Project 01: jWordle

COSC 102 - Spring '22

For your first project, you will complete a Java version of *Wordle*, the latest popular word game. The front-end, i.e., the graphical view is provided, and you will implement the back-end, that is the game logic. The goal for this project is to practice Java syntax using the procedural programming paradigm you studied in COSC101.

1 Overview

Wordle is a web-based word game developed by Josh Wordle. Players must guess a five-letter "secret word"—every day the same one for everyone so we share our grid pattern.

1.1 Basic Rules

The game plays as follows:

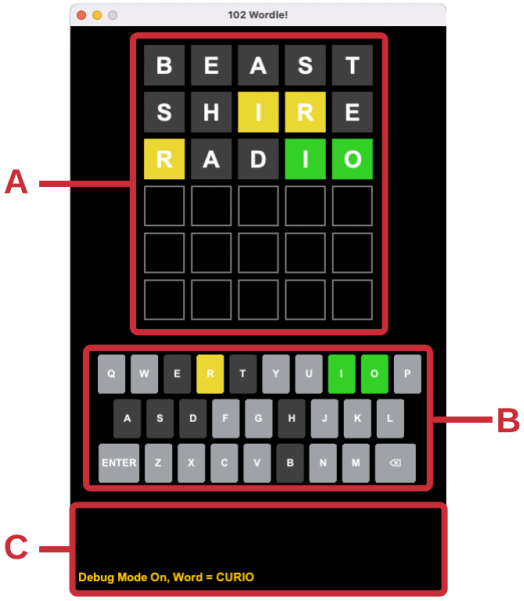
- An English five-letter word is selected at random: **the secret**.
- Players have *six* attempts to guess the secret.
- After each five-letter English word entered by the player, feedback regarding each letter is provided in the form of a colored tile. Specifically,
  - if the letter is in the secret word and in the correct spot, it is colored `green`
  - if the letter is in the secret word but not in the correct spot, it is colored `yellow`
  - if the letter is not in the secret word, it is colored `gray`

Note that the secret word and all of the player's entered attempt must be actual, English dictionary words (*ex:* `"ZXVQR"` is invalid and not an accepted attempt). Words may have duplicate letters.

1.2 The GUI

The Java implementation of the game you will complete has a *Graphical User Interface* (GUI). The program presents a graphical front-end in an interactive window such that it allows and responds to player input.

An example of the GUI with the logic fully implemented is described below:



- **A:** the *game board* where the player's guesses and subsequent feedback are displayed.
- **B:** the *graphical keyboard interface* which users can click to enter guesses. Note that not only the game board but the keys on the keyboard change color to reflect the game board. The bottom left and right buttons are mapped to `return` and `backspace` respectively.
- **C:** the *dialogue area* where messages are displayed to the user. If *debug mode* is enabled, the debug text is displayed here (more on this below).

1.3 Game Flow Breakdown

Using the GUI screenshot presented, we can breakdown the game logic as follow using the rules of *Section 1.1*:

- As indicated by the *debug text*, the secret word for this game is **`CURIO`**.
- The player's first attempt is **`BEAST`**. Upon pressing *enter* on this five-letter English word, each letter match is evaluated.

  None of the guess' letters, `'B'`, `'E'`, `'A'`, `'S'` or `'T'`, appear in `CURIO`, thus all the cells of the row along with their respective keys on the keyboard interface are colored **`gray`**.
- The second attempt, **`SHIRE`**, reveals that `'I'` and `'R'` appear in `CURIO`, but are not in the correct position. Thus, these cells along with the `'I'` and `'R'` keys on the keyboard are colored **`yellow`**. The row's remaining cells are colored **`gray`** along with the `'H'` key on the keyboard.
- For the third attempt, **`RADIO`**:
  - `'I'` and `'O'` are in the proper position, thus their cells are colored **`green`**, and on the keyboard.
  - `'R'` is still in a wrong place, and its cell is colored **`yellow`**, and its keyboard key remains **`yellow`**.
  - The remaining two cells and the `D` keyboard key are turned **`gray`**.

