Project 01: JWordle

COSC 102 - Fall ’22

Goal: In this project, you will complete a Java implementation of the popular word game *Wordle*. In doing so, you’ll better acclimate yourself to Java syntax and reinforce the concepts of programmatic design.

1 Overview

Wordle is a web-based word game developed by Josh Wardle. The game was purchased by the *New York Times* in 2022 and can be played for free on their website. In the game, players must guess a five-letter "secret word" while receiving incremental feedback. Furthermore, there is only one secret word each day which all players share.

1.1 Basic Rules

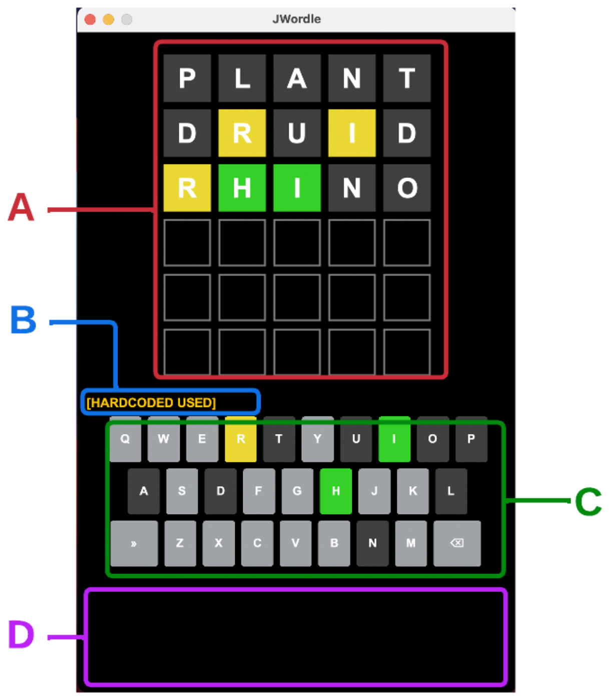
The game plays as follows:

- Players have *six* turns to guess a randomly chosen five-letter word.
- After a guess, players are given feedback regarding the guess’ individual letters. Specifically, for each letter:
  - 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 **`dark gray`**
- The secret word and all of the player’s guesses must be actual, English dictionary words (*ex:* `"ZXVQR"` would not be a valid guess). Guesses and secret words may have duplicate letters.

1.2 The GUI

The Java implementation of the game you will complete makes use of a *Graphical User Interface* (GUI). This means the program runs in a separate window with a graphical abstraction of the game being played, and allows the user to click buttons within the window to enter their words (though they may use the physical keyboard as well).

All of the GUI code in this project is **already implemented for you**. A breakdown of the GUI is described below:



- **A:** the *game grid* where the player's guesses and subsequent feedback are displayed. The grid's dimensions are **six rows** (the maximum player guesses) by **five columns** (number of letters in the secret word).
- **B:** the *debug toggle notifications* which display any activated *debug toggles*. These toggles are optional parameters which can be enabled/disabled to help you in your implementation (more on these below).
- **C:** the *graphical keyboard interface* which users can click to enter guesses. Keys on the keyboard change color to reflect the game board. The bottom left and right buttons are mapped to `enter` and `backspace` respectively.
- **D:** the *dialogue area* where messages are displayed to the user.

1.3 Game Flow Breakdown

Using the GUI screenshot above, we can see an example of the game logic outlined in *Section 1.1*:

- In this instance the secret word is **`SHIRE`**.
- The player's first guess, **`PLANT`**, gets entered into the first row. Upon pressing *enter* the word is evaluated.

