CS Project Report

u1983738

## Introduction

The title of the task given to us was “Head or Tails Game”. As the name implies, it is a casual game, so we can assume that the intended users are children or people who wants to take a rest or relax in their leisure time because of its simple steps and clear victory. Brief descriptions of its functions are: 1. A Men-Machine game, the machine comes up with the result Head or Tail, and players guess. 2. According to the guessed answer, the machine judges whether right or wrong and player gets one point if correct. 3. The process repeats 10 times per game. 4. Scores need to be recorded and be able to access later.

## Requirements

We were asked to develop a solution and write a program in 2 hours using Python programming language. “Must-do” requirements of this task are listed clearly and thoroughly in the picture below.

A screenshot of a social media post

Description automatically generatedFigure 1: Task Description

However, there are several extra requirements we need to accomplish. Firstly, considering user stories, we can design a user interface (UI) including main menu with rules illustration. Secondly, games are more fun with rules and strategies, while under the original rule, players only rely on possibility (50%) to win. We can add more rules to the game, e.g. strategy like “spend 3 points to guess 5 more times”. In addition, functions like restart or multiplayer mode may increase user stickiness. Thirdly, we should detect player’s input and notify them to correct wrong input. Last but not least, the marking scheme suggests that we should use algorithms and data structures outside the course effectively, write helpful and consistent comments to get a HD.

## Implementation

To design a solution, we need to break down problems first (Decomposition).

A close up of text on a whiteboard

Description automatically generated

Figure 2: Top-down problem solving

This is an approach to problem-solving that involves starting with the big picture, then decomposed into several sub-problems where details are left until later. Using this method, the process of implementation is more manageable (Abstraction). From the picture there are some common characteristics of game development. Most of games have similar “structure” part as “Heads or Tails” game, so we only need to write new “main game program” part, apply this “template” and test for compatibility instead of starting to develop from scratch. This is called “structured programming”, which makes the implementation much easier (Pattern Recognition).

Problems are solved by applying algorithms. Here are my codes to achieve the given task.

A screenshot of a cell phone

Description automatically generated

Figure 3.1: Structure

This is the “structure” of the game. It has a selective structure nested in a cycle structure, which controls the process, entry of functions and exit. The “try…except” statement detects and handles the error. Components in each condition can be changed.



Figure 3.2: Clear screen

A picture containing table

Description automatically generated

Figure 3.3: Menu

Main menu interface and clear screen operation to provide a user-friendly experience.

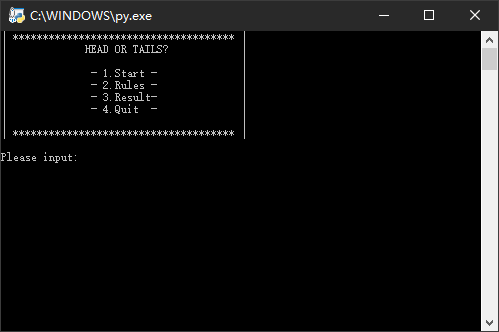


Figure 3.4: Menu interface

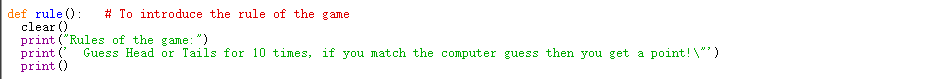


Figure 3.5: Rule

To show the rules of the game. Can also be changed

A screenshot of a social media post

Description automatically generated

Figure 3.6: Main Game

This is the “main game program” part. It uses a random seed to generate pseudo-random numbers with equal probability to represent the heads and tails, writes the name and score into a file, and provides a sense of reality by using time.sleep instruction. The words used to interact with the player is neither pushy nor disappointing considering the intended users. It makes the players feel relaxed and enjoy this casual game.

A screenshot of a cell phone

Description automatically generated

Figure 3.7: Checking results

To print the results in played order or sorted order. Another method used to deal with file operating.