1.4 Data file seeding jWordle

At game launch the program must read in a file containing a list of five-letter words: this list represents the valid English words. From it a secret word is chosen at random to which each valid attempt will be compared to.

The word data file must be in plain text, using the `.txt` extension for example. It contains one word per line, and may contain any number of words. Our provided **`englishWords5.txt`** contains **5758** words.

2 Provided Code

The implementation is separated in **two** `.java` files: **`WordleView.java`** (the view we provide) and **`WordleLogic.java`** (the model you implement). Details regarding these two files are provided in the sections below.

2.1 `WordleView`

`WordleView.java` creates and maintains the GUI of the game, it draws the game window and processes interface events, mainly player's key input on the virtual and physical keyboards. **You do not need to modify this file**.

You have to **call** several of its functions, specifically the **six** functions listed below:

1. `public static void setCellLetter(int cellRow, int cellCol, char letter)`
2. `public static void setCellColor(int cellRow, int cellCol, Color newColor)`
3. `public static Color getKeyboardColor(char letter)`
4. `public static void setKeyboardColor(char letter, Color newColor)`
5. `public static void gameOver(boolean didPlayerWin)`

6. `public static void wiggleRow(int row)`

They are located at the top of `WordleView.java`, and are designated with a comment. You will need to read each function's docstring to determine their purpose and how to use them in your implementation.

2.2 Debug Mode

The provided code can be ran in *debug mode*. When debug mode is enabled, the secret word is displayed in the bottom left-hand corner of the window:

Debug mode is toggled on/off via a `boolean` constant named **`DEBUG_MODE`** located at the top of `WordleLogic.java`.

2.3 `WordleLogic`

The **`WordleLogic.java`** file will contain all of the back-end logic for playing the game: it keeps track of the game state, processing each grid line, evaluating each attempt entered by the player against the valid words and the secret. **All of your code you write must be in `WordleLogic.java`.**

Six functions are declared, which you must complete :

1. **`public static void warmUp():`** This is a function meant for you to play around with the functions in **`WordleView.java`** that were provided to you. You must use those functions provided to you in your code to implement the game. A common practice among expert programmers when using a new library or a code that was implemented by someone else is to first understand how to use them, that is how to call these functions and what is the effect of using these functions. This is exactly what you need to do in this function. In the next section, we describe exactly what functionality to implement in this function that will entail exploring the functions provided to you in **`WordleView.java`**.
2. **`public static String init() throws FileNotFoundException:`** This function is meant to be called at the beginning of the game. It is meant to load up the words and pick a word for the player to guess.
3. **`public static void inputLetter(char key):`** This function is called each time a player clicks on a letter. It should display the letter in the next available cell of the current row.
4. **`public static void deleteLetter():`** This function is meant to be called each time a player clicks on the backspace key. It should remove the last letter in the row.
5. **`public static void checkLetters():`** This function is meant to be called each time a player clicks on the ENTER key. It should check the word entered by the player against the secret word, and then color the letters green, yellow or gray as described in the rules of the game.
6. **`public static void main(String[] args) throws FileNotFoundException:`** The `main` method runs the game initializing it and launching its graphical view. Note that `main` can also run a **`WARM_UP`** version (see *Section 3.1* below).

You are expected to define and use more helper methods as you design thoughtful, well designed, and modular code.

Additionally, there are **constants** declared at the top of the class. Use **named constants** in place of magic numbers so that your implementation is easy to read and to upgrade. Review the ones declared with their respective comments to understand their utility and use them throughout.