  None of the guess' letters, `P`, `L`, `A`, `N` or `T`, appear in `SHIRE`, thus all the cells of the row along with their respective keys on the keyboard interface are colored **`gray`**.
- For the player's second guess, **`DRUID`**, the `R` and `I` characters appear in `SHIRE`, but are not in the correct position. Thus, these cells along with the `R` and `I` keys on the keyboard are colored **`yellow`**. The row's remaining cells are colored **`gray`** along with the `U` key on the keyboard.
- for the third guess, **`RHINO`**:
  - the `H` and `I` characters are in the proper position, thus their cells are colored **`green`**. The `H` key on the keyboard is colored **`green`**, and the `I` key changes from **`yellow`** to **`green`**.
  - the `R` is still in a wrong place, and its cell is colored **`yellow`**. The `R` keyboard key remains **`yellow`**.
  - the remaining two cells and the `O` keyboard key are turned **`gray`**

1.4 Word Data File

The game reads in a word data file containing a collection of five-letter words when it is ran. The program then uses this data file to pick a secret word (chosen at random) and to validate the player's guesses.

The word data file must be a `.txt` text file. Additionally, it must be formatted to contain only one word per line, but may contain any number of words, so long as they all contain only five letters. An example is provided to you, named **`words.txt`**, containing **5758** words.

2 Provided Code

Provided to you are **three** `.java` files, though **you only need to add code to one of these**. Details are below:

2.1 `GameLauncher`

The **`GameLauncher.java`** file contains the `main` method used to launch the game.

Additionally, at the top of this file are three `final boolean` variables representing various **debug toggles**. These toggles change the game functionality to make the game logic more transparent and/or make testing your implementation easier. When a toggle is enabled, a notification is displayed in the game window (area **B** in the diagram above).

Read the comments for each variable to understand the effect that each toggle has. You **do not need to modify** this file aside from changing the values of the debug toggle variables.

2.2 `JWordleLogic`

The **`JWordleLogic.java`** file contains all of the back-end logic for reacting to user input, evaluating guesses, and controlling the state of the game. **All of your code for this project will go in this file**.

This class contains **two functions** which you must complete (though you will **add more** helper functions):

- **`public static String initGame():`** this function gets called **once** when the game is very first ran, and performs any initialization needed for your game. This function must **return** the secret word as a `char` **array**.

- **`public static void keyPressed(String key):`** this function gets called automatically anytime the user presses a valid key (either by pushing a key on their physical keyboard, or clicking a key on the graphical keyboard interface). The argument passed in is a `char` representation of the key pressed.

Additionally, there are numerous **final variables** declared at the top of `JWordleLogic` that are very important in your implementation. Review these variables and their comments to get an understanding of their utility.

Lastly, you are **not allowed** to add any more non-final global variables to `JWordleLogic.java`. However, you are welcome (and encouraged) to add additional `final` variables.

2.3 `JWordleGUI`

The **`JWordleGUI.java`** file contains all of the GUI code for the game, including drawing the game window and recognizing user input. **You cannot modify this file**.

That said, you will need to *call* several of the functions it implements. There are **eight** function you may need to call in this file; their names are listed below:

1. `public static char[] getSecretWord()`
2. `public static void setGridLetter(int row, int col, char letter)`
3. `public static char getGridLetter(int row, int col)`
4. `public static Color getGridColor(int row, int col)`
5. `public static void setGridColor(int row, int col, Color newColor)`
6. `public static Color getKeyColor(char key)`
7. `public static void setKeyColor(char key, Color newColor)`
8. `public static void endGame(boolean didPlayerWin)`

All eight functions are located near the top of the file, and are designated with a comment. You will need to read each function's comments to determine their purpose and how they can be used by your implementation.

3 Your Task

Your task is to implement the back-end game logic for the JWordle game, including responding to player input, evaluating guesses and coloring cells/keys appropriately, and determining when a game over state has been reached.

Your implementation will be broken up into **two milestones**; you will submit your code after completion of each milestone and receive feedback from your instructor. The below sections will guide you through both milestones.

It is important you **implement each section in order**, but should **read this entire document** before you start coding.

