DSC 395T Algorithms and Data Structures	Group Project
Evil Hangman	Programming Assignment #2

In this assignment, you will write a program that plays an evil version of Hangman, which will give you experience in choosing efficient data structures to solve problems. In this course unless stated otherwise you are only allowed to use Python standard library and Pytest.

1 Hangman

Hangman is a 2-player asymmetric game, in which a word-maker creates a hidden word and a word-guesser tries to guess letters in the word with the ultimate goal of guessing the word. In the traditional pen and paper version of the game, the number of guesses is limited by the number of strokes it makes to draw a stick figure of a person. You can play Hangman by selecting the "Hangman" run configuration and clicking on Play within PyCharm. This will start a new game that you can use to learn the rules.

In this assignment, you will implement the brains behind an adversarial word-maker. Typically, the word-maker's only task is to select a word and to then provide feedback on the guesses made by the word-guesser, but your task is to develop a program that cheats by choosing the word as late in the game as possible.

For example, if the word-guesser has found 5 of the 6 letters, `DO-BLE`, then there are two words in English that satisfy these constraints—`double` and `doable`—so regardless of what the guesser chooses next, your program will indicate that the guess is incorrect, as your program essentially waits until the last possible moment to choose the word.

In general, your program will start with all words of a given length. Then, when a guess is made, your program should respond so that the set of possible words that remains is as large as possible. For example, one strategy would be to apply the following policy: if more words will remain possible without the letter, then we say that the letter is not in the word. Otherwise, say that the letter is in the word. This is a decent approach, but it is not optimal (why?).

To improve upon this approach, remember that as the cheating agent, we have the power to say exactly where the letter is in the word, not just whether the word contains the letter. Thus, instead of splitting the set of words into two sets (one containing the letter and the other not containing the letter), we should split the set of words more precisely:

- The set of words that do not contain the letter.
- The set of words that contain the letter only in the first position, the set of words that contain the letter only in the second position, etc.
- The set of words that contain the letter only in the first and second positions, etc.

You'll then want to select the largest of these sets. This new approach, which you will implement in this assignment, is a much better approach than our first approach (although it is still not optimal, because it doesn't take into consideration the number of guesses that remain).

To clarify the strategy you should implement, let's consider a quick example. Suppose the entire 4-letter English dictionary consists of the following words:

`ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX`

Now, suppose the word-guesser guesses the letter 'E.' Your program ostensibly needs to then identify the Es in the word that your program has "selected." Of course, your program hasn't yet selected a word, and so your program has multiple options about where you reveal the E's. Here's the above word list, with E's highlighted in each word:

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

You'll notice that each word falls into one of the aforementioned subsets:

- `----`, which contains `ALLY`, `COOL`, and `GOOD`.
- `-E--`, containing `BETA` and `DEAL`.
- `--E-`, containing `FLEW` and `IBEX`.

- `E--E`, containing `ELSE`.
- `---E`, containing `HOPE`.

Your program can now choose to reveal one of the five letter placements that corresponds to a subset of the dictionary. There are multiple approaches that you could take—you could pick `--E-` because `FLEW` and `IBEX` are difficult words to guess. However, to make the project well-defined, we will pick the letter placement that maximizes the length of the remaining set of words. In the case of a tie, we should prefer the letter family with the fewest letters, since it reveals as little information as possible to the guesser. Also, if `----` is one of those tied sets, it should be most preferred, because it will cost the word-guesser a strike. Otherwise, if there is a tie between two word families with equal amounts of letters (e.g. `-E--` and `--E-`), then either set can be picked.

In the example above, your program should choose to reveal no E's, leaving us with the following words:

```
ALLY COOL GOOD
```

This choice would give a strike to the word-guesser, marking their guess as incorrect. Let's proceed with this example one more time. If the word-guesser now guesses the letter 'O,' your program will create two subsets:

- `-OO-`, containing `COOL` and `GOOD`.
- `----`, containing `ALLY`.

The first of the two lists is larger, and so your program should reveal both Os. Note that your word-maker does not have access to the number of guesses remaining, so it cannot intelligently pick `----` if it is the guesser's final guess and there are words in that subset.

2 Your Assignment

You will find a Human implementation of a word-maker inside of `py_evil_hangman/word_maker.py`. This is called when running the Hangman run configuration inside of PyCharm. Your task is to implement the WordMakerAI class, which you can invoke in a game of hangman with the Evil Hangman run configuration.

Building on the starter code in `py_evil_hangman/word_maker.py`, create a word-maker program that cheats at Hangman. To do this, implement the `guess` method. This method takes a single parameter: the letter that is guessed. Then, you should return the positions of where the letter is, for maximal cheating. For instance, if you wish to return the letter family `-OO-`, return the sorted list $[1, 2]$ (since it is 0-indexed). If you wish to return the letter family `----`, return []. There is also a `reset` method that will indicate a game restart with a given word length. Unless you do Karma, please only edit the `py_evil_hangman/word_maker.py` file.

2.1 Efficiency

We want to try to make our Evil Hangman agent fast. To do this, we will want to preprocess the dictionary so that resetting the game can be done in O(1) time.

2.2 Other methods

You should also implement the `get_valid_word` method, which will be called when the game is over. This method should return one of the remaining valid words, which will be revealed to the word-guesser as the original word.

Additionally, implement the `get_amount_of_valid_words` method. If you run the game in verbose mode (-v), then the game will output the number of possible valid words that are available to your cheating word-maker.

Finally, implement the `get_letter_positions_in_word` method. You should call this within the guess() method to return the positions of a letter within a single word, as a sorted tuple.

2.3 Tips and Tricks

- Letter position matters just as much as letter frequency. In other words, `BEER` and `HERE` both have 2 E's, but define two distinct subsets. Do not represent subsets by frequency of a letter—you must also account for position.

- You can use the -f flag to specify a dictionary file, in case that helps with your debugging. The format of this dictionary file should be one word per line, similar to `dictionary.txt`. You can also use the -g flag to indicate how many guesses the word-guesser should get, to give yourself as many guesses as necessary.
- Do not explicitly enumerate over all possibilities of letter locations. There are 2^n different possible letter positions for a letter (why?), but only a few of those are actually represented by words in the dictionary.
- Hash Tables (also known as Dictionaries) are an especially useful data structure in Python and in this project. Find more information online or at the official documentation (click here).

2.4 Karma

As an optional challenge, implement an intelligent word-guesser inside of the `py_evil_hangman/word_guesser.py` file. There is a lot of freedom in how to design a word-guesser, so think carefully about your assumptions. If you do Karma, write about what you've done inside of a karma.txt file next to hangman.py. You can run Hangman or Evil Hangman with your WordGuesserAI by adding the -k flag.

3 What To Turn In

Use the **Gradescope** page of the **Canvas** website to submit your repository via a linked Github account. Please refer to the "Submission Instructions" page on edx for Gradescope submission instructions. Canvas is available at `http://canvas.utexas.edu/`.

Important Details.

- This assignment is due at **11:59pm** on the due date. Late assignments will be penalized 20% per day (unless your Slip days are assigned to it at the end of the semester).
- You may work in a group of up to 3 students, but make 1 submission per person.

Acknowledgments. This assignment was derived from the original Evil Hangman assignment produced by Keith Schwarz. It has been modified by Matthew Giordano and Calvin Lin.