3 Your Task

Your task is to implement the back-end logic of the game, which includes selecting the secret word first, and then handling the view calls as each letter is entered or delete and when a word needs to be evaluated as valid five-letter word and if so each of its letters match against the secret. Accordingly your logic code calls tiles and keyboard keys

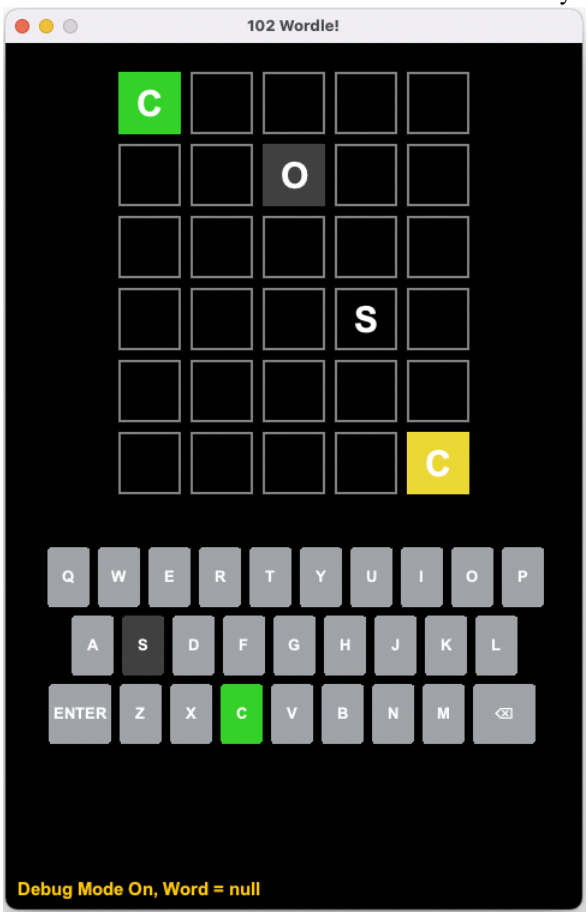
coloring functions of the view and will determine when the game over is reached, indicating a win or a lose to be displayed by the view.

The sections below will guide you in achieving this. It is important that you **read and implement each section in order** – do not move on to a new section until you have completed all of the previous ones.

3.1 Warm-Up

Before you begin implementing the actual game logic, you must complete the following two steps to acclimate yourself to the provided code base. For each you write few lines, which mainly call the functions listed in *Section 2.2*.

1. Complete the `warmUp` function so that the game view looks exactly like the screenshot below: four letter tiles on the game board some of which colored and hence as such on the virtual keyboard.



2. Write a few lines in the `inputLetter` function so that whenever the character `'W'` is sent to it the fourth row from the top on the game board (containing the `'S'`) "wiggles", meaning it quickly shakes left and right.

Once you have successfully completed this Warm-Up, **toggle it to off** by setting its constant named `WARM_UP` to `false` at the top of `WordleLogic.java` and comment out the lines you added in `inputLetter()`.

3.2 Basic Functionality

Next you will implement the basic game functionality, so that it will be playable from start to finish with some concessions. Details of what should and should not be included are outlined below, **read both lists** before starting.

Your basic implementation consists of the following.

- Read the data file defined by the `FILENAME` constant and store its words so as to select a random secret word from it.

- Handle player keyboard characters and either draw the letters on the current row of the game board or delete them (*Note: if the line is full, additional letter are ignored*).
- Evaluate the player's attempt by completing the function `checkLetters` so that game board tiles and keyboard keys are colored appropriately.
- End the game when the player found the secret word or runs out of attempt, displaying appropriate game over text.

