Assignment 1 – LRC Report Template

Catie Bronte

CSE 13S – Winter 24

Purpose

This program is intended to replicate the game Left, Right, Center, a basic dice rolling game involving luck, and token passing between players. Random player names are pulled from a list, the number determined by how many players are playing, and a new random seed is generated to determine the "random" dice rolls for the game. The program then executes the game, exchanging turns between the players, displaying what happened during the turn along with the name. The turns rotate in a clockwise fashion until only one player has "chips" left. The game then ends, displaying the name of the player who won, and returns 0.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

Randomness

Whenever we want a set of random numbers, the computer has already pre generated a very long list of numbers that are seemingly of random order, which it can pull from. So when we ask for numbers, it picks a place in this long list, and sequentially takes numbers from it, one by one. But in a sense, since these "random" numbers are pulled from a pre-determine set of numbers that will never change, there is no randomness to it. If you know the location, you know exactly what number will come next. So in order to give the illusion of randomness, we have the computer / user generate a new location start (called a seed) in this random numbers list at the beginning of the game. This way each game has different numbers from the last.

What is an abstraction

An abstraction in computer science is a description of how the program code works, what its intended purpose is, what it is supposed to output, etc. Usually it takes form as a comment in the actual code, or as a report draft like this. Generally it is meant to help people (those who did not write the program, or forgetful coders), understand what is happening with the program, without having to figure it out for themselves.

Why?

Having to re-read through the parts of your program you think might have the bug, and having to re-familiarize yourself with what it does or how it works is very time consuming and frustrating. Especially at 11:50 at night. Simply having the descriptions there already makes this search much more efficient.

On the other hand, it is one thing trying to re-familiarise your self with code you have written yourself, but it is a much more difficult task trying to figure out what someone else's code is trying to do. Of course we usually understand what we mean when we write out code, and we are familiar with what we were trying to do in certain sections of the code, but other people have none of this to go off of. They are looking at the code fresh. Abstraction just help move that process along.

Functions

When it comes to writing functions, you really must wager if the function will ultimately make your code more efficient. Writing a function that only gets used once makes no sense, but writing one that gets used 100 times will make things a lot easier to read. So if you were to decided what you wanted to turn into a function, one must consider if it truly simplifies the code. The 2 most reasonable choices would be to make a function for print statement formatting, and one for dice rolling. Each element has different parts to it, and each line is printed a lot. However, if you had to break it up into 8 functions, you could make 2 different print variations, one function for dot, anything you can really thing of that gets used more than once. And if you couldn't make any functions, you would have to heavily rely on loops. I personally favor less functions over more because following the code can become more difficult and all over the place. I just just a few nested loops.

Testing

First off, it would be good to test if all of the dice rolls perform their proper actions. For that test you would just select a random die face and test what it does to the player chip count. Then I could also test and end of game scenario by setting all but one of the players chip counts to zero and seeing if the program ends the game. As long as I have at least one test for each phase of the game (Ideally more), that should be a good bench mark for getting my program operational.

Putting it all together

When testing it is easy to fixate on one specific scenario, and while it is nice to be able to look at a problem one item at a time, it is not very realistic. If there is an element of randomness to the testing then you could catch errors you might not have run into before, just testing what you anticipate to be issues. It essentially gives you unique scenarios to observe / test. In addition, abstraction helps when it comes to scouring your code to see what areas might generate issues or are causing the issues your tests are getting stuck on.

How to Use the Program

Your job is simple for this program. Simply enter the number of players you want in the game (an integer between 3 and 10 inclusive). Then enter a seed value (any positive integer). From the game plays its self and you don't have to do anything else.

Program Design

The code is broken up into 2 major sections. Pre-game and game code. There is a long line of comment that breaks up the two halves. Everything that defines the fundamentals of the game like num players and chips, or seed can be found in pre game. The actually running and looping of the game is found in game code. From there, comments describe the rest of the code.

Pseudocode

At the very top in purple are the libraries required to run the program, followed by the main definition. The next blocks of code define the enumeration of the die, the number of players given by the user stored in num players, and the seed as inputted by the user. If the number of players inputted is not between 3 and 10 it will default to a 3 and output a minor error message that does not stop the code. If the seed is ¡ 1 it will default to 4823 and output a similar error message. The seed is then inputted into srandom, and the random numbers will be ready to go. Following this is the definition of the chips array, which will be used to keep track of the exchanging chips through the game. Chips array length equals the number of players, each value starts out as 3. At the end of the pre game defining section there is some filler code that keeps enumerate from erroring. The following long section of code is the actual game code, and it starts with a long list of variable initialisation needed for the program. A forever loop encapsulates the rest of the program. First

the code checks if the game is over by looping through chips, and seeing if only one player left has code. If this is true, the game prints the winner name and returns 0. If this is not true the program checks to see if the index is at the beginning or end of chips, which affects how to chip passing functions. Then the program checks if the player has more than 3 chips. If they do then the max rolls is set to three. If not, chips = rolls. Then the program enters the dice rolling loop. If the player has 0 chips the loop is broken and prints that the player has 0 chips. If this is not the case, the loop generates a random number between 0 and 5, and this determines the action that takes place. Once the respective chip passing action has been done, it loops until there are no more rolls left, then ends with a print statement. When all of the rolls have been completed or the player has 0 chips, and the print statements have finished, the current player is set to the next person on the list. If they are the last person it loops back to the beginning.

Function Descriptions

This program has no functions and fully depends on loops to operate.