CS 11 Machine Problem 1 - Simple Word Unscrambler Game

University of the Philippines Diliman

September 2018

Instructions

- Before starting with the machine problem, please read the following chapters of *How to Think Like a Computer Scientist*:
  - Chapter 03, Functions
  - Chapter 04, Case study, interface design
  - Chapter 06, Fruitful functions
  - Chapter 09, Case study: word play
  - Chapter 11, Dictionaries
  - Chapter 12, Tuples
  - Chapter 13, Case study: data structure selection
  - Chapter 14, Files
- On this machine problem, your goal is to create a simple game. The terms program and game will be used interchangeably in this document. The terms user and player will also be used interchangeably.
- You can do the machine problem solo, in pairs, or in groups of three. For group submissions, please submit your names. A form will be posted in UVLe.

Submission Guidelines

- Submission links will be posted in UVLe.

Deadline

- The deadline of all deliverables is on November 4, 2018.
- Late submissions will receive a 10% deduction per day late, with a maximum of 70% deduction.

Evaluation

1. The machine problem will be evaluated based on the following:

1	Game Engine	60%
2	Interface and Randomization	10%
3	Source Code - Modules	10%
4	Source Code - Documentation	10%
5	Software Demo	10%
Bonus	GUI	10%
	TOTAL	100%
	TOTAL (WIITH BONUS)	110%

   Each item is described in the succeeding sections.

2. For group submissions, each member must rate other members by submitting a peer evaluation. Each individual student must also submit a self-evaluation. Peer evaluation and self evaluation forms will be available after the software demo.

3. For pair submissions, the grade of each student will be

   $$\text{(MP Grade)} \times \text{(Average of Peer Evaluation and Self Evaluation)}$$

   Where the average of peer evaluation and self evaluation is a number from 0 to 1.

1 Game Engine (60%)

For the machine problem, your goal is to create a single-player *Word Unscrambler Game*, where the player (the user), will gain points by *unscrambling* a set of letters to get certain words. You can view this game as a larger version of *Exercise 06, Problem 5.*

In implementing this game, your program must meet the following requirements:

1. **SEEDING WORDS (10%)**: When the program starts, it must get the words to be unscrambled from a dictionary that is external to the program. Given the *file name* of the dictionary, the program must be able to load its contents. Loading the dictionary must be performed only once during the run time of the program.

2. **PICKING WORDS (10%)**: For every round of the game, the program must pick a word/a set of words to be unscrambled from the dictionary. This leads you to the problem of retrieving a word from a dictionary given its position in the dictionary.

3. **SEARCHING FOR ANAGRAMS (10%)**: The player may have an option to choose a game mode. For example, one mode could be to get all anagrams of a certain word by unscrambling a word. This means that for every word in the dictionary, the program must be able to find all the anagrams of the word.

4. **COMBINING WORDS (10%)**: Another game mode could be to get all the words from a random sequence of characters, and not from a word.

   To ensure that at least one word can be generated from the random sequence of characters, the random sequence of characters must be generated from one or more words in the dictionary. Given a set of words, one approach to generate a random sequence of characters is by getting the shortest string containing the characters that are required to unscramble each word (for the sequence to be random, the words must be randomly picked by our program). The program must be able to generate the shortest string containing the characters that are required to unscramble a set of words.

5. **CHECKING WORDS (10%)**: From the string of letters that the program generated, the player will get points by entering a word whose letters are found in the string. Note that the word may not be in the set of words that the program picked (along with their anagrams) when generating the string. This means that a player will only get points if the the word that he/she entered contains letters that are found in the string of letters AND the word is found in the dictionary. The program must be able to check the two conditions.

6. **COMPUTING SCORES (10%)**: When a player unscrambles a word, the program must give points to the user. One way to compute scores is to give one point for every valid word unscrambled, where the total score is the total number of words. Another way is to compute the scrabble points of a word. In this approach, the total score is the sum of the scrabble points of all words.

   The program must be able to compute the scrabble points of all words. Moreover, given a string of letters, the program must be able to compute the highest score that can be achieved by unscrambling words from the string.

By meeting the requirements one by one, we will be able to do the game by combining the solutions. A huge part of this machine problem will be dedicated in formulating the program logic for the game engine. To prepare you in implementing the engine, you need to answer the problems found in `cs11mp1programlogic.pdf`. The problems are close variants of the requirements listed above. For the actual project, you have to *modify* the solutions that you made on the problems and create modules that provide functionality.