Debug is also a crucial part of implementation. There are 6 typical types of debugging: Print Statements, Error Handling, Commenting Things Out, Debugging Tools, Tests and Asking Other Developers. Since the program is not that sophisticated, we can use small tests to debug – run the program a couple of times to test different input and check output. Comment time.sleep module to speed up the process.

## Programming Techniques Identified

Three basic structures (sequence, iteration, condition) are involved in the program. Validation, Annotation, Functions could be found from the images above. Libraries and Modules are used as below.

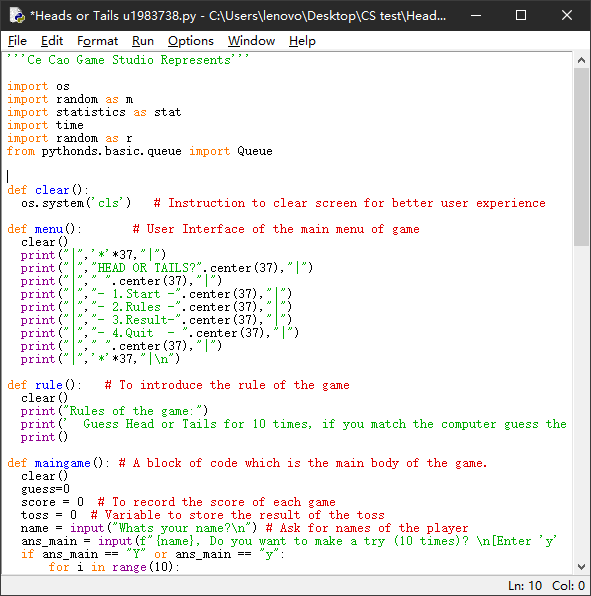


Figure 4: Libraries

Multiple algorithms could be applied when the results need to be printed in sorted order.

A screenshot of a cell phone

Description automatically generated

Figure 5.0: Original snippets.

(#Sorted line could be changed)

Common sorting algorithms are: Bubble Sort, Selection Sort, Insertion Sort, Shell Sort, Merge Sort and Quick Sort.

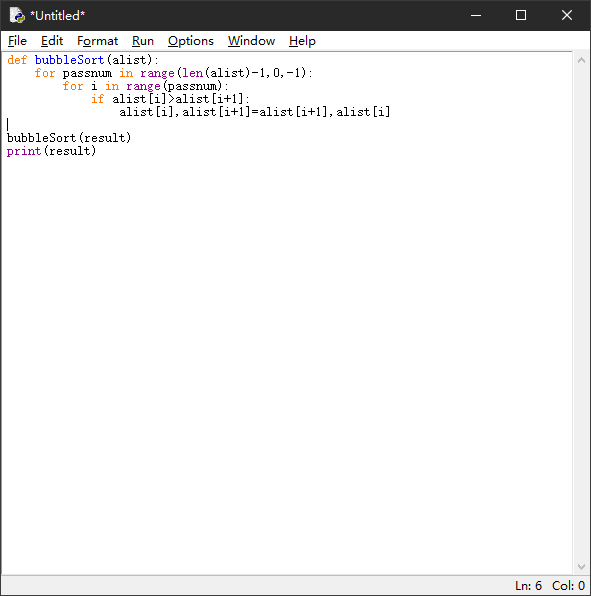


Figure 5.1: Bubble Sort,

A simple sorting algorithm.

It repeatedly visited the sequence to be sorted,

compared two elements at a time,

and exchanged them if they were in the wrong order.

