreAugment == 12,
                                  message: $"With a streak of 5 score should increase by 12, instead increased by {score.scoreAugment}");
                break;
        }
    }

    // If this code is reached, the test passed
    File.AppendAllText( path: "TOM Test Log.txt",  contents: $"{DateTime.Now}: Test passed\n");

    Console.WriteLine($"Completed testing {data.GetName()}");
}
```

Reflection and feedback

5. The most important thing I learned
My skills with linq queries have greatly improved

6. Most challenging aspect
Creating an algorithm to determine the unpaired dice, I approached it using two for loops then replaced the inner loop with a linq query

7. What I want feedback on the most
Coding style and readability