**Project 1**

***Title***

**Hangman Game**

***Course***

**CIS-17C**

***Section***

**49215**

***Due Date***

**October 28, 2020**

***Author***

**Maliha Fatima**

**Introduction**

Title : Hangman

Hangman used to be my favorite game as a kid, and that is one of the reasons I chose to code this game. I am very familiar with the rules of the game, and I have a lot of experience playing it, so that saved me a lot of time when it came to deciding the layout of the game. My hangman game consists of two different versions of the game. One being the original hangman, and the other is the evil hangman. I spent the majority of my time on the evil hangman version because that one is a little different from the classic hangman, and I had to plan everything before getting started on it. I first coded the classic hangman and then coded the evil hangman. I began the process by writing all the concepts that I’m expected to utilize onto a paper, and then I tried connecting those concepts with the hangman game. I also referred to the c++ standard template library consistently, and that way, I was able to incorporate most of the concepts efficiently. I also looked at programs online that were associated with the STL library, so I have a clear understanding of the features and functions of each of them. After the project was assigned, I spent the first week exploring the STL library, and I think that really helped me when I started coding my game.

**Game Rules**

Classic Hangman and Evil Hangman are two player games. In my program the computer is one of the players.

**Classic Hangman:** When the game starts, the dashes are revealed according to the length of the word. That way the player knows how many letters they can guess. If the player makes a guess that is contained in the word, the other player reveals all the instances of that letter in the word and then the player takes another turn guessing the letter. If the player makes a guess that is not contained in the word, then a portion of the hangman is added (head, arms , torso, legs). The player has a total of 6 wrong guesses before they can guess the word correctly. The game ends when either all the letters in the word have been correctly guessed or when the player runs out of guesses.

**Evil Hangman:** When the game starts, the player is asked to input the length of the word that they would like to guess. Then the player is asked how many attempts they would like to have. The computer then finds a word of the length specified by the player and outputs the dashes according to the length of the word. The difference, however, is that the computer is actually cheating to make the game as difficult as possible for the player. The computer doesn’t actually settle on a word until the end of the game approaches. Instead of picking a word at the very beginning of the game like in the original hangman, it chooses a word but consistently changes it to a different word if the player starts guessing the letters correctly. The computer makes use of “word families” to narrow down to a word. Suppose the player enters “four” for the length of the word that they want to guess. Rather than choosing a word, the computer will make a set of all four lettered words that are in the dictionary. For example: Our initial word list could look like this :

ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX

If the player guesses the letter E, then the computer will partition the above words to form word families with respect to letter E.

ALLY B**E**TA COOL D**E**AL **E**LS**E** FL**E**W GOOD HOP**E** IB**E**X

So it would basically underline the instances of E in each word.

This is what the word family would look like:

* ----, which contains words ALLY, COOL, and GOOD
* -E--, which contains words BETA and DEAL
* --E-, which contains words FLEW and IBEX
* ---E, which contains word HOPE
* E--E, which contains word ELSE

Of the five families listed above, the word family ---- contains the most words and it is the most common one , so the computer will update the word list by eliminating all words not in this family. This yields the word list

ALLY COOL GOOD

And then the computer would report to the user that their guess was incorrect.

This is the idea behind the main process of the evil hangman game. The player wins the game if they guess the word correctly. If the player fails to guess the word, then the computer will display a word from the initial word list that the computer initially chose.

**Development Summary**

|  |  |
| --- | --- |
| Lines of Code | Around 500 |
| Variables | Around 30 |

|  |  |  |
| --- | --- | --- |
| Concepts | How it works | Location |
| Priority Queue | I used priority queue to store the name of the player. | main.cpp  line 45 |
| Sets | Stores a set of words with word length matching initial length. | Main.cpp  Line 278 |
| Maps | Contains the word families according to the word length and guesses. | Main.cpp  Line 314 |
| Stacks | I used stacks to keep track of the errors that the player makes. | Error.h |
| Find | I used the find function to find the letter used in the word. | Main.cpp  Line 171 |
| Count | Is used to count the number of dashes. Which determines whether the player wins or not. | main.cpp  line 292 |
| Bidirectional iterator | Uses maps that contain the word families to update the current guess set according to maximum size. | Main.cpp  Line 73 |
| Forward iterator | Used to iterate through the current word set to generate partitions. | Main.cpp  Line 323 |

