ITSE 1430- Fall 2016
Program Set #3

See C# Grading /Program Guide Sheet for directions and grading/submission information.

Note: Create a windows application (form) program for each of the following problems.

1. Write a C# windows application to produce the country flag of Ireland. Center the flag on the screen. The length of the flag is 1.5 times the height. The form title should be named `Flag of Ireland`. Do not use an image.

Name the application: `IrishFlagXX`, where XX are your initials.

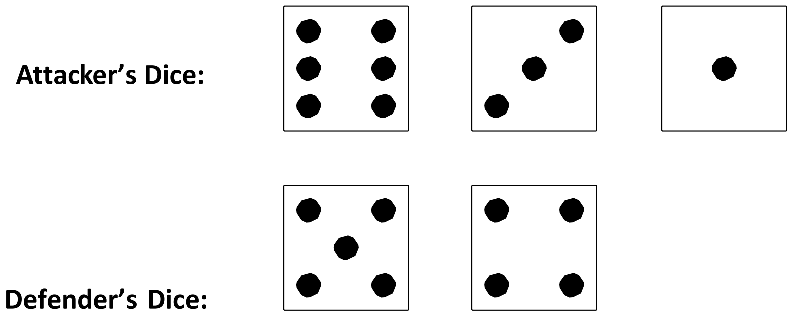
3. Write a C# windows application that plays the game Shut the Box. In the game, the numbers 1 through 12 are initially displayed. The player throws two dice and may cover the number representing the total of the dice or the two numbers on the dice. For example, for a throw of 3 and 5, the player may cover the 3 and the 5 or just the 8. Play continues until all the numbers are covered. Note, if every numbered tile higher than 6is covered, then only one die is used (disable the other die). The goal is to cover all the numbers in the fewest number of rolls. No points awarded - only keep track of the number of rolls. The form title should be named `Shut the Box`. Use appropriate controls as needed. Place all classes into one file. Dice provided by the professor in a file named `dice.zip.` Output should be user friendly.

Name the application: `GUIShutTheBoxXX`, where XX are your initials.

3. The board game *Risk* requires the throw of dice to simulate battles. Write a C# windows application that simulates *Risk* battle rolls. The rules are as follows.

> You, the attacker, will roll 1, 2 or 3 red dice: You must have at least one more army in your territory than the number of dice you roll. The defender will roll either 1 or 2 white dice: To roll 2 dice, he or she must have at least 2 armies on the territory under attack. To decide a battle, compare the highest die each of you rolled. If yours (the attacker's) is higher, the defender loses one army from the territory under attack. But if the defender's die is higher than yours, you lose one army from the territory you attacked from. If each of you rolled more than one die, now compare the two next-highest dice and repeat the process. In case of a tie, the defender always wins. The attacker can never lose more than 2 armies on a single roll.

Example Roll:



From the example above, compare the highest die from each person: 6 (attacker) and 5 (defender), so the defender loses one army. Next, compare the next-highest dice: 3 (attacker) and 4 (defender), so the attacker loses one army. In total, each player loses one army for this particular roll. Input will initially consist of the number of attacking and defending armies using a textbox. All battles will fight to the end, and assume the maximum possible dice for each roll. Output each dice roll and determine if the territory was captured or defended. When a dice roll is initiated, the dice should change to a new random configuration. The change should not happen immediately, however. Instead, the dice should change through a series of random configurations before settling on the final configuration, thereby simulating the look of a real dice roll. The roll should cause the dice to pass through roughly 5-15 configurations in 0.5-1.5 seconds. Use `Timer` and `Random` classes for this. Otherwise, use appropriate controls (labels, buttons, textboxes, etc.). in your application. The form title should be named `Risk Dice Roller`. The dice will be provided in a file named `dice.zip.` The sample run below is a text based example. You can use it help test your solution-graphically.

Sample Run (text based):

```
Enter number of attackers: 20           10  Attack: 4 3 1       9
Enter number of defenders: 18               Defend: 6 1         9

Roll                 Armies Left        11  Attack: 5 2 1       7
  1  Attack: 4 3 1      18                  Defend: 5 5         9
     Defend: 4 3        18
                                        12  Attack: 2 2 2       5
  2  Attack: 3 2 1      16                  Defend: 6 5         9
     Defend: 6 2        18
                                        13  Attack: 5 4 2       5
  3  Attack: 6 4 1      16                  Defend: 1 1         7
     Defend: 4 3        16
                                        14  Attack: 6 1 1       4
  4  Attack: 5 4 2      14                  Defend: 1 1         6
     Defend: 6 4        16
                                        15  Attack: 2 1 1       2
  5  Attack: 6 3 1      14                  Defend: 6 6         6
     Defend: 3 2        14
                                        16  Attack: 6           2
  6  Attack: 4 4 4      14                  Defend: 4 2         5
     Defend: 3 2        12
```

```
 7  Attack: 6 3 2       12
     Defend: 6 3        12
 8  Attack: 6 6 6       11
     Defend: 6 1        11

 9  Attack: 6 1 1       10
     Defend: 5 4        10
```

```
17  Attack: 1          1
    Defend: 4 2        5

Territory defended!!
```

Name the application: `GUIRiskDiceRollerXX`, where `XX` are your initials.

4 (**). Write a C# windows application that simulates a binary clock. Represent each of the six digits used for hours, minutes and seconds as a binary number. Since each number represents a single decimal digit, you will need 4 binary digits. To match a hardware version of the clock, represent the numbers vertically with the top LED representing "8". So, using "light off" for 0 and "light on" for 1, the clock below represents `09:50:34`.

Binary Values						Decimal value of row
0	1	0	0	0	0	8
0	0	1	0	0	1	4
0	0	0	0	1	0	2
0	1	1	0	1	0	1
H	**H**	**M**	**M**	**S**	**S**	
0	**9**	**5**	**0**	**3**	**4**	Decimal time ←

The clock starts with the current system clock time and is updated every second. Output both binary and standard time (digital). Use appropriate controls (labels, buttons, textboxes, etc.). in your application. The form title should be named `Binary Clock`. Output should be user friendly.

Name the application: `GUIBinaryClockXX`, where `XX` are your initials.