Clever Hangman

CompSci 101

April 16, 2013
1 Introduction/Goals

The goal of this assignment is to write a program that implements a cheating variant of the well known Hangman game on the python terminal.

Contrary to the normal Hangman game where the word is chosen at the beginning, in this variant we want the word to change with each guess that the player makes. The change should be done so that the pool of possible words remains as large as possible. More specifically, after each guess the player makes we want the program to change the secret word respecting the following two conditions:

1. The new word is “harder” to guess. How to do that will be explained later.

2. The new word must be consistent with all the previous guesses.

The user/player isn’t aware that the computer is changing the secret word, because each time the player guesses, the computer may change the secret word in a consistent way that’s not apparent to the player.

In order to do so, you can use the code from the previous honest Hangman game, the code can be found here: [http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/Hangdemo.py](http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/Hangdemo.py) Using this code is optional.

Among others you will need a text file, which contains all the various words which from your main program will choose from. In this url [http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/lowerwords.txt](http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/lowerwords.txt) you will find the words with which the game will be played.
2 Specifications

This section explains in detail how the computer changes the secret word into another, which is consistent with the past guesses, but is more difficult for the human player to find. In order to do so, first we present an example of how the computer changes the secret word as the game progresses.

The output below shows how the computer plays a game of clever Hangman. The secret word is shown before each guess the human player makes, as well as the number of possible words that could be the secret word. In that sense, the secret word is not unique, but it is rather just one possibility out of the pool of consistent words.

Welcome to (Clever) Hangman:

Computer’s side: (secret word: curries) | words possible: 7359
Progress: . . . . . .
letters missed: i
i not in secret word

Computer’s side: (secret word: tresses) | words possible: 4048
Progress: . . . . . .
letters missed: i
guess a letter: e
you guessed a letter correctly!

Computer’s side: (secret word: waffles) | words possible: 969
Progress: . . . . e .
letters missed: i
guess a letter: a
a not in secret word

Computer’s side: (secret word: toppled) | words possible: 455
Progress: . . . . e .
letters missed: i a
guess a letter: o
o not in secret word

Computer’s side: (secret word: jumbled) | words possible: 159
Progress: . . . . e .
letters missed: i a o
guess a letter: u
you guessed a letter correctly!
An analysis of this game follows:

• Initially the first hint is shown to the user: the secret word has 7 letters. This means that out of our lexicon’s ‘lowerwords.txt’ 45,356 words, all 7,359 with 7 letters are possibly our secret word. The computer randomly picks up ‘curries’ as the secret word - this is not necessary but it will help illustrate the mechanism under which the computer changes the possible words.

• The first guess of the user is ‘i’. The computer checks out of all 7,359 words how many contain the letter ‘i’ and it sees that only 3,311 contain it, whilst 4,048 do not contain it. Based on this observation the computer decides that the letter ‘i’ is incorrect and thus changes its possible words to all the 7 letter words that do not contain the letter ‘i’. This changes the state of the game: the possible words are 4,048 and the new secret word is a randomly chosen word “tresses”.

• The user guesses ‘e’. The computer checks the possible words, either the word has no ‘e’ or it has one or more ‘e’ s in any position. After this check, the computer observes that the category with the most words is the one where ‘e’ is the second to last letter, with a total of 969 words. This changes the state of the game: the possible words are now 969 and the new secret word is a randomly chosen word “waffles”.

• The user guesses ‘a’. The computer checks the possible words, either the word has no ‘a’ or it has one or more ‘a’ s in any position. After this check, the computer observes that the category with the most words is the one where there is no ‘a’. This changes the state of the game: the possible words are now 455 and the new secret word is a randomly chosen word “toppled”.

• The user guesses ‘o’. The computer checks the possible words, either the word has no ‘o’ or it has one or more ‘o’ s in any position. After this check, the computer observes that the category with the most words is the one where there is no ‘o’. This changes the state of the game: the possible words are 159 and the new secret word randomly chosen is ‘jumbled’.