Comparing: O(n^2)

Exchanging: O(n^2)

Total: O(n^2)

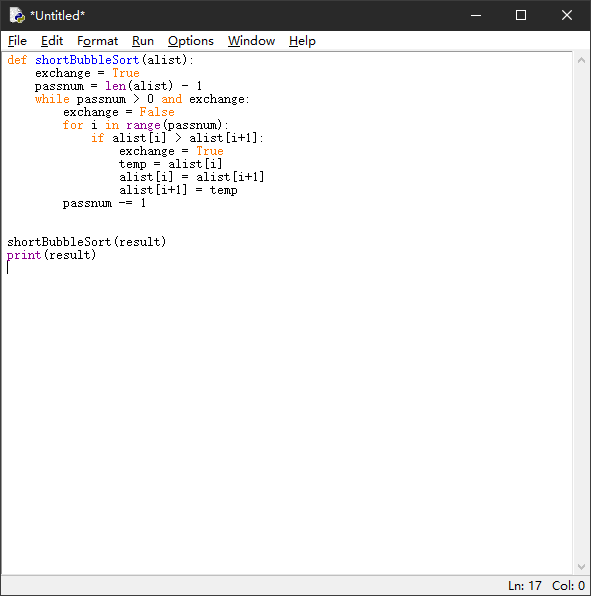


Figure 5.2: Bubble Sort, undated version

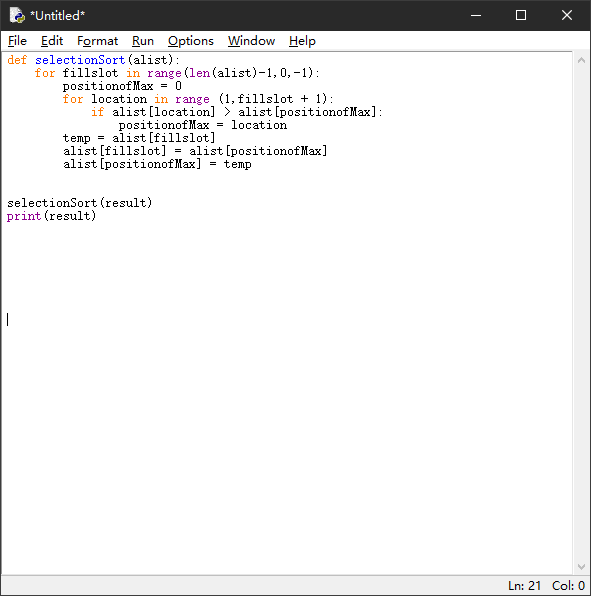


Figure 6.3: Selection Sort,

A simple and intuitive sorting algorithm.

First, find the smallest (largest) element in the unsorted sequence and store it at the beginning of the sorted sequence.

Then, continue to find the smallest (largest) element from the remaining unsorted elements and place it at the end of the sorted sequence.

Comparing: O(n^2)

Exchanging: O(n)

Total: O(n^2)

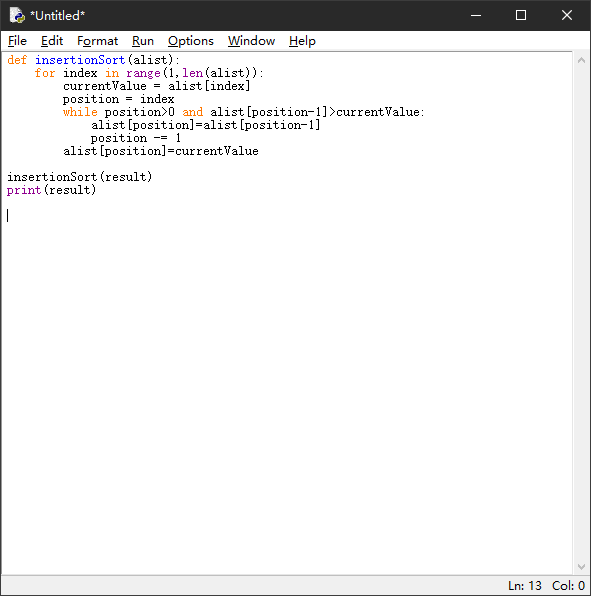


Figure 5.4: Insertion Sort,

A simple and intuitive sorting algorithm.

It works by constructing an ordered sequence.

For unsorted data, scan back through the sorted sequence,

find the corresponding position and insert it.

