**CPSC1012 Core Portfolio 4 – Methods**

**Due Date: 11:55PM on Wednesdayday, March 25, 2020**. Submit to GitHub link provided on Moodle.

**Weight: 5% of your final mark.**

**Unicorn Race Game**

In the “Unicorn Race Game”, the player rolls a virtual 6 sided dice and the Unicorns (represented by the \* character) move that many spaces down the track (represented by the **=** character). Create a new Console application named ***CorePortfolio4-YourName*** for a program that will:

* Prompt and get the length of the track from the user. Ensure it is a number > 0.
* Clear the screen, then display the track (7 lines in height not including the border). Display the player’s Unicorn on row 2 and the computer’s Unicorn on row 5, just before the start of the track.
* Ask the player to press enter to roll the die and display the results of the roll (random number between 1 and 6).
* Display the Unicorns at the correct position they should be in after the roll.
* Once the Unicorn has **passed** the end of the track, clear the screen and display a winning, losing or tie message! If both Unicorns cross the finish line with the same number of rolls it is a tie!
* After each game has been played the user will be asked if they would like to play again by entering a ‘y’ or a ‘n’ (case does not matter). If they enter **y** start the game over again. Validate that user enters a ‘y’ or an ‘n’.
* When the user has decided to not play any more games, display a count of how many games they won, lost, and tied.

\*\*\*Your program must demonstrate at least one **while { } loop / Do { } while loop** and one **for** **loop.**

In addition you must use the following methods. You will need to determine the parameters and return datatype that each method will require. No method should be longer than 50 statements.

* **Main ()**
* **GetValidPositiveInteger () – Prompts for and returns an integer over 0**
* **DrawTrack() – Draws a length of track (represented by the = character)**
* **DrawCharacters() – Draws the Unicorns at the correct position after each roll**
* **GenerateRandomDieValue() - Returns a random number between 1 and 6**
* **GetValidCharacter() – Prompts for and returns a character ‘(y’,’Y’,‘n’,’N’)**

**SCREEN SAMPLES**

|  |  |
| --- | --- |
| Welcome to the Unicorn Racing Game!  Enter the length of the track: -50 Error: must provide an integer > 0. Enter the length of the track: | Track length must be positive integer. |
| ====================  \*  \*   ==================== Hit Enter to roll. | Initial starting position of the Unicorns must be just before the starting line. Keep this mind if you use the default Unicorn (\*) or a custom Unicorn (- - >). |
| ====================  -->   \*   ==================== Hit Enter to roll. |
| You moved 4 positions. Computer moved 4 positions.  ====================   -->   \*   ==================== Hit Enter to roll. | Hit enter to move the Unicorns down the track |
| Computer wins! Would you like to play again? | Prompt to play again |
| Player wins! Would you like to play again? F Error: Must enter ‘y’ or ‘n’.  Would you like to play again? | Must enter ‘y’,’Y’,‘n’,’N’ |
| You won 0 games. You lost 2 games. You tied 1 game. | Summary of game play. |

**\*\*Additional credit will be given if you use an additional method to allow the user to design their own Unicorn (string). \*\***

**HINTS!**

\*\*To assist you with placing the Unicorn you can use the following code which pads the string (e.g. \*) you are displaying with spaces in front of it so the string is right aligned at that position (i.e. currentPosition) on the screen.

Console.Write("\*".PadLeft(currentPosition));

**Example**:

Console.Write("\*".PadLeft(10)); //Places \* with 9 spaces in front

To clear the console window:

Console.Clear();

Remember to create the random object only once in your application to ensure true random number generation. It can be passed to a method when needed.

**Example**:

Random die = new Random();

static int NewRandom (Random rnd)

{

//use random object (rnd) as required.

}

**Do not make assumptions. If you have questions, ask your instructor.**

**Coding Requirements**

* A C# comment block at the beginning of the source file describing the **purpose**, **input**, **process**, **output, author, last modified date** of the program such as shown below:

/\*

Purpose:     \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input:       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Process(es):     \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output:      \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Author:            Your full name

Last modified:     yyyy.mm.dd

\*/

**Marking Rubric**

|  |  |
| --- | --- |
| **Mark** | **Description** |
| 5 | Excellent – program passes all scenarios with expected results; coding follows best practices and class standards; logic structure is efficient with no redundant code |
| 4 | Very Good – program produces the expected results for most of the scenarios; coding does not follow best practices and class standards; minor logic errors; redundant code |
| 3 | Acceptable – program produces the expected results for some scenarios; coding does not follow best practices and class standards; major logic errors; redundant code |
| 2 | Needs Work – coded all the requirements but program fails to produce expected results; coding does not follow best practices and class standards; major logic errors; redundant code |
| 1 | Unsatisfactory – coded less than 50% of the requirements; coding does not follow best practices and class standards; major logic errors; redundant code; aborts when executed |
| 0 | Not done; poorly attempted; major logic errors; major design problems |