George Kalioras

CS-499

Enhancement Narratives

**THE ARTIFACT’S ORIGIN**

This single artifact has been used for all three enhancements in this ePortfolio,

It is a C++ program that simulates a board game called “Left Center Right” between multiple players. The nature of this game is that participating players are each given a starting number of chips representing their scores. During one’s turn, the player rolls a number of dice based on their current score, and depending on what values they rolled, they may lose chips or pass them to other players. Whenever a player runs out of chips, they are removed from the game. There is no way to increase the total number of chips held by players, and as the game goes on, that total gradually decreases until only one player remains with any chips, and they are declared the winner.

The artifact was initially created by me approximately one year ago at the time of writing this, for another class during my degree program.

**WHY IT WAS CHOSEN**

I selected this item because enhancing it pertained to skills I knew I was able to demonstrate for the purpose of completing this course.

The predominant components of the artifact which showcase my skills would be the multiple library functions which are each serve some purpose and are called respectively in the main method. The functions in question, after all enhancements have been made, are “void pause()”, “void print\_scores()”, “void GetPlayerInput()”, “void rollSequence()”, and “void login()”.

**ENHANCEMENT ONE**

This program experienced two major flaws. The first was that it crashed frequently when compiled, and the second was that players’ scores within the game were not being tracked properly. I was able to solve both problems. The first was fixed by adjusting the variable being updated in the middle of an iterative for-loop. The second was fixed by creating a second array containing the players’ scores, in addition to the first one, that would only store score outside of the die rolling sequence (the number of dice being rolled was based on the player’s current score, and said score was updating in real time). In between die rolls, these two arrays’ values would reconvene, and that way, the player will only roll dice based on their score at the beginning of their turn rather than in the middle of it.

One year ago I was not able to identify my programs or improve upon them. By re-examining it, I learned much in the way of error diagnosis as well as what methods were needed to fix them.

**ENHANCEMENT TWO**

This section, like the preceding one, contains & describes two improvements which collectively encompass “enhancement two”, because they were both made in the same time span and are present in the same state of the artifact during its changes in this course. The first and primary one was to migrate a large chunk of source code from the main method into a library function, “void rollSequence()”, which would be called in the main method instead. Tests following this process also exposed a few defects present in the code, which led to a secondary improvement i.e. to fix them. In this application, when a player rolls the number 3 on a die, the lose a point in their score. However, when the game is played this only ended up happening when the last participating player in the turn order rolled a 3, and when a different player did, their score was unchanged. I fixed this problem by identifying & removing an unnecessary conditional nest that remained from when a segment of the code had been lazily copy-pasted from one line to another.

My program also uses two array variables which keep track of the player scores; one of them updates in the middle of the die roll sequence, and the other, more important one converges on the first array’s value right after die rolling. An issue became apparent to me that caused player scores to not update correctly, particularly following the event of a participating player being eliminated from the game. The solution was to add another statement setting the inner-loop array equal to the outer-loop right after one of the players had been eliminated. I only caught this bug by running the program with three or more players instead of just two.

The third error was a simple one with a simple solution, but it also pertained to player elimination. Once a participant was removed from the game, the turn order needed to be set back by 1 since player numbers each shifted back from where the eliminated player had been in the turn order (“player 3 becomes player 2, and so on”). I only needed to add an “i--;” to the for-loop to fix this.

The primary enhancement of this milestone caused me a bit of trouble because I mistakenly assumed that every variable the library function takes as an argument needed to be a reference or a pointer, but in truth this was only the case for the one variable which was being updated in the function. After I realized this the rest of the process was smooth sailing.

The biggest challenge I experienced was remembering the fact that library functions don’t need such extensive usage of pointers or references when only one relevant variable is being updated by the function in question.

**ENHANCEMENT THREE**

I added a single layer of security by creating a library function which requires the user to enter a correct password before the rest of the application is permitted to run. The function “void login()” begins a do-while(input == false) loop, with the input variable of course being initialized to equal “false”.

I unconsciously believed for a time when creating this library function that it needed to take certain variables as arguments, but at some point I realized that was completely unnecessary. This process was a wake-up call for me, because previously I’d been in a mindset which led me to forget that not all functions require any arguments to work as intended. As a side note, I also learned that strings need to be enclosed in double quotations (“”) instead of single quotes (‘’) in order to be used as conditionals in if-statements.