O(n^2)

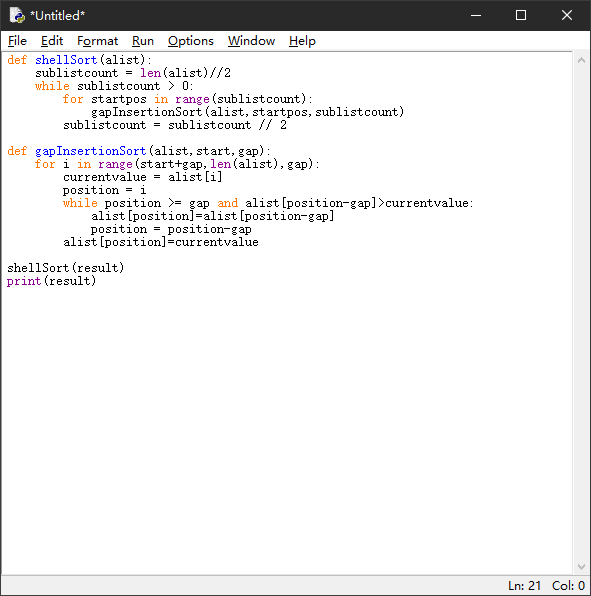


Figure 5.5: Shell Sort,

An improved version of Insertion Sort.

Split the entire sequence of data into several sub-sequences and apply insertion sort each.

O(n^3/2)

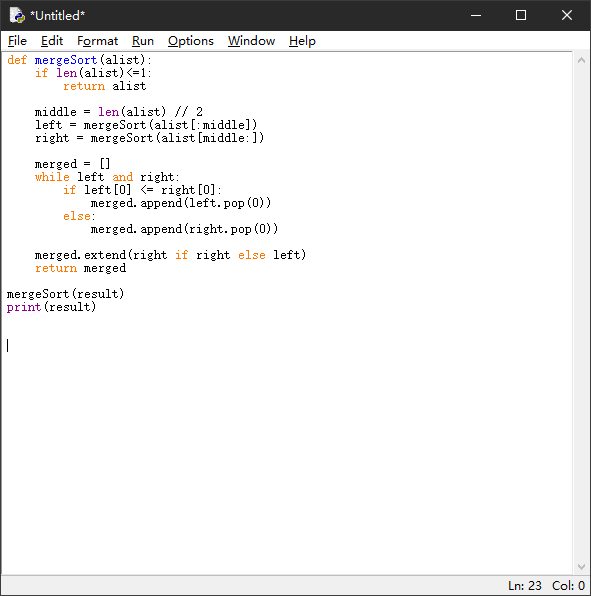


Figure 5.6: Merge Sort,

A very typical application of Divide and Conquer strategy.

The existing sub-sequences are combined to obtain a completely ordered sequence;

that is, each sub-sequence is ordered first, and then the sub-sequence segments are ordered.

Dividing: O(log n)

Combining: O(n)

Total: O(nlog n)

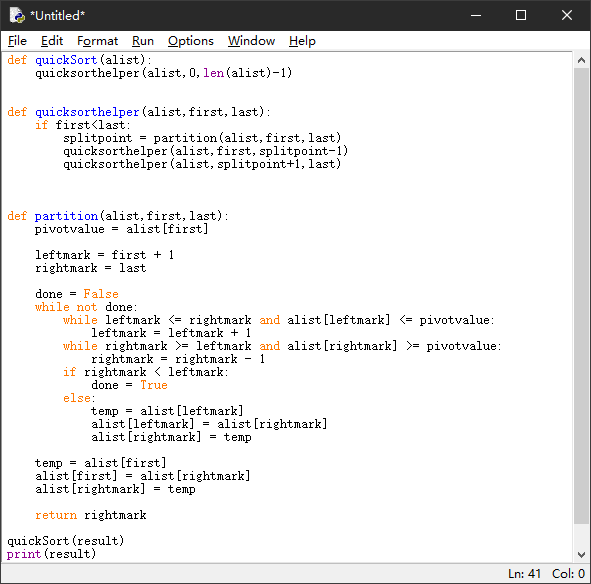


Figure 5.7: Quick Sort,

Separate the records to be sorted into two independent parts.

