Programming Language Paradigms

Haskell Lab 4 Evil Hangman

CS-231 **Spring, 2019**

Evil Hangman was presented as a Nifty Assignment at SIGCSE, the special interest group in computer science education, in 2011. This project was presented by Keith Schwarz of Stanford University. I have made modest changes in the project for this course.

The basic idea for this project is, it is difficult to write a program to play a game well. The area of artificial intelligence looks at how to write programs to make the computer seem smart. IBM has had success with this, they have created programs/systems to do well in Chess and on the Jeopardy game show. **The game you are programming will not use any fancy game playing, it will cheat!**

Here are the rules for hangman.

1. One player, player 1, chooses a secret word, then writes out a number of dashes equal to the word length.

2. The other player, player 2, begins guessing letters. Whenever he guesses a letter in the word, player 1 reveals all locations of that letter in the word. A letter not in the word counts as an wrong guess. The number of wrong guesses is actually a constant in hangman. (For this program, the number of wrong guesses will be a function of a command line argument to the program.)

3. The game ends when all the letters of the word have been revealed, in which case player 2 wins. If player 2 makes a number of wrong guesses equal to his limit, then the game ends with player 1 the winner.

When playing the game, player 1 draws a picture of a hanged man in stages; the picture is completed when player 2 runs out of guesses. We will not be drawing pictures as part of this project.

In a fair game, player 1 honestly picks a word before play begins; shows the correct number of letters; and shows locations of correctly guessed letters. **This is not a fair game.** The number of letters will be fixed before game play, but the word that player 2 is trying to guess will not be fixed, it will be chosen from a family of words as play continues. For example, suppose that player 2 has been guessing letters and is at the point of the game where the revealed letters are

H_ND

Both of the words 'hind' and 'hand' fit this pattern. Assume that these are the only two words in the dictionary that fit the pattern. In a fair game, one of these words was chosen from the beginning. Let us assume that the word chosen was hand. Then if the player 2 guesses 'A' he wins. In this version of the game, player 1 has not chosen a word. If player 2 guesses 'A' as a last guess, player 1 wins by revealing the word 'HIND'. However, if player 2 guesses 'I', player 1 wins by revealing the word 'HAND'. This is not a fair game, player 1 has a large advantage.

Here is a larger example of play given in Keith Schwarz's handout:

Suppose you are player 1 and you are supposed to choose a word of length 4. Instead of choosing a secret word, you will instead compile a list of all 4 letter words. Assume that the following list contains all 4 letter words in the dictionary in order to make the discussion simple.

ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX

Your opponent guesses the letter 'E'. You must tell your opponent which letters in the word you have picked are E's. Remember that you are cheating; you have not picked any word. Look at the patterns in words determined by the pick of an 'E':

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

There are 5 families of words determined by the 'E':

- _ _ _ _, the words ALLY, COOL, and GOOD.
- _ E _ _, the words BETA and DEAL.
- _ _ E _, the words FLEW and IBEX.
- E _ _ E, the word ELSE.
- _ _ _ E, the word HOPE.

Since you must eventually have some word in the dictionary, you will choose the family with the largest number of words at this step. In this case, it is the first listed family corresponding to the words ALLY, COOL, and GOOD.

Given that you now have this three word family, if your opponent chooses the letter 'O', this gives two families:

- _ O O _, the words COOL and GOOD.
- _ _ _ _, the word ALLY

At this point, you would choose the family _ O O _ because it contains two words, and the other family has only one word. Picking a letter that does not appear anywhere in the list of words making up a family has the effect of making a wrong guess, the family of words to choose from remains unchanged.

Requirements

This is a 60 point project. You will implement a solution to the evil hangman problem. Note that in the examples, all letters output by the computer are upper case. You will keep your output (for the word pattern, word, and guessed letters) in upper case. In particular, print guessed letters in a String of size 26 with a letter guessed in its proper alphabetic place, a letter not guessed will have a space in its place in the String. For example: "A EF K NO V Z". The word pattern will consist of letters correctly guessed and underscores. Separate these characters by a space to make it easier to read. For example: "_ E _ _ R _".

Your program will do the following:

- Use command line arguments to control the particular run of the game. These command line arguments in order are

1. The name of the dictionary file. We will be using the same dictionary file we used for both spell checking assignments.
2. An integer representing the length of the word. This must be a valid word length, that is, there must be at least one word in the dictionary with this word length. If that is not the case, your program will print an error message about invalid word length and exit.
3. An integer representing the number of wrong guesses allowed. If this integer is smaller than five (5), print a message that the number of wrong guesses allowed is 5 and set that value to 5. If the number of guesses is greater than 10, set the number of wrong guesses allowed to 10, and print a message stating that this is so. For any number in the range 6 to 9 inclusive, set the number of wrong guesses allowed to that value. Note that when the player makes the number of wrong guesses allowed, he has lost the game.
4. This last command line argument is not necessary for a run of the program. You will not actually employ it when you are using your program to cheat at a game of hangman. This command line argument will consist of the string of characters '-s'. This will be used to print the size of the current family of words with each output after every guess. This is useful when you are testing your program (and when I am grading it).

If the command line arguments are wrong (wrong number, wrong form) Your program will write a message on how to run the program and exit, something like:

usage: ./Hangman dictionary_name length_of_word number_of_guesses

Note that the fourth command line argument will not be documented in the usage. If the fourth command line argument is not exactly the two characters '-s', this argument will be ignored. If there are more than 4 command line arguments, the usage message will be printed.

- All the words in the dictionary of the specified word length will be read into some data structure (probably a list).
- Until the game is over, do the following:
  1. Print the number of remaining guesses.
  2. Print the list of letters already guessed. (Print this as a string of 26 characters, all blank except the guessed letters, which will be in the appropriate alphabetic ordered position of the string, all upper case).
  3. Print the current blanked-out version of the word (the word pattern composed of blank separated underscores and letters).
  4. if there was a fourth command line argument, and its value was '-s', print the number of words in the current family.
  5. Prompt the user to enter a letter. Make sure the input was a letter and had not already been guessed. Continue to prompt the user until he enters a different letter from all previous entries.
  6. Partition the current family of words into new word families based on the input letter.
  7. Make the current word family the word family with the most words.
  8. If the guessed letter is not in any word of the word family, subtract one (1) from the number of remaining guesses.

9. If the player has run out of guesses, pick a word from the word family and display it as the word the computer initially "chose". This ends the game.
10. If the player correctly guessed the word, congratulate him and end the game.

- Once the game has completed, end the program.

Advice

Think about how to solve this problem before writing code. Some things to watch out for:

1. Letter position matters as much as letter frequency. The two words REED and ELSE both have two E's, but because of the position of the E's in the words, they are in different families.
2. There are word length gaps in the dictionary. There is a word with length 29, but there is no word with length 27. Account for this in your solution.

Deliverables

This project is due by the end of the day Friday June 7.

You will submit your code (contained in a single file) to Blackboard.