Introduction to Computing & Programming

C-CS111

***Fall 2022***

***Hangman***

Team Members

|  |  |
| --- | --- |
| Student Name | Student ID |
| Yasmine Mohammed | 22-101174 |
| Rahma Mohamad | 22-101209 |
| Manar Yousri | 22-101051 |
| Mohammad ali | 22-101166 |

**Contents**

[**Abstract** 2](#_Toc123825635)

[**1.** **Introduction** 2](#_Toc123825636)

[**2.** **System Design** 2](#_Toc123825637)

[ System modules and Functions 2](#_Toc123825638)

[ Hierarchy Chart 2](#_Toc123825639)

[ Flowchart 2](#_Toc123825640)

[**3.** **Team’s Workload Distribution** 2](#_Toc123825641)

[**4.** **User Manual** 3](#_Toc123825642)

[**5.** **Testing and Validation** 3](#_Toc123825643)

[**6.** **Challenges and Conclusions** 3](#_Toc123825644)

[**References** 3](#_Toc123825645)

# **Abstract**

The design and development of a Hangman computer game are described in this report. The software is created in python, written on Pycharm together with IDLE. The program that results from the design is modular in nature and makes the most of software reuse and abstract data types. A complicated Hangman game with levels and categories that raise without exception is created with special care. A synopsis of the game's development process is included in the report.

# **Introduction**

Hangman is a well-known word-guessing game where players try to piece together a word by guessing one letter at a time. A specific number of inaccurate assumptions causes the game to end and the player to lose. Additionally, the game ends when the player accurately guesses all letters of the missing word. The game is usually played with two or more players. Nonetheless, a human player will compete against the computer in the version we created. The player must guess a word or phrase that the computer has come up with by suggesting letters within a set amount of guesses.

The word to guess is displayed at the beginning of the game as a row of dashes, with each dash standing for a different letter. The computer will write a letter in all of its proper placements if the player guessing offers a letter that is present in the word. The computer adds one portion of a hanged stick figure as a tally mark if the recommended letter does not appear in the word. Typically, the game is over when the word is correctly identified or when the stick figure is full, signaling that all guesses have been made. Every time the player manages to guess the word with no wrong guesses the game is leveled up, and a new word is waiting for the player to guess.

# **System Design**

## System modules and Functions

Initiative values:

|  |  |
| --- | --- |
| Name | value |
| letters | list(string.ascii\_letters) |
| vowels | ['a', 'A', 'e', 'E', 'i', 'I', 'o', 'O', 'u', 'U'] |
| file | open('hangman\_words sample.txt', 'r') |
| text | file.read |
| words | text.split("\n") |
| words | "" |
| all category and level lists | [] |
| correctly\_guessed | [] |
| already-guessed | [] |

Letter\_frequency:

Checks how many times each letter in a word is repeated and returns the maximum

Vowel\_frequency:

Checks how many times each vowel in a word is repeated and returns the maximum

Lvl\_list():

Reads the sample text file and creates lists of separate categories by comparing them to the index of the name of the category in the file. ie: index(Vegetables:)>index(word)>index(Fruits:)

Then compares each word in each category to a set of criteria to determine its level

Using letter\_frequency and vowel\_frequency

level 1 words are the easiest words, have vowels repeated more than twice, and completely different letters

level 2 words are medium difficulty words, have vowels repeated only twice, and letter repeated twice or more

level 3 words are the hardest words, have vowels repeated only once, and letters repeated twice or more

Lvl\_checker:

Takes number of strikes, level, and category as an argument.

If the count of strikes is 0 the player goes up a level otherwise they remain in the same level. Returns the next word the player, category, level, and count is going to get.

Word\_generator():

Returns the word the player gets - easy word - when they choose to play a new game .

Draw():

If the player is playing a previous game the program uses fetched data otherwise the program uses initiative values

Play\_loop():

Prompts the user to enter their name

Checks if a file with the same name exists

If yes user is asked if they want to play a new game or continue a previous play

Otherwise, user is asked if they want to play or exit

Main():

Assigns values of fetched data from file to variables if user chose to continue a previous play and calls the previous\_play function otherwise calls the intial\_hangman function

Previous\_play():

Takes real\_word, display, word, already\_guessed, correctly\_guessed, strike count, level, and category as arguments.