The keywords of one part of the records are smaller than the keywords of the other part.

Then the two parts of the records can be sorted separately.

Dividing: O(log n)

Combining: O(n)

Total: O(nlog n)

Need twice less storage than Merge Sort

A Queue is a linear structure which follows a particular order in which the operations are performed. We can use a Queue when the results need to be displayed in played order, because the order of Queue is First In First Out (FIFO).

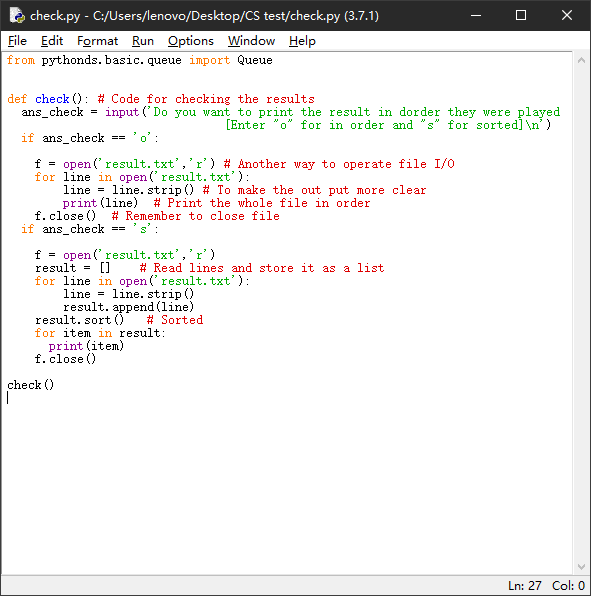


Figure 6.0: Original snippets



Figure 6.1: Queue operation

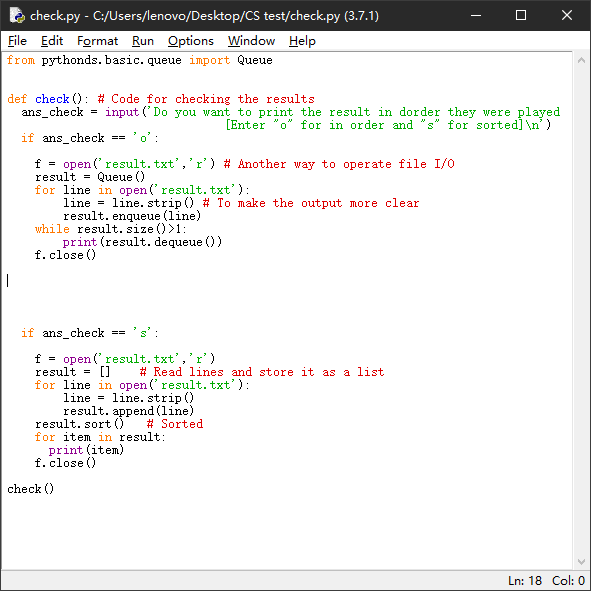


Figure 6.2: Using queue structure

Then we obtain the same result as before.

## Instructions & Testing

Example of Data that have been used for testing: Different player’s gaming records.

From the debugging process we can see the program meets users’ original requirements perfectly. It can execute the game correctly, record users’ gaming history and print them out in 2 orders asked.

