Class Assignment – Yahtzee
Part E

Due Date: 7th June, 2016 . Weighting: 40%

The Objectives of Part E

The objectives of Part E are threefold.

The first is to include a small amount of code which will assist in the marking of the assignment.

The second is to provide the means by which a game can be saved and later be reloaded so that the saved game can be continued.

The third is to implement the rules for **Yahtzee Bonus** and **Yahtzee Joker** so that a complete game of Yahtzee can be played.

Even if your code does not play the game correctly for two players you must implement the additions to your program as specified in the **first objective** below as well as the functionality for the **End Game** – see page 4.

First Objective

In the **Game class** add the directive `using System.IO;`
and then add the following two strings declarations as global class variables

```
public static string defaultPath = Environment.CurrentDirectory;
private static string savedGameFile = defaultPath + "\\YahtzeeGame.dat";
```

In the **Die class**, also add the directive `using System.IO;`
and then add the following declaration to the instance variables

```
private static string rollFileName = Game.defaultPath + "\\basictestrolls.txt";
```

and change **rollFile**, the StreamReader variable to the following, (it was **not static** in the original UML diagram of Part C) and add the instantiation to this declaration

```
private static StreamReader rollFile = new StreamReader(rollFileName);
```

Set the static class variable **Debug** to true.

In the method **Roll()** enclose your code in an **if** statement as shown below

```
if (!DEBUG) {

  // your original code is here

} else {
      faceValue = int.Parse(rollFile.ReadLine());
      label.Text = faceValue.ToString();
      label.Refresh();
}
```

In your original code you may not have use `label.Refresh();` this statement causes the **label** to be immediately redrawn. It is a good idea to use it whenever any visible property of a control is updated as its semantics guarantee that any change happens before anything else occurs in your program.

From the Assignment Specification page download the file **basictestrolls.txt** and place it in the **Debug** subdirectory of the **bin** subdirectory of the **Yahtzee Game** subdirectory of **Yahtzee**. *Wherever you have the project folder***Yahtzee\Yahtzee game\bin\Debug**

If you have followed the above directions, you should be able to run your program for two players with the supplied text file which assumes two players who on each turn roll the dice only once and select the scoring combinations in order, Ones, Twos etc.

That is, each player only rolls the dice 13 times and selects the next available combination after their first roll regardless of the dice face values. If you do this and each scoring combination is correctly implemented at the end of the game Player 1 will have a grand total of 314 and Player 2 a grand total of 121.

You should run through this process at least once. After that you are free to design your own test data or use this test file in any way you want.

Second Objective: Saving a game to a file

This objective will enable your game to be saved as a file if you need to stop playing but want to be able to continue playing the game at a later time. It is not necessary that you fully understand how this is achieved.

Download the zip file **Serialisation Code.zip** from Blackboard. Expand the file which contains four folders. In each folder, **XXX Class Load**, is the **Load** method for the specified class. The Game Class folder also contains the **Save** method and two helper methods.

Each method should be placed into the appropriate class of your project to replace the empty methods which were defined in Part C for the Game, Die, Player and Score classes.

Your code will not compile at this stage.

Saving a Game of Yahtzee

In the week 12 lecture reading and writing to files was discussed at the basic level of saving the separate individual variables of an object. The Save and Load methods of Game are similar to those in the lecture notes however they save and load objects through a process called **serialization (**known as **pickling** in **python).**

You serialize a class by marking it with the following attribute **[Serializable]** this serializes all members of a class except those marked as **[NonSerialized].**

Game Class

Place the following directive in **Game.cs**

```
using System.Runtime.Serialization.Formatters.Binary;
```

Place **[Serializable]** on the line above the class name:

```
[Serializable]
public class Game {
```

Place **[NonSerialized]** on the line before the **form** and **dieLabels** declarations:

```
[NonSerialized]
private Form1 form;

[NonSerialized]
private Label[] dieLabels;
```

Die class

Place **[Serializable]** on the line above the **Die class** name.

