Thicccque
November 25, 2018
Dr. Parham-Mocello

Assignment 5 Part 1

Orientation:

1. Would you prefer a common First Year Engineering Experience, where you are encouraged to explore disciplines, learn university skills, and declare a major after your first year, or enter directly into a major with discipline specific courses. Explain why or explain the pros and cons of both.

    - I think there are two kinds of students entering the field of engineering: those that have been in a club for years (robotics or coding or CTF, etc) and those who just know they like science or engineering things or that they just love to make things. These two kinds of students would benefit more from different first-year experiences, but both would greatly benefit from an exploration of university skills and interdisciplinary skills. Because of this, I think it would be awesome to have a first-year experience that is general and exploratory, but one that still allows for a 'focus.' Don't make 18 year old kids declare a major in engineering. Engineering is still such an infantile section of secondary education that they can't have put out their feelers. Where general engineering is the exploratory option currently in effect, I think a general 'focus' would be much less ambiguous feeling. For those that know what they want to study, they can have a focus on a specific field, such that they start eliminating pre-reqs in their desired major, but they still explore other fields and university skills to set them up for success. I think this would help to decrease major insecurity / switching later on in education, as people get a taste of engineering. Even if this didn't greatly influence this statistic, it would probably increase satisfaction with a second major, for those students that switch into another major, as they have an idea of what they enjoy studying. A specific major seems a little too much of a punch to the face, and, in my experience, the majors are completely abstracted, which just isn't how industry usually works. Interplay is essential and it is currently lacking.

2. Do you think all disciplines in the college of engineering should be made to take a programming course? Why or why not? How would your answer change for the whole university, instead of the college of engineering?

    - I think all disciplines in the college of engineering could greatly benefit from introductory programming education, at a minimum. I think this just because all engineers are going to be adjacent to programming and computer science problems and implementation. Not every engineer should need to be able to fully implement a solution, but I think it'd help immensely if every student could understand what's going on and debug solutions on their own, without completely relying on the assistance of computer science coworkers or compatriots. For these reasons, I think college of engineering students could all benefit from this education (especially in the case of MIME, because my group-mate just had no idea what was going on all term, and a basic understanding could really help), but I don't think this should be extended to the entire university, however. Although the modern world is continually moving further toward computer science, programming, and general computer literacy being a veritable necessity, at this point, not everyone needs to know how to code. Making programming classes a requirement for all students seems somewhat extreme. I think there could be a bacc core requirement that is satisfied by a programming course, but not that only programming courses should be able to fulfill it. Make programming an easily accessible (and highly recommended) possibility, so anyone could (and really should) take them, but don't actually require them to. Why make business major learn a skill they may never use as a future CEO, for example. That's my reasoning.

Computation:

1. Understanding the Problem / Problem Analysis:
    - Do you understand everything in the problem? List anything you do not fully understand, and make sure to ask a TA or instructor about anything you do not understand.

        Yes, I think I do understand everything in the problem.

    - What are the user inputs/requirements, functional requirements, etc.?
        - User requirements:
            - Choose whether to use a custom phrase or a canned one.
            - Guess a single letter at each iteration.
            - Dictate whether they're playing again or quitting the program.
        - Program requirements:
            - Store user input as a secret phrase.
            - Store number of strikes, starting at 0.
            - Convert phrase to a hidden string with underscores.
            - Update this hidden string.
            - Check if an input is a single letter / a full phrase of legal characters.
            - Check if a guess is correct, incorrect, or previously guessed already.
            - Check if user has lost or won.
            - Print out ASCII art of hanged man, that changes with value of strikes.

    - What assumptions are you are making?
        - I'm assuming that I should only handle single letters as inputs at each iteration, and that, to win, the user must guess every letter, not guess the whole phrase as an input.

    - What are all the tasks and subtasks in this problem? (Pseudo-code).
        - Pre-game setup
            - Establish hidden phrase
            - Establish strike = 0
            - Get hidden string

        - Main game loop
            - Get guess
            - Check validity and quality of the guess
            - Increment strike or update hidden string based on quality
            - Check whether user won, lost, or needs to play another round
            - Send them there.