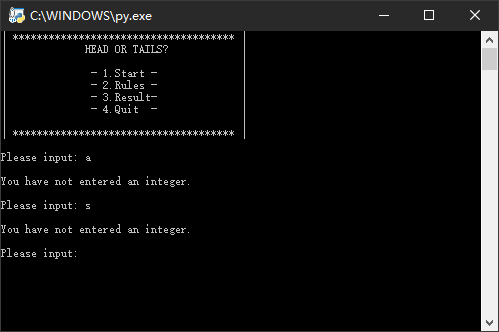


Figure 6.1: Debugging 1

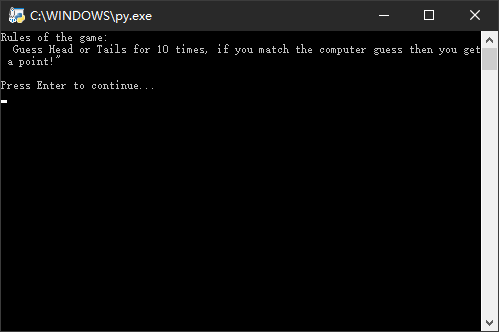


Figure 6.2: Debugging 2

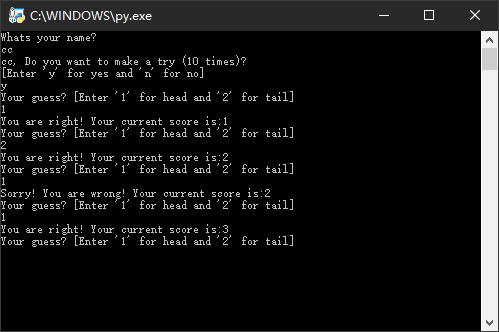


Figure 6.3: Debugging 3

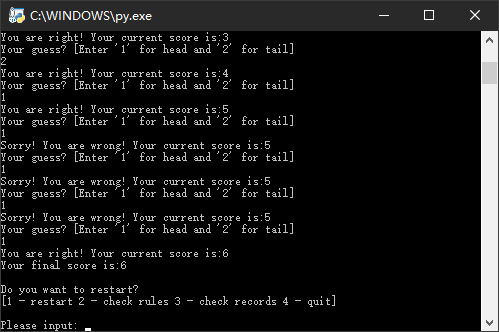


Figure 6.4: Debugging 4

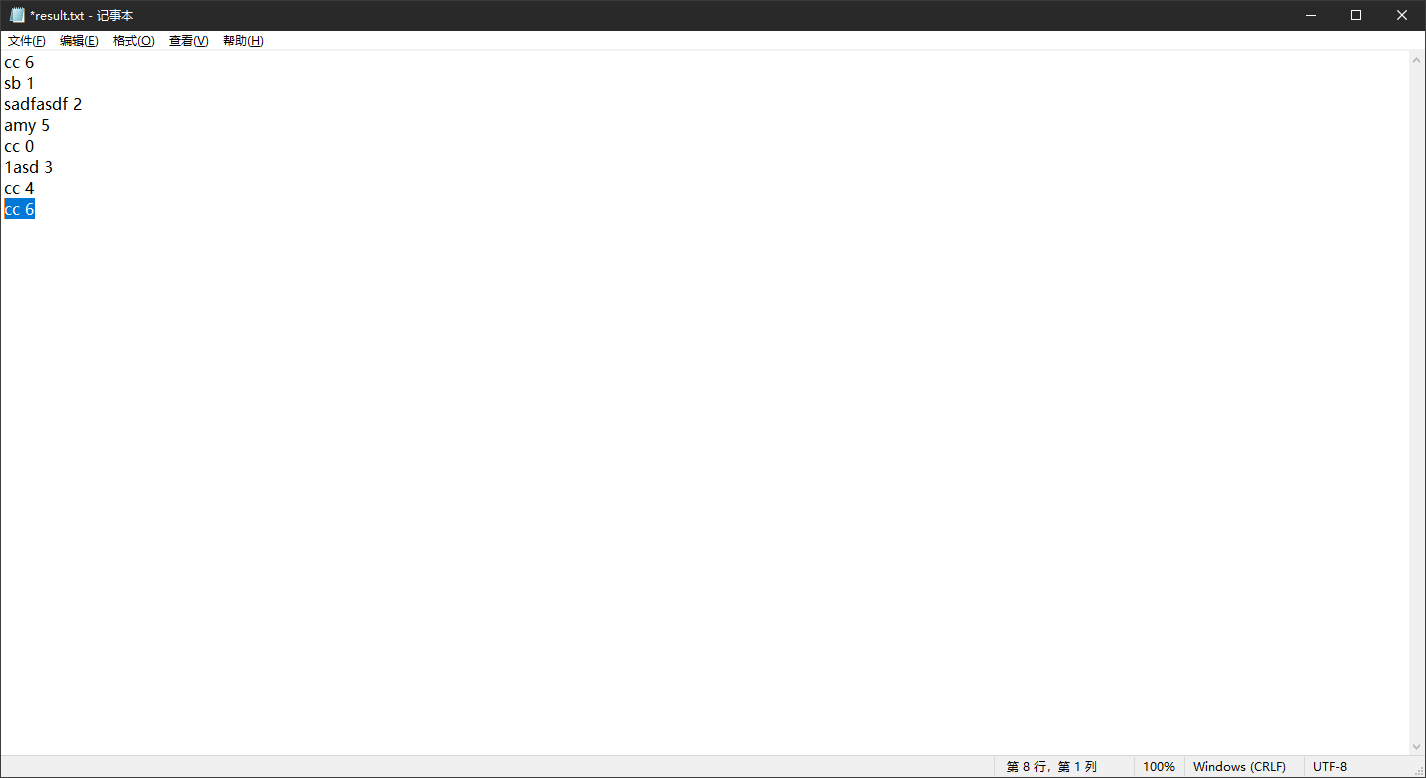


Figure 6.5: Debugging 5

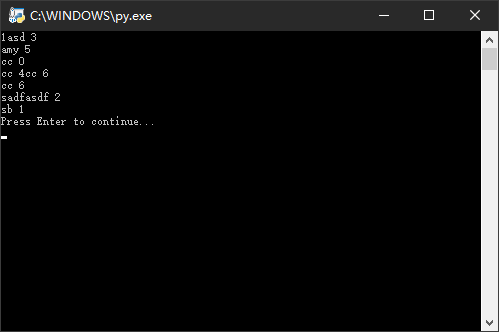


Figure 6.6: Debugging 6

To maintain the game program, we can promote several updates (game patches or more functions) later to make the game more interesting hence retain players. The debugging part will no doubt be more complicated in the future. A better method will be needed to debug our program. We can write a test program which has multiple random inputs then cross-checking the output or use debug tools for move advanced operations.

## Reflection and Next Steps

Overall, this project has been completed very well. It was finished ahead of time with extra functions contained. Previous exercise of dice game is helpful for this task because the method of function realization is similar (import random). Plenty of times left for debugging. The program would be better if I was able to discuss with other developers and exchange our ideas. In next steps we can add more functions to the game, for example multiplayer mode, other little games like dice game, or have some additional rules like “Spent 3 marks to guess 5 times” ( the Expectation of score if guessing 5 times is 2.5 points so the player have to decide whether take the risk or not). Here is an example.