Place **[NonSerialized]** on the line before the following variables, **label** and **rollFile.**

Score class

Place **[Serializable]** on the line above the **Score class** name.

Place **[NonSerialized]** on the line before the following variable, **label**.

For all other classes:

Place **[Serializable]** on the line above the **class** name of **Player**, **Combination**, **CountingCombination**, **FixedScore**, **TotalOfDice** and **BonusOrTotal**.

Your code should now compile without errors.

ContinueGame "helper" method to Load method in Game class

ContinueGame() assumes that a **Game** has been saved at the start of a Player's turn before they have rolled the dice for "Roll 1".

This has been deliberately done to keep the code of **ContinueGame** generic and able to work with whatever code you have written in **Form1.cs** and **Game.cs**.

You should play a new game and at the start of some player's turn select **Save** from the **Game menu**. Of course you will have had to write an event handler for the **Save** option. The **Save** option should not be available until after **New** has been selected. The event handler just needs to class the **Save** method of the Game class.

The body of the event handler for **Load** option of the **Game menu** is simply

```
game = Game.Load(this);
playerBindingSource.DataSource = game.Players;
UpdatePlayersDataGridView();
```

This option should only be available when the program starts. Once the user has selected **New**, it should not be available.

You should test that you can save a game and then test that that saved game can be loaded the next time you run your program.

Third Objective: Scoring Yahtzee Bonus and Yahtzee Jokers

This functionality should only be attempted if your completed Part D with the ability at the start of the game to select the number of players playing the new game rather than the default number of two players which was the case up to the end of part D. The number of players can range from one to six.

To implement this additional scoring functionality, you will need to implement the **isYahtzee** and **yahtzeeNumber** instance variables and their associated properties as defined in the **Combination class UML diagram** provided in Part D.

These instance variables will need to be set by code added to each of the **CalculateScore** methods for all the **Combination types** and in the **CheckForYahtzee** method in the **Combination class**. Extra code will need to be added to the **CalculateScore** method of **Player** as well.

The **CalculateScore** method in the **Player** class should initially calculate the points for the combination the same as in the basic game. Once this has been done, there should be code which checks for the use of a **Yahtzee Joker** and determine if the **Yahtzee Bonus** points can be applied.

Whether a **Yahtzee** has been rolled previously or not, the player may still be able to use the **just rolled Yahtzee** as **a Joker**. According to the rules, you can use a **Yahtzee Joker** for the **3 of a Kind**, **4 of a Kind**, **Chance**, **Small Straight**, **Large Straight** and **Full House** combinations and providing the appropriate combination in the **Upper Section** has already been scored.

Since the **3 of a Kind**, **4 of a Kind** and **Chance** combinations will score exactly the same whether a **Yahtzee** is rolled or not, no extra coding is required here. That is why there is only a **PlayYahtzeeJoke**r method in the **FixedScore class**. Your code in **CalculateScore** will need to check that it is valid for **a Joker** to be played, according to the rules. This checking will need to use the **YahtzeeNumber** property (or getter). If **a Joker** can be played, then the **PlayYahtzeeJoker** method will be called on **the FixedScore object**, and this method should adjust the points scored to the full amount for that combination.

The **Yahtzee Bonus rule** means that if the dice just rolled produced **a Yahtzee** and **a Yahtzee** has previously been rolled and scored 50 points for the **Yahtzee combination**, then the player receives an extra 100 Yahtzee Bonus points.

The End Game

Should a game be played through to completion, you need to announce who won if they were at least two players playing and then ask if they wish to play another game.

For a solo player announce that the game is finished and do they want to play again.

Not accessible and not part of this assignment

It would be more realistic if a game could be saved at any time during play. **ContinueGame()** method would need to be rewritten extensively. If you want a challenge over the semester break try rewriting **ContinueGame** to handle that possibility. Restoring the saved game exactly as it was when saved is not straight forward.