On the actual game, you have the option to modify how the game is played. Note that the game is in its most difficult form if you will integrate requirements 1, 2, 4, 5, and 6. You can modify how the game is played by modifying the algorithms, using a smaller dictionary, etc. However, you are still required to answer all the problems in `cs11mp1programlogic.pdf`.

2 Interface and Randomization (10%)

Once you're done with the game engine, your next goal is to design the interface of the game. The game engine contains the core algorithms in order to play the game, but it is not useful if there is no interface that lets the user play the game.

There are two suggested interfaces for this game:

1. **INTERFACE 1**: GAME WITH RETRIES (10 pts) - Given a string of letters, let the user guess a valid word until he/she gets a certain number of mistakes (around 3). The score of a user will be based on the total score of all valid words that are entered. This interface can be implemented on a terminal using the concepts that we already have in CS11.

2. **INTERFACE 2**: TIMED GAME (Bonus, 10 pts) - Given a string of letters, let the user guess a valid word for a limited time (around 1 minute). This may require concepts that are not in the scope of CS11, hence implementing this (along with a GUI) will give you bonus points. (See Bonus section for more information.)

3. **INTERFACE 3**: TIMED GAME WITH RETRIES (Bonus, 10 pts) - A combination of the previous two. Implementing this (along with a GUI) will give you bonus points. (See Bonus section for more information.)

You can implement at most 2 interfaces, but the minimum required is to implement interface 1. This means that if you are targeting to implement a game with GUI, you still need to implement interface 1. There are no strict rules on implementing the interface. You can add or modify the features (number of retries, program flow, displaying scores, strings, etc.) if you want, as long as you can provide interface 1.

To add unpredictability to the game, we have to randomize some of its components. To randomize some aspects of the game such as picking a word from the dictionary, use Python's `random`. The random module contains the function `random.randint(a,b)`, which returns a random number in the range $[a, b]$, `random.shuffle(l)`, which shuffles a list l, and `random.choice(c)`, which picks and element from a `collection` c at random.

3 Source Code - Modules (10%)

The program must be divided into at least three (3) user-defined modules:

- `engine.py` (4 pts) - This file must contain all the program logic of your game engine, including randomization.
- `interface.py` (4 pts) - This file must contain all the code for setting up an interface for your game.
- `main.py` (2 pts) - This is the file to be executed when a user wants to start the game.

Since you have to divide your code into modules, you need to encapsulate your algorithms into functions so that the algorithms can be shared across the modules.

You can add as many user-defined modules as you like, but the minimum is to separate the program into 3 major components: the game engine, the user interface, and the main program.

4 Source Code - Documentation (10%)

Your program must be properly documented:

1. (4 pts) Create a programmer's guide for your program. It is like a user manual but instead of "how to play the game," it answers "how the program is designed" and "how the program works."
2. (2 pts) Create a properly documented code by using descriptive names for variables, functions, and modules.
3. (2 pts) Add comments to the code if the source code itself does not suffice in describing what the program does.
4. (2 pts) All resources (books, sample codes, online resource or person) used should be properly cited in the source code (comments) and/or the programmer's guide.

5 Software Demo (10%)

After submitting your code, you have to present your program in public. This means that your audience may not only be your instructors. Your audience may include other CS11 students, other students, faculty, etc. The demo is scheduled on lab hours, so please make sure that you and your group mates are available.

During the demo, you have to present the following:

1. Sample Run *(5 pts)*
2. Source Code *(2.5 pts)*
3. Documentation *(2.5 pts)*

Bonus: GUI (10%)

For a more interactive and user-friendly interface, you might want to implement another version of your game, but this time, with a *graphical user interface* (GUI). The challenge in implementing a GUI using Python3 is there are no graphics libraries that are bundled with the default installation of Python3.

If you want to use a module that is not available in the in the default Python installation (e.g. graphics module), you have to download it using `pip` (`https://pip.pypa.io/en/stable/installing/`). The problem with using external modules is that there could be programs in your computer that also depend on those modules, and installing/updating a module for this machine problem might affect other programs that are using it. This means that the modules that your program use must be isolated from the modules that other programs use. One technique to isolate modules is by creating a *virtual environment* for your program.

If you are going to use an external module for this machine problem, please isolate your installation using the `venv` module `https://docs.python.org/3/library/venv.html`.