This function is called when player chooses to continue a previous play

Calls draw() function

Initial\_hangman():

This function is called when player chooses to play a new game

Resets values of variables back to initiative values

Takes word value from word\_generator

Calls draw() function

Hangman():

Takes category, level, and count values from lvl\_checker

Calls draw() function

# **Team’s Workload Distribution**

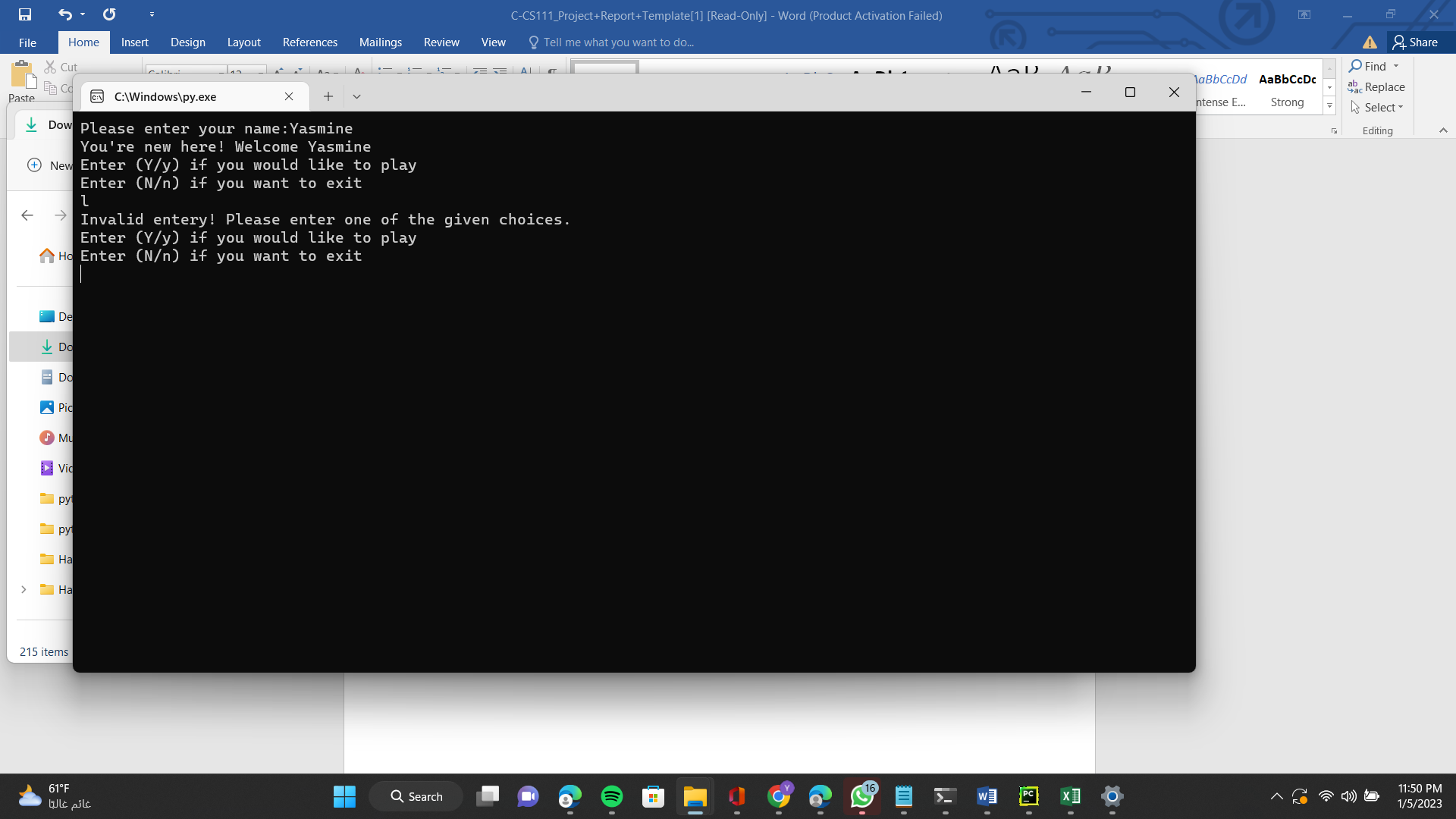
|  |  |
| --- | --- |
| **Name** | **Responsibilities** |
| **Yasmine Mohammed** | **Lvl\_list()**  **Word\_generator()**  **Lvl\_checker()** |
| **Rahma Mohamed** | **Draw()**  **Vowel\_frequency()**  **Lvl\_checker()** |
| **Manar Youssri** | **Draw()**  **Letter\_frequency()**  **Initial\_hangman()**  **Hangman()** |
| **Mohammad ali** | **Play\_loop()**  **Main()**  **Previous\_play()** |