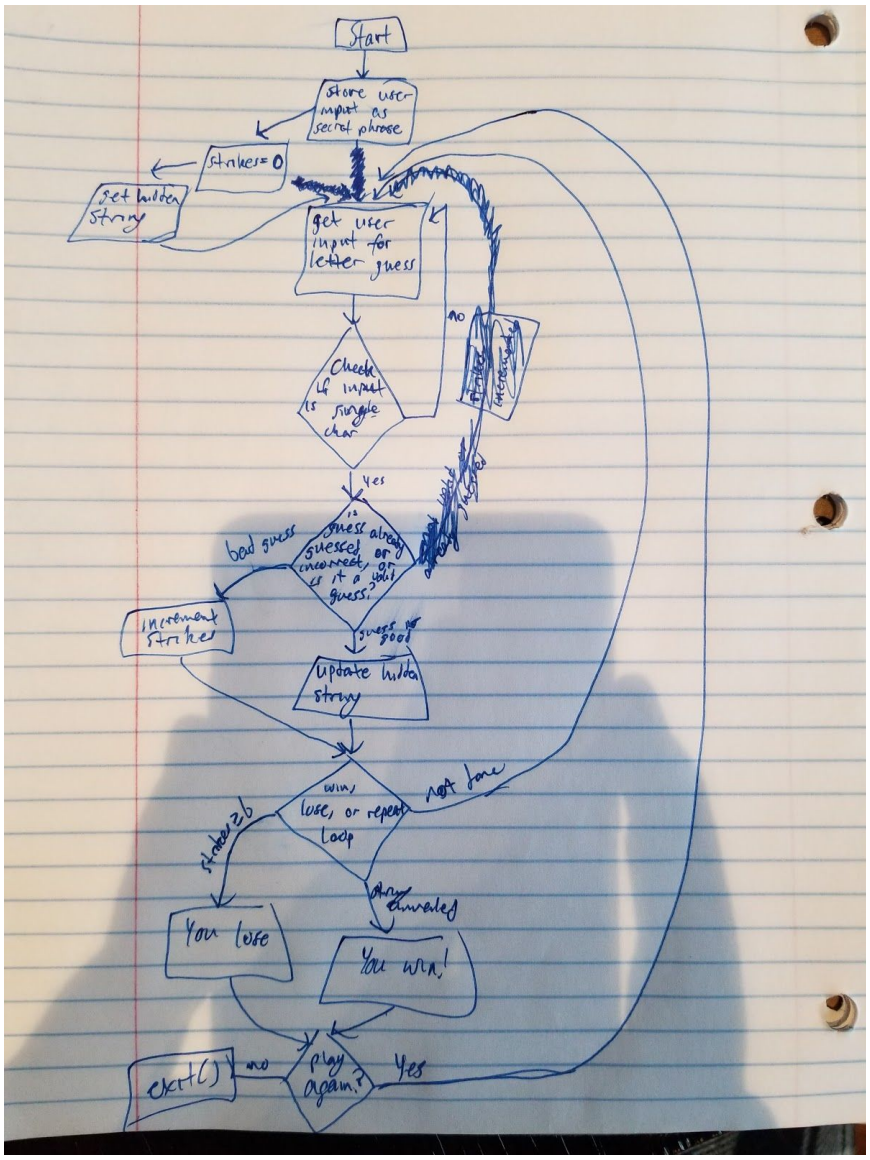
        - Check valid input for guess
            - Check if guess is length 1 and is valid char
            - Return appropriate value

        - Check valid input for phrase
            - Check if phrase is made of appropriate chars

        - Return appropriate value

    - Get hidden string
        - Go through list of characters in hidden phrase, adding either _ or the char depending on whether it has been guessed correctly.

- Program Design:
    - What does the overall big picture of this program look like?



    - What are the decisions that need to be made in this program?
        - Canned hidden phrase or user-inputed hidden phrase?
        - Ask for re-input (is it valid)?
        - Is a guess good (either add strike or reveal part of phrase)?
        - Did user win, lose, or should they repeat the main game loop?
        - Play again?

