**CS1400 Fundamentals of Programming**

**Programming Project #7**

**Dice Game**

**Version 1.0**

**Objective:**

This Project will give you significant practice in designing a complex Graphical User Interface (GUI) program. The objective of this project is to give you practice in functional decomposition and object-oriented design. When you finish this project you will have written a program that:

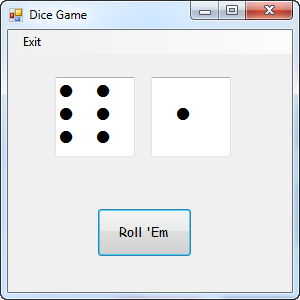
* uses at least one class that you have written to manage all of the business logic of the program,
* uses a number of methods to break the work of the program into manageable modules (pieces),
* uses at least one method that correctly uses parameters that are passed by reference and out.

**Project:**

In one of the labs you wrote a Console program that rolls a pair of dice. In this project you will write a Graphical User Interface program that does the same thing. However, in this program you must design and use a class that represents the pair of dice. All of the business logic that deals with the pair of dice must be encapsulated in this class, i.e. rolling the dice, determining if the roll was snake-eyes, and so on. The methods in your Form class should only deal with displaying information to the user and responding to events.

Since we have been studying methods, think carefully about what methods you want in the class you write to represent the pair of dice. You want methods that just do one thing, do it well and move on. Try to break the problem into several small pieces. You should design your class so that it contains at least one method that needs to take one or more of its parameters by reference and out. Do not write method(s) that arbitrarily takes a parameter by reference and/or out, there should be a significant legitimate reason for the method to do so.

You may design the user interface however you would like. One example is shown here:



You can get a program executable on Canvas.

If both dice are ones (***snake-eyes***), display a message that says "***Snake Eyes***". If both dice are sixes (***box-cars***), display a message that says "***Box Cars***". If the roll of the dice is neither snake-eyes or box-cars, display the face value of the dice.

How you design your program is up to you. There is no single right solution to this problem; however, there are outstanding, good, fair and poor solutions. Your program must work under all conditions. Remember that magic numbers are NOT allowed. Also be sure that your code is properly documented.

Format and document your code in accordance with the Course Style Guidelines.

**File(s) to Submit:**

Place your entire Project folder into a zip file and name the zip file  
Proj\_07\_your-initials\_V1.0.zip. For example, I would name my file Proj\_07\_DAF\_V1.0.zip. Submit this assignment as Project #7 on Canvas.

**Hints**

If you need some help writing the code for this Project, there are some hints below.

**Hints for Project #7**

**Where Can I Pass Parameters by Reference?**

Since a method can only return one value, what do you do if you want a method to return two or more values?

**Displaying the Values on the Dice**

You can do anything you want to in the project to show the user what values are on the dice. The easiest thing you can do is display the integer values in two ***TextBoxes***. The sample program uses a ***multi-line Textbox***. Each line of the ***TextBox*** is formatted to show the dots on the dice. The code to show the dots for a roll of three looks something like this:

**string three = "l" + nL + " l " + nL + " l";**

The ***Font Property*** of the ***TextBox*** uses the ***WingDings font***, The letter ***l*** calls out the dot character in the ***WingDings font***.

The ***nL*** in the above code stands for System.Environment.NewLine or \n.

|  |  |  |
| --- | --- | --- |
|  | **Grading Checklist** |  |
| # | Program | C(correct)  X(incorrect) |
| 1 | Meets & works to specifications | 6 points |
| 2 | Error Free, elegant & efficient | 4 points |
| 3 | Pseudo-Code | -3 points |
| 4 | Style Guidelines | -2 points |
| 6 | Source Files(s) & Formatting | -2 points |
| 7 | Project Prolog | -1 points |
| 8 | Function Prologs | -1 points |
| 9 | Zip Filename | -1 points |
| 10 | Lab & Project Names | -1 points |
| 11 | Zip File is invalid or will not unzip | Lab = 0 pts |
|  | Total Points | 10 | 0-9 |