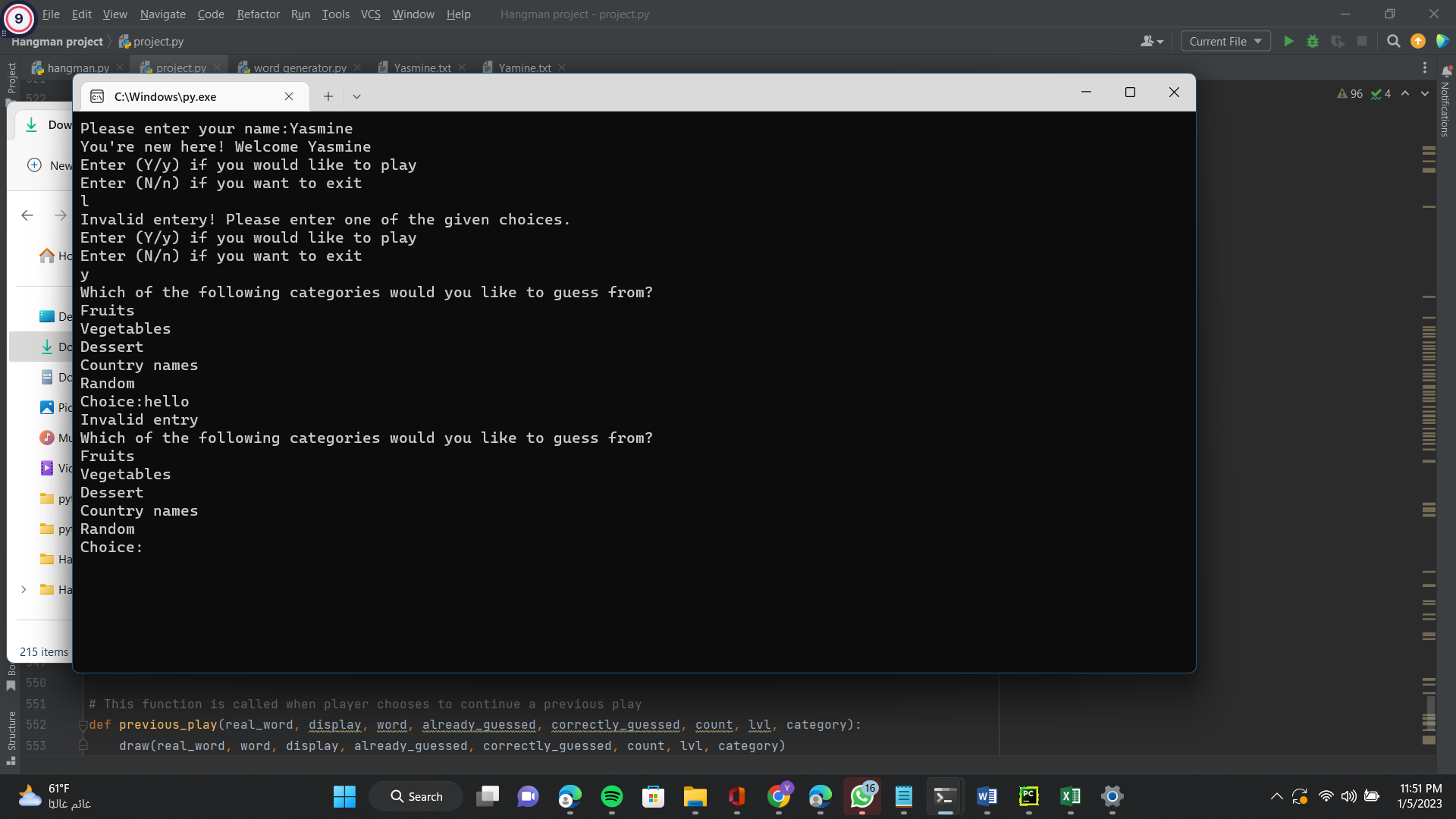
# **User Manual**

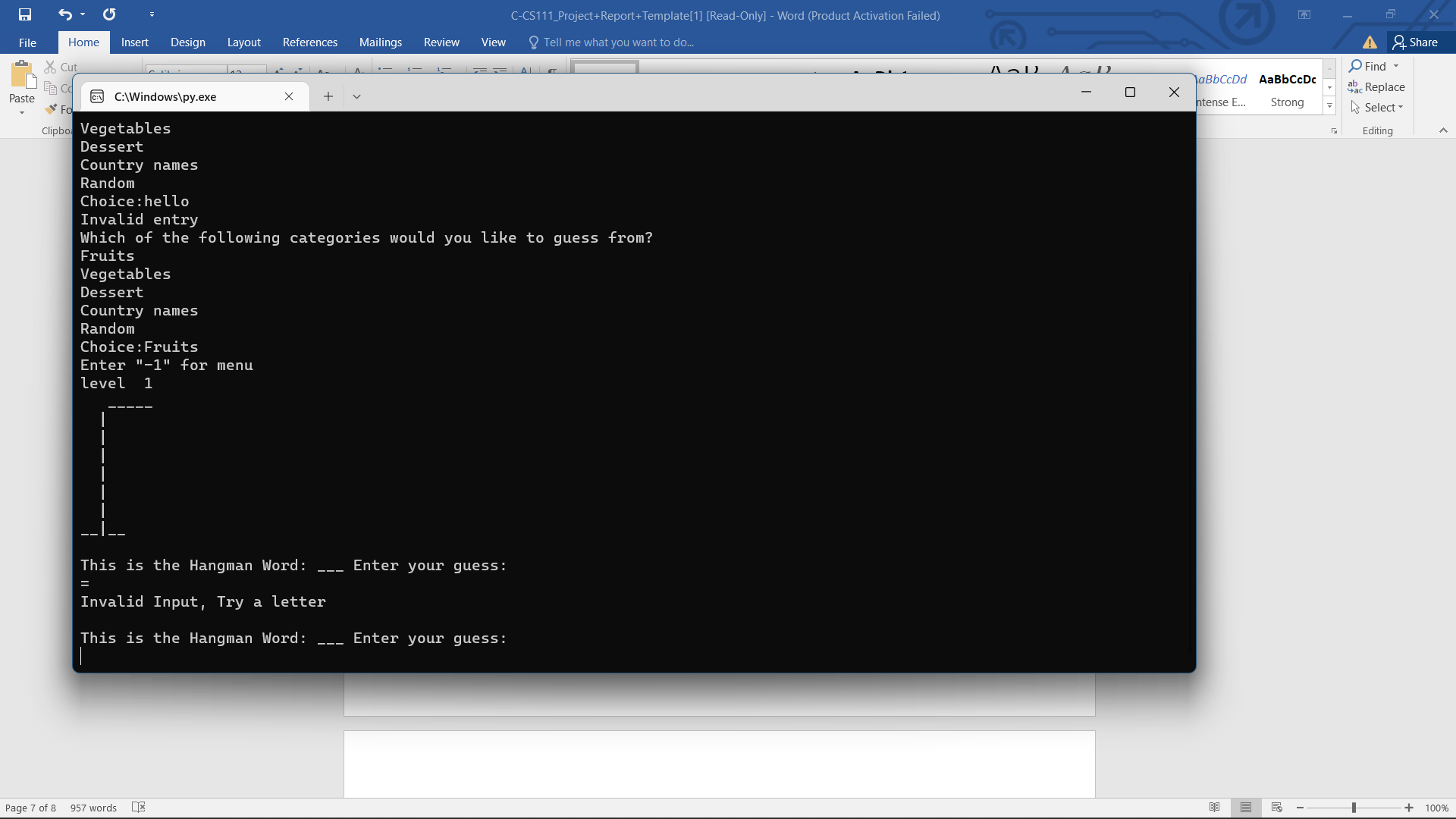
* When you download the game make sure it is saved where the python you will run the program on is saved
* Our hang man game allows you to choose from a list of categories to guess from
* Playing the game is simple just follow the prompts print when you enter anything
* When choices are given, enter them as they are exactly

# **Testing and Validation**

Think of tests that you can carry out to see if your system works. Remember to try and use normal, boundary and erroneous tests.







# **Challenges and Conclusions**

* I think our program met all requirements set
* We’ve learned that programming isn’t about getting the code right on the first go but about being persistent to fix and debug and errors
* One of the challenges we faced was that the program required more than one playing function as there was no way of knowing if the value of the count of strikes and level remained the same after playing or it is just the initiative value