**Pseudocode**

*Start*

*Prompt the user for their name.*

*Ask the user which version of hangman they would like to play.*

*If the player chooses the original hangman.*

*Call the original hangman game function.*

*Else*

*Call the evil hangman game version.*

**Hangman Pseudocode**

*Select a random word from the word list*

*Display the dashes according to the length of the word*

*Prompt the user for a letter.*

*If the letter is contained in the word*

*Prompt the user for another letter*

*Else*

*Display sections of hangman according to the number of errors*

*If player guesses all the letters correctly*

*The player wins*

*Else*

*Player losses*

**Evil Hangman**

***Initial steps***

*Read the dictionary file*

*Prompt the user for the word length*

*Prompt the user for the number of guesses*

*Prompt the user for whether they want a total of words remaining in the current word family*

***Start game***

*Display the dashes according to the word length that the player entered*

*Display how many attempts are left*

*Prompt the user to enter a letter*

*Partition the words into sets by world family*

*Divide the list of possible words into word families and choose the largest category (so the computer has more options)*

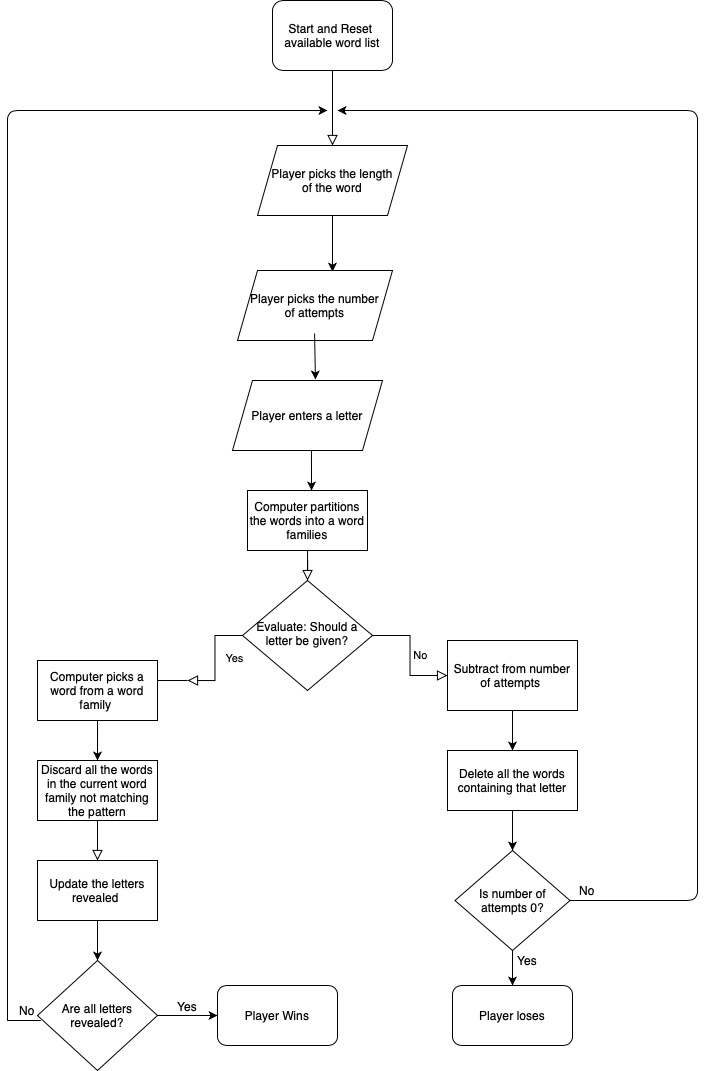
*If the player runs out of guesses*

*Display the word that the computer initially chose*

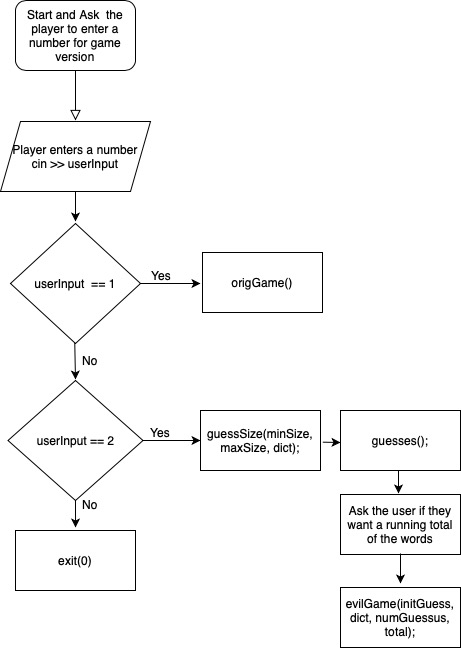
*Else*

*Player wins!*

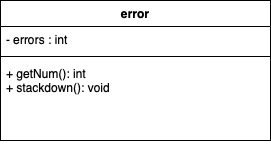
**Flowchart : Evil Hangman**



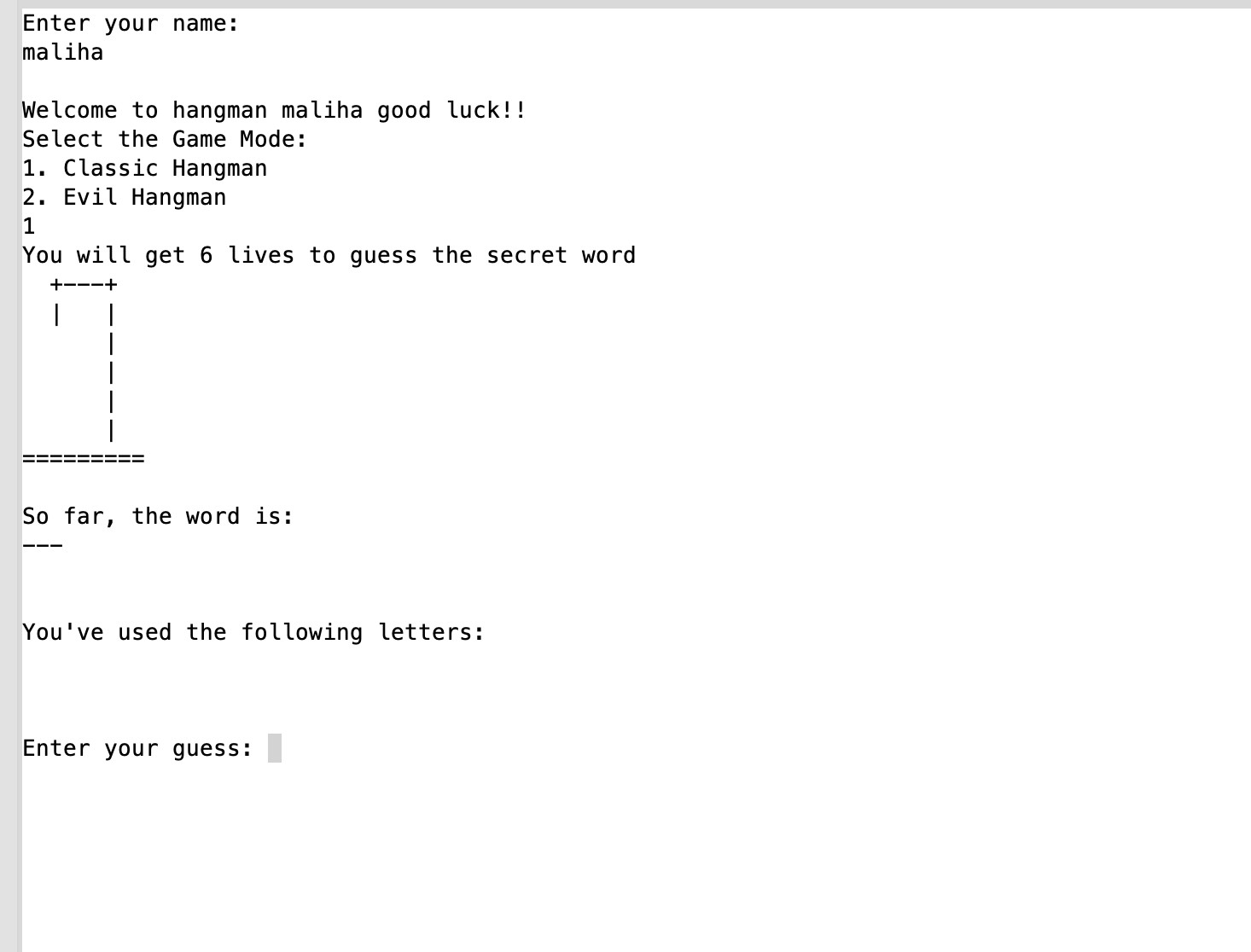
**Flowchart for the entire game**

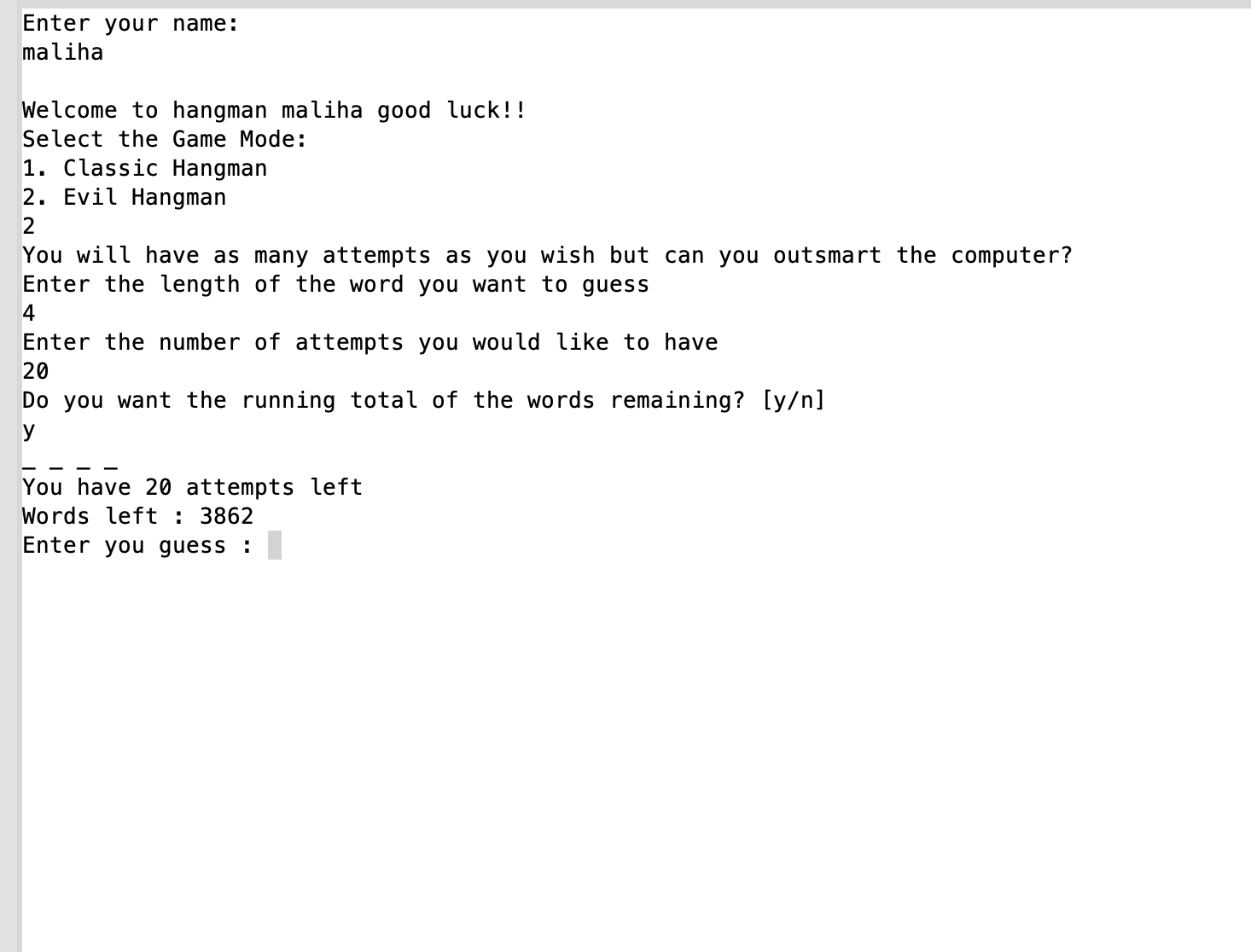
****

**UML**



**Sample Input/Output**

****

****

**Development process**

I made use of priority queue to store the user’s name. I then began to code the fundamentals of a hangman game. I declared all the variables that were needed. I created a vector of string to store the word list. I also used the random generator so that the computer generates a different word each time the user plays. I tried implementing the random\_shuffle function in my program but for some reason it wouldn’t pick words in random order. So I had to figure out an alternative way to achieve that. I created a class named “error” that keeps track of the errors using stacks. I created an object of that class in main and used that to output the hangman sections accordingly. For evil hangman game, I read in a file called “dictionary.txt” into a map of integers and strings. This file contains all the english words. I made use of maps and sets to partition the dictionary file according to the current word set. Partitioning the dictionary into word families took me a great amount of time to code. I had to keep referring to the textbook to understand the concepts deeply.