Your basic implementation does **not** yet need to:

- prevent invalid words
- properly handle words with duplicate letters

3.3 Advanced Features and Logic

Your final code must implement the below advanced features and logic. **Do not attempt to implement these components until you have completed the *Basic Functionality* as described above!**

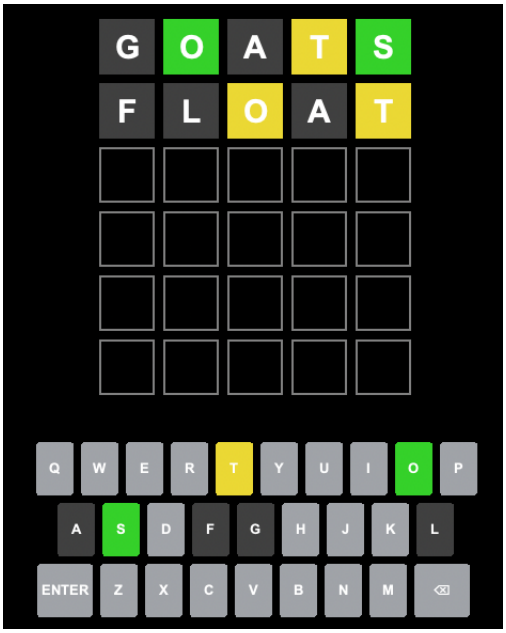
3.3.1 Invalid Inputs

If a player attempts is an invalid word, the row displaying it should "wiggle" back and forth to indicate that there's a problem with the input. The letters themselves **should not** be deleted or changed in any way.

There are **two scenarios** when an attempt is invalid – think about both and handle them in turn!

3.3.2 Keyboard Coloring Priority

Your coloring of virtual keyboard must follow certain prioritization rules. Review the scenario below:

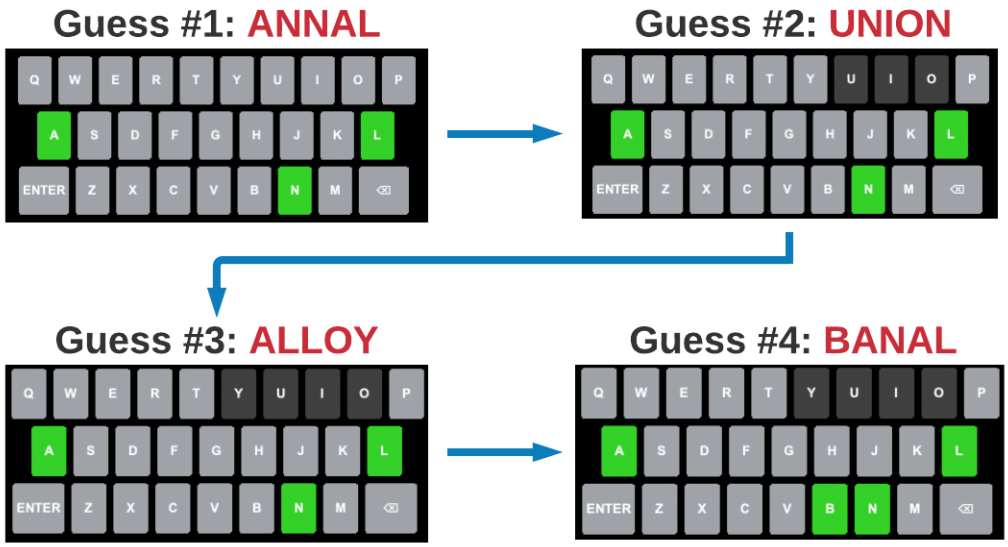


On the first guess, `O` is in the correct position and thus highlighted `green`. On the second guess, `O`, is in an incorrect position and thus highlighted `yellow`. Given this scenario, deduce the prioritization of the keyboard key coloring.

3.3.3 Duplicate Letter Handling

The web-based Wordle game has particular rules when evaluating words/guesses with duplicate letters, which your implementation must match. Review the screenshot below, showing a series of guesses where the secret word is **`BANAL`**:

Furthermore, the following pictures showcase the progression of the keyboard colors over the above four attempts:



From the above screenshots, you must reverse-engineer how the duplicate letter tiles and keyboard key coloring works. Implement this logic into your game, ensuring it matches the logic demonstrated above.

4 Submission

This project is due at **11:00P.M.** on the **Monday, February, 21, 2022**.

Upload **only** your `WordleLogic.java` file on your Moodle COSC102 lecture.