- Based on these tasks, what do the algorithm details in each task look like? What are the decisions you'll make within each task? How and where are you going to handle bad input? (Pseudocode)
  - Pre-game setup:
    - Get whether user chooses their own phrase or uses a random pre-generated phrase.
      - Get user input for hidden phrase.
      - Or choose a random phrase from list.
    - Establish 'int strikes = 0;'
    - Get hidden phrase with '_' representation.
    - Launch main game loop.

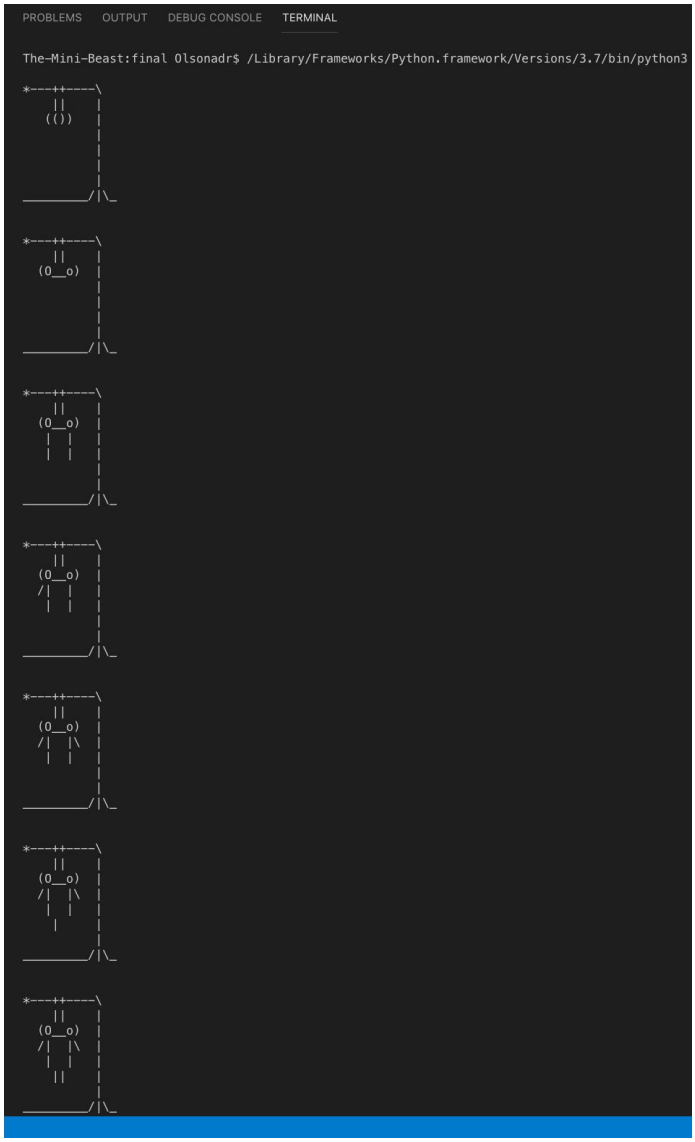
  - Check user guess validity:
    - Check if it is a string object.
    - Check if a string is a valid character ('a'-'z' U 'A'-'Z' U ' ').
    - Check if it is length one.
    - Return 0 for valid or an int above 0 for specific errors.

  - Check hidden phrase validity:
    - Check if it is a string object.
    - Check if it is made of valid characters ('a'-'z' U 'A'-'Z' U ' ').
    - Return 0 for valid or int above 0 for specific errors.

  - Main game loop:
    - In loop until broken:
      - In loop until broken
        - Print prior guesses and ascii
        - Get user guess
        - Check validity
        - If validity == 0:
          - Break
        - Else:
          - Print why it was invalid, continue
      - Check if guess is a character in the secret phrase
      - If it is a good guess:
        - Print that it is a good guess
        - Hidden_string = current hidden representation
      - Else:
        - Print that it is a bad guess
        - Strike += 1
      - If strike >= 6:
        - Print that user lost
        - break
      - Else if secret phrase = hidden representation:

        - Print that user won
        - Break
    - Else:
        - Continue
- Ask user if they want to play again
- If play again:
    - Run setup again
- Else:
    - exit()

Extra-credit:



Code attached.