Brian McDonald & Ed Powley

Introduction

In this worksheet, you will implement a version of the "terminal hacking" minigame from Fallout 4 (Bethesda, 2015); see Figure 1. In this minigame you must guess a secret n-letter word, one of several options presented to you. On choosing an option, you are told the *likeness*: the number of letters which match the secret word (i.e. the same letter in the same position). For example if the secret word is HOUSE and your guess is MOUSE, the likeness is 4 out of 5. If your guess is HOPES, the likeness is 2 out of 5 (the letters S and E do not count as they are in the wrong positions).

The GitHub repository contains a project named PartA_TerminalHacking for you to build upon. This contains code to read words from a file, and choose the secret word and other words. You will implement the rest of the game.



Figure 1: A screenshot of the hacking minigame in *Fallout 4*. Image from http://fallout.wikia.com/wiki/Terminal.

1

Algorithm 1 takes a guessed word and the secret word, and returns the likeness score as described above.

Implement the algorithm as a C++ function in main.cpp, choosing appropriate data types for the parameters, return value, and any variables.

2

Implement the main loop of the game, structured according to the flowchart shown in Figure 2.

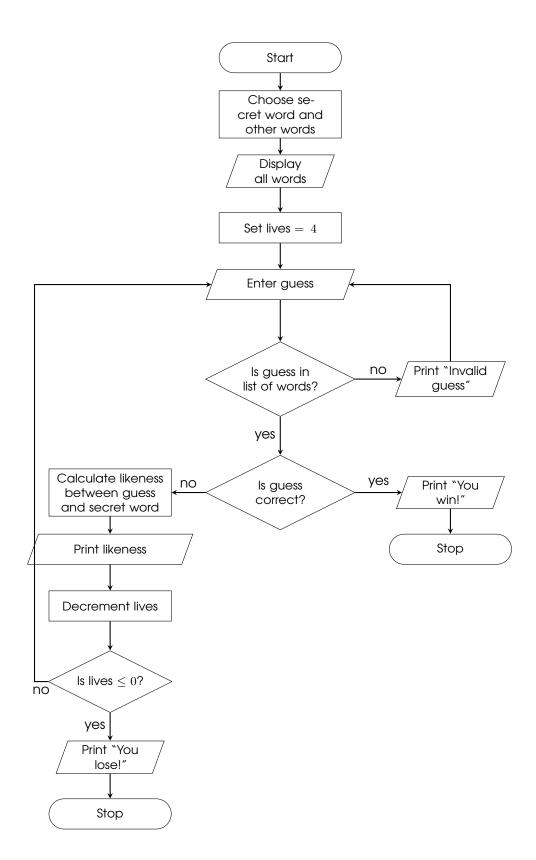


Figure 2: Flowchart for the Terminal Hacking game

Algorithm 1 An algorithm for calculating the likeness score for the terminal hacking minigame.

```
 \begin{array}{l} \textbf{procedure} \ \mathsf{GetLikeness}(\texttt{guessedWord}, \texttt{secretWord}) \\ \textbf{result} \leftarrow 0 \\ \textbf{for} \ i = 0, 1, \dots, \texttt{secretWord}. \texttt{length} - 1 \ \textbf{do} \\ \textbf{if} \ \texttt{secretWord}[i] = \texttt{guessedWord}[i] \ \textbf{then} \\ \textbf{increment result} \\ \textbf{end if} \\ \textbf{end for} \\ \textbf{return result} \\ \textbf{end procedure} \\ \end{array}
```

3 Stretch goal

In the skeleton project, the words are chosen at random. This may lead to instances of the game which are unsatisfying, for example where all words have a low likeness score with respect to the secret word.

Design an improved word choosing algorithm. Present your algorithm as pseudocode and/or a flowchart in your readme.md file on GitHub.

Implement your algorithm within your C++ project.

Submission instructions

Begin by **forking** the GitHub repository at the following URL:

https://github.com/Falmouth-Games-Academy/comp140-worksheetA

You should submit this project as part of the final coursework submission.

References

The following text will help you to complete this Worksheet, it has chapters which map quite nicely onto some of the tasks you have to complete.

Michael Dawson. 2014. Beginning C++ Through Game Programming (4th ed.). Delmar Learning.

Below is a link to the digital copy of the book and you should use this as the main reference text to complete the worksheet

http://voyager.falmouth.ac.uk/vwebv/holdingsInfo?bibId=792031

Marking criteria

Remember that it is better to submit incomplete work than to submit nothing at all.

To demonstrate **basic competency**, complete the following:

• **Timely Submission:** Obtain the marks for timely submission, you must submit (as a GitHub pull request). As with other worksheets, you may resubmit after these deadlines in order to collect extra correctness or quality marks. This is awarded as long as you submit *something* for each part by the deadline, even if your submission has bugs or other issues.

To demonstrate **basic proficiency**, complete the following:

- Achieve basic competency
- Complete Algorithm 1. Note: You will not be penalised for trivial errors which do not affect the overall functioning of your programs
- Appropriate use of GitHub, with descriptive commit messages
- Comments are used where appropriate, and are well written.

To demonstrate **novice competency**, complete the following:

- Achieve basic proficiency
- Complete Algorithm 2. Note: You will not be penalised for trivial errors which do not affect the overall functioning of your programs
- Your code is well formatted. Variable and function names are clear and descriptive.

To demonstrate **novice proficiency**, complete the following:

- Achieve novice competency
- **Design** an improved word choosing algorithm

To demonstrate **professional competency**, complete the following:

- Achieve novice proficiency
- Implement an improved word choosing algorithm