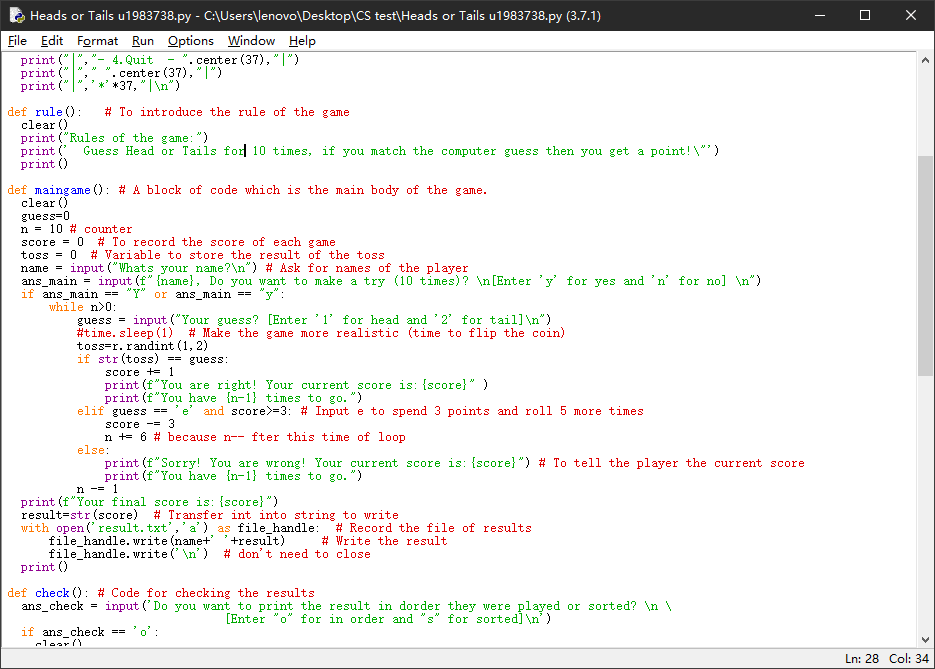


Figure 7.1: Code of main game after adding strategy

Moreover, designing a better-looking UI seems to be a good idea.

In order to do so, we can install OpenCV, Pillow and Matplotlib library to handle images. Here is a piece of code that can convert the picture we want into character drawing (ASCII art).

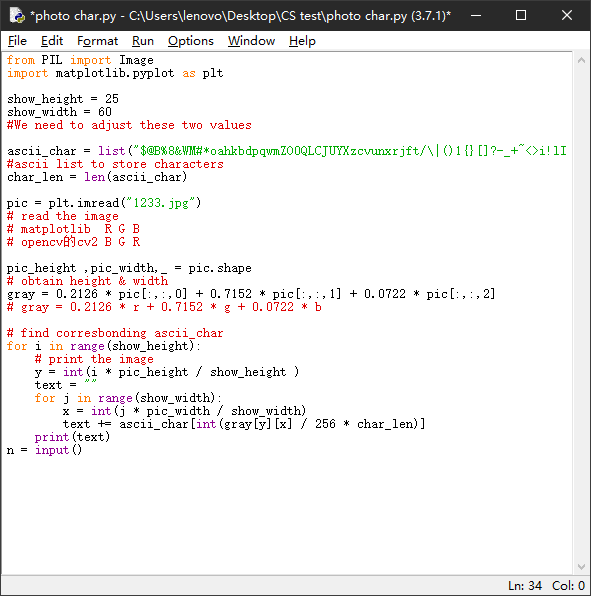


Figure 7.2: Code of ASCII art

Then we can obtain the drawing we want to use in designing UI.

A close up of a logo

Description automatically generated

Figure 7.3: Original picture



Figure 7.4: ASCII art picture we got

And at last, using pygame or Windows desktop application in the future is a good way to have better visual effects and interaction.

## References

1. Gibson J. (2014). Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#: Addison Wesley Publication.
2. Wing J.M. (2018). Computational thinking and thinking about computing. Discussion Meeting Issue ‘From computers to ubiquitous computing, by 2020’ organized by Marta Kwiatkowska, Tom Rodden and Vladimiro Sassone. Volume 336, Issue 1881.
3. Chen, J.W. (2015). Games Psychology: Communication University of China Press
4. Rouse G., O'Byrne S., Pitt J. (2015). My Revision Notes OCR A level Computer Science.
5. Codecademy (2019). How to debug your code? [online]. (Last updated 23 April 2019.)
6. gamasutra.com (2013). 10 Ways to Retain Your Players. [online]. (Last updated 31 January 2013.)
7. Pekin University MOOC (2019). Data Structure. [online]. (Last updated 14 April 2020.)
8. Geeks for Geeks: Queue-data-structure (2019). [online]. Available at: <https://www.geeksforgeeks.org/queue-data-structure/>
9. Blogs on Zhi Hu, CSDN (2019). [online]