• Similarly in the last round shown, the user guesses ‘u’ and the computer decides that the pool of words containing ‘u’ at a certain position is bigger than all the other categories of words.
The important points that you need to implement are the following:

**Consistency** Whenever the computer changes its pool of possible words (and thus the secret word), it does so in a fashion which all the previous guesses of the player (right or wrong) are consistent with the new choice.

**Categories** As you saw in the example, after each guess the computer calculates the possibilities of the words and chooses the category with the most words. For example, when the user guesses ‘e’, the computer decides to show one single ‘e’ on the $6^{th}$ position of the word, it does so because that’s the largest category of seven letter words after the computer is enforced to reveal information about any ‘e’ in the word.

More precisely, after the user guesses ‘e’ the computer divides the possible words in the following categories:

- Words with no ‘e’: _ _ _ _ _ _ _ — 641 possible words
- Words of the form: _ _ _ _ e _ _ — 849 possible words
- Words of the form: _ _ _ e _ _ _ — 311 possible words
- Words of the form: _ _ e e _ _ _ — 41 possible words
- Words of the form: _ e e _ _ _ _ — 14 possible words
- Words of the form: _ _ e _ _ _ _ — 210 possible words
- Words of the form: _ _ e _ e _ _ — 172 possible words

etc.

As you can understand the number of possible categories depends on the pool of possible words at that time, and could potentially be very large. After splitting the pool of words into these categories, the computer checks the size of each category and chooses the largest one - in this case the second category.

You must implement the algorithm so the computer decides if the guessed letter is in the word or not, depending on the categories. *Whenever the user makes a guess, the pool of possible words should shrink by the least possible amount.* Moreover, if the computer decides that the guessed letter is in the word, then it must decide in which position(s) it is - like in the example above where it will choose that ‘e’ is second to last letter.

Ideally when the user has run out of guesses, the pool of possible words will contain more than one word, which essentially guarantees that the user lost the game.
3 Guidelines

In this section general guidelines on how to implement the categories part are given. You are not required to follow these instructions, as they are here to just give you an idea on how to proceed with your implementations.

The main idea in creating categories is to use a dictionary in which each possible template of words is a key and the corresponding value is a list of words that matches that template. For example, words matching:

't _ t _ _'

are:

[titan, tithe, title, titus, total, tutor]

To create this dictionary, you iterate over every possible word - initially this is all words, but the list of possible words changes after each guess made by the player. Then for each word, you either assign that word to an already existing template (or even increase a counter associated with that template), or you create a template with respect to the guessed letter where that word fits in the case that there was no template that the word fitted. This means that the first word will always create a template. An example is shown below:

Assume that the pool of words is:

["oboe", "noon", "odor", "room", "solo", "trio",
"goto", "oath", "oxen", "pick", "frat", "hoop"]

And the guessed letter is 'o'. Then the following dictionary should be created:

categories = {
    "o _ o _": ["oboe", "odor"],
    "_ o o _": ["noon", "room", "hoop"],
    "_ o _ o": ["solo", "goto"],
    "_ _ _ o": ["trio"],
    "o _ _ _": ["oath", "oxen"],
    "_ _ _ _": ["pick", "frat"]
}

In a game-playing program, the largest collection of values (most words) corresponds to key " _ o _ _" so the computer would pick a secret word at random from the list of three words: ["noon", "room", "hoop"] and the template for the game would be " _ o o _" with three possibilities. The player may think she has hit the jackpot with two o’s in the word, and that may be true, but there are more words to eliminate than with any other template! It is a clever hangman game.
4 Deliverables

Feel free to form up teams 2 persons in order to complete this assignment. Students who form a team have to submit separately their code and must write down their coding partner in the README file. The following are expected:

- The python module that implements the clever hangman application.
- A README plain text file that contains valuable information like additional comments, the resources you used to finish the assignment etc.

Working together is permitted, what is not permitted is working together without stating so in the README file. Plagiarism will not be tolerated and we expect everybody to comply with the Duke Community Standard.

Important Dates

- Turn in by April 25: Full points
- Turn in by April 28: 50% down

All deadlines are at 11:55pm on that day. Anything turned in after 11:55 on April 28 will not be graded.

How to submit: Either via Eclipse Ambient or the web submit system. Links to these two methods can be found on the Large Assignment tab on the course website.