Assignment 5: Software Design

In this assignment you will design a word game that may be used by a single player to practice for Scrabble. The detailed requirements for the application are provided below.

To create your design, you should follow the same approach that we present in the P3L2 lesson. That is, analyze the requirements to identify and refine **(1) classes, (2) attributes, (3) operations, and (4) relationships** in your design. Just to be completely clear, **your task is to design the system, not to implement it**. The requirements for the application are listed below, in the "Requirements" section. Please note that not every requirement will be fully and directly represented in your design. For instance, at this level of detail, you do not have to show any purely GUI specific classes, if they are only doing user display and input and not performing any significant business logic. Similarly, any database support layer may be left out, if it is purely doing persistence tasks on **data fully represented in the diagrammed layer**.

Your design should be expressed using a UML class diagram, and the level of detail of the design should be analogous to the level of detail we used throughout the **whole** P3L2 lesson (i.e., do not limit your design to only the elements focused on in the final video, but consider the entire process). Specifically, **you must provide enough details for the design to be self contained and for us to be able to assess whether the design suitably realizes all system requirements**.

To help with this, you must also provide a "design information" document, in which you concisely describe, **for each of the requirements listed below**, how that requirement is either realized in your design, or why it does not directly affect the design and is not shown. **Copy the list of requirements, and add your explanation for each one of them**. For example, using some partial requirements for a cash register application:

…

2. After starting the cash register, the cashier enters her name, and the total amount of money available in the till.

`To realize this requirement, I added a 'currentCashier' to the register class to track the signed in cashier, and a float 'till' to represent the money in the till. These values are entered by the startup() method, after prompts are handled within the GUI.`

…<additional requirements reflected in example UML diagram>...

16. The User Interface (UI) must be intuitive and responsive.

`This is not represented in my design, as it will be handled entirely within the GUI implementation.`

This explanation should be clear enough to allow us to follow the rationale behind your design and **how it will fulfill each specified requirement, including any that are not directly depicted in your class diagram.** You can also provide in the document additional information about your design, such as assumptions or rationale for some design decisions. Use the document to review your design and ensure you have included everything necessary for your application to fill the list of requirements.

You can use any UML tool you prefer, but do not hand draw your design. If you are not familiar with any specific tool, we recommend that you ask on Piazza for suggestions. There are usually some student discussions and recommendations for that on Piazza.

Requirements

- When the application is started, the player may choose to (1) Play a word game, (2) View statistics, or (3) Adjust the game settings.
- When choosing to *adjust the game settings*, the player (1) may choose for the game to end after a certain *maximum number of turns* and (2) may adjust the *number* of and the *letter points* for each *letter* available in the *pool of letters,* starting with the default matching the English Scrabble distribution (12 E's worth 1 point each, 4 D's worth 2 points each, etc).
- When choosing to *play a word game,* a player will:
    - Be shown a 'board' consisting of 4 letters drawn from the pool of available letters.
    - Be shown a 'rack' of 7 letters also drawn from the pool of available letters.
    - On each turn, up to the *maximum number of turns* either*:*
        - Enter a word made up of one or more letters from the player's rack of letters, and one from the board. The word must contain only valid letters and may not be repeated within a single game, but does not need to be checked against a dictionary (for simplicity, we will assume the player enters only real words that are spelled correctly).
          or
        - Swap 1-7 letters from their rack with letters from the pool of letters. This is the only time letters are returned to the pool during a game.
    - After a word is played, that letter on the board will be replaced by a different random letter from the word that was just played. Example: If 'c' is on the board, and 'j','a','k','e','t','s' are part of the rack, then the player may enter 'jackets' as a word, and the 'c' will be randomly replaced by the 'j','a','k','e','t', or 's' for the next turn on the board.
    - After a word is played, the tiles used from the rack are replaced from the pool of letters.
    - After a word is played, the player's score will increase by the total number of points for the letters in the word, including the letter used from the board. (So 'jackets', if using default values, would score 20 points.)
    - If the *pool of letters* is empty and the rack cannot be refilled, the player will score an additional 10 points.

    - When the *maximum number of turns* has been played, or the *pool of letters* is empty and the rack cannot be refilled, the game will end, and the final score will be displayed before returning to the first menu.
- A player may choose to leave a game in progress at any time. Selecting to play a game from the menu should then return to the game in progress.
- When choosing to view statistics, the player may view (1) *game score statistics*, (2) *letter statistics* or (3) *the word bank*.
- For game score statistics, the player will view the list of scores, in descending order by final game score, displaying:
    - The final game score
    - The number of turns in that game
    - The average score per turn

  The player may select any of the game scores to view the settings for that game's *maximum number of turns, letter distribution,* and *letter points*.
- For letter statistics, the player will view the list of letters, in ascending order by number of times played, displaying:
    - The total number of times that letter has been played in a word
    - The total number of times that letter has been traded back into the pool
    - The percentage of times that the letter is used in a word, out of the total number of times it has been drawn
- For the word bank, the player will view the list of words used, starting from the most recently played, displaying:
        - The word
        - The number of times the word has been played
- The user interface must be intuitive and responsive.
- The performance of the game should be such that students do not experience any considerable lag between their actions and the response of the application.
- For simplicity, you may assume there is a *single system* running the application.

Submission Instructions

To submit your assignment, you should do the following:

- Create a directory called `Assignment5` in the usual **personal GitHub repository we assigned to you**. This is an **individual assignment**; do **not** use your new team repositories.
- Save your UML class diagram in the `Assignment5` directory as a PDF file named "`design.pdf`". **Important:** Make sure to open your PDF after generating it and double check it, as we have had a number of cases of students not realizing that the conversion to PDF had not worked as expected.
- Save the "design information" document in the same directory, in markdown format, and name it "**`design-information.md`**".
- Commit and push your file(s) to your remote repository.
- Submit the commit ID for your solution on Canvas.