Cheater's Hangman

Andrew Rosen

Abstract

You will be writing a special game of hangman. It will test your ability to use many data structures in concert. **This is pair programming assignment, so you are allows to work with ONE other person.** You will work on this assignment in class on the 26th and 28th.

Credit goes to Keith Schwarz for the original assignment. I used, by which I mean pretty much stole, many of his exceptional explanations for this assignment, although I did reword it in many places.

1 Premise

Playing a game relies on trust. We intrinsically assume that if we are playing a game against someone, then the rules will be followed. In other words, people aren't prepared when someone might be cheating at a game, which makes it an *excellent* strategy (if you prepared to accept the consequences).

Your goal is to write a computer program that allows a user to play a game of Hangman, but with a hidden twist. If you don't know what Hangman is, go look it up on Wikipedia and play a few games on a whiteboard or a piece of paper with your friends.

A core assumption in hangman is that the person who knows the word **does not change the hidden word.** Our program is going to to that and do it without getting caught. Here's an example of how this might work.

Suppose the player has made a lot of guesses and has only one wrong guess left. The player's guesses have revealed:

```
GOA_
```

Based on the player's previous guesses and knowledge of the English language, there are only two possible words left: "GOAD" and "GOAL." In a normal game of hangman, where the person with the hidden word is playing according to the rules, this means the player has a 50% of guessing the right word.

In our game, the computer will cheat: if the player guesses the blank letter is "L," the computer will say something to the effect of "Sorry, you're out of guesses! You lose! The hidden word was "GOAD."" If the player guesses "D" instead, the computer pretends that the hidden word "GOAL" the whole time. In other words, the player will always, always, always lose in this scenario because the computer will cheat about what the hidden word was.

2 How To Cheat at Hangman for Programmers and Time Travelers

Let's learn how to cheat with a more thorough example. Pretend you're playing hangman and have to choose a four letter hidden word for someone else to guess. You also happen to have a dictionary, unlimited time, and a galling, tragic, yet hilarious lack of morals.

Rather than choosing and sticking with a single hidden word like you're supposed to, you make a list *all* four letter words. There's a lot of them and many are inappropriate in an academic setting, so for simplicity's sake, we'll pretend English only has the following four letter words:

```
THIS TEAM LOOK GOOD EGGS ECHO EDGE
```

So you gather all those words into a list and lie to the player, telling him you've chosen your hidden word.

The player starts by guessing 'E,' a sound strategy, since 'E' is the most common letter in the English alphabet! You now have to reveal all the E's in your word, if any. You haven't chosen a word yet though, so you have a few options. Let's underline all the E's in our list of possible hidden words to see all out options.

```
THIS TEAM LOOK GOOD EGGS ECHO EDGE
```

We can categorize each of these into what we'll call **word families**, which is grouped by what the hidden word would look like to the guesser if we had chosen that word as our hidden word.

- The player would see `____` if the hidden word was "THIS," "LOOK," or "GOOD."
- The player would see `E___` if the word was "EGGS" or "ECHO."
- The player would see `_E__` if the hidden word was "TEAM."
- The player would see `E__E` if the hidden word was "EDGE."

You have to choose one of those families and that will limit your future choices. For example, if we choose the second word family, that means we're committing to pretending we choose a hidden word that started with "E," so we can't lie about that any more. If we go with the first group, we commit to pretending our hidden word has no "E" in it. We can't go back later and pretend it has an "E" in it, because the user will be able to remember he guessed that. Remember, we don't the guesser to figure out we're cheating.

Which word family should we go with then? There are many, many valid strategies. Maybe you decide to go with the word family with the most obscure words. Maybe you choose the word family that reveals the least letters.

I will use a much simpler strategy: choose the word family with the most words. Like any programmer with a propensity for cheating or time traveler, we're afraid of commitment, so we'll choose the largest word family to avoid being boxed into a single hidden word for as long as possible. Again, this isn't necessarily the best strategy, but it is a relatively quick one and works rather well. We tell our guesser their guess is wrong, reveal ____, and now our list of hidden words is:

```
THIS LOOK GOOD
```

Next they guess "O," so your word families are:

- _OO_ containing "LOOK" and "GOOD"
- ____ containing "THIS"

So I'll choose _OO_ as the word family and reveal the "hidden" O's.

The game ends when the user guesses all the letters, which is unlikely given all the cheating we're doing, or if they run out of guesses. If all goes according to plan, the user will never know they've been hoodwinked!

3 Cheater's Hangman

Your goal is to create my Cheater's Hangman, which plays Hangman, but cheats by avoiding choosing a hidden word according to some strategy. Here's the flow of the program.

1. Read the `dictionary.txt` and store it in your program at runtime. This is the list of words your program will use.
2. Ask the user to choose the size of the hidden word they want to try to guess. Keep asking them if there are no words of that length.
3. Ask the user to choose how many wrong guesses they get to have before they lose.
4. Create an initial list of hidden words using the dictionary and choosing all the words that have the length the using asked for.
5. Play hangman using our cheating algorithm:
   (a) Print out the revealed letters, their wrong guesses, and the remaining number of wrong guesses
   (b) Get the user's new guess and be nice and ask them to reenter their letter if they already guessed it.
   (c) Separate your list of hidden words into word families based on the input.

(d) Choose a word family using some strategy (I used the word family with the most words) and make that the new hidden word list. If this reveals letters to the user, reveal letters, otherwise the player loses a wrong guess.

(e) Keep going until:

(f) If all the letters are revealed, the player wins. If they player is out of wrong guesses, randomly pick a hidden word from the hidden word list and reveal it to the user, pretending that was your hidden word all along.

6. Ask if they want to play again.

First and foremost, so long as your program works, there is no right or wrong way to write this program. However, sitting down and thinking about what data structures to use will go a long way towards making your life easier. Plan before programming!

3.1 One last important thing

You'll want to remove it when you play against your unsuspecting victims, but you should print out the size of you hidden word list at each guess. This will let you know your program is cheating

4 Advice

Since this is coming right after learning about `Maps` and `Sets`, that's probably a good indication that you should use those data structures. The key here is to use a `Map` to help with word families, as maps are good at grouping things into categories.