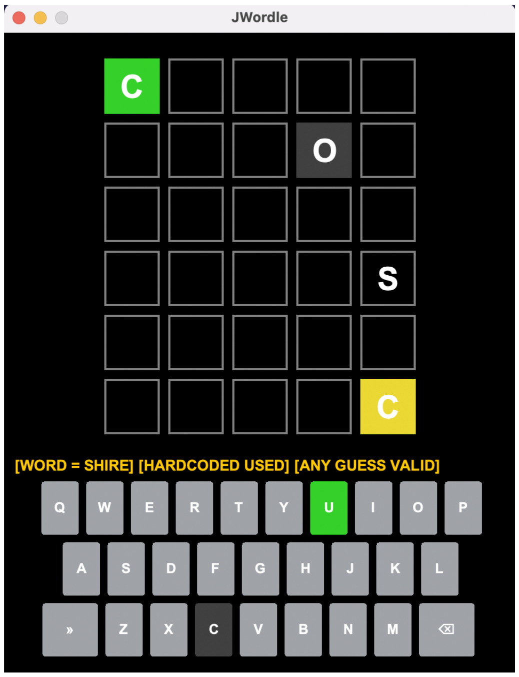
3.1 Milestone #1 (Warmup and Basic Functionality)

The first milestone will acclimate you to the code base and have you implement the basic game functionality. At the end of Milestone #1, the goal is to have your game playable using a preset, **hard-coded word**, albeit without some of the advanced logic (things like reacting to duplicate letters, validating guesses, etc).

3.1.1 Warmup

To get started, complete the following:

1. When you first attempt to launch the game, JWordle will **crash with an exception** which you must fix. Read the exception, trace the code, and review the relevant comments to help identifiy this issue. Remember, the goal of the first milestone is to get the game working using a hard-coded word.
2. When the previous step is complete, the game should launch a window with an empty game grid that does not react to user input. Add code to `JWordleLogic` so that, when the game is first launched, the game's grid and keyboard look exactly like below:



[WORD = SHIRE] [HARDCODED USED] [ANY GUESS VALID]

3. Make it so that whenever the user presses "`W`", either via their physical keyboard or the graphical keyboard interface, the fourth row from the top (containing the "`S`") "wiggles" (meaning it quickly shakes left and right).

3.1.2 Basic Functionality

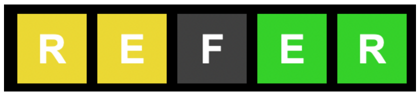
Next you will implement the basic game functionality, so the game will be playable from start to finish with some concessions. Before continuing, **comment out** the code you implemented to complete warmup *steps 2* and *3* above. Your basic implementation must:

- use a preset, hard-coded word as the secret word.
- react to player keyboard input and draw the characters on the current row of the game board. This includes the backspace key to delete characters (*Note: if the row is full, additional letter key presses are ignored*).
- evaluate the player's guess when the *enter* key is pressed, coloring the grid cells and keyboard keys appropriately. Keyboard key colors should prioritize correctly (ex: a green colored key shouldn't change to yellow, etc).
- end the game when the player guesses the word or runs out of guesses, displaying appropriate game over text.

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

- read words from a file and pick a random secret word
- prevent invalid inputs (i.e. guesses which aren't actual English words)
- properly handle words or guesses with duplicate letters; it can simply highlight letters green, yellow, or gray depending on if they're in the word and/or in the correct place.

  For example, given a secret word of **`EMBER`**, a guess of **`REFER`** would color like so:



3.2 Milestone #2 (Advanced Features and Logic)

Your final code must implement the advanced features and logic described below.

3.2.1 Secret Word Randomization

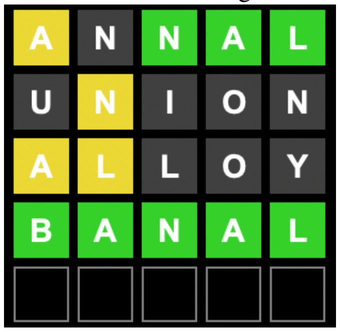
Your JWordle program must randomly select a word out of the provided `words.txt` as its secret word.

3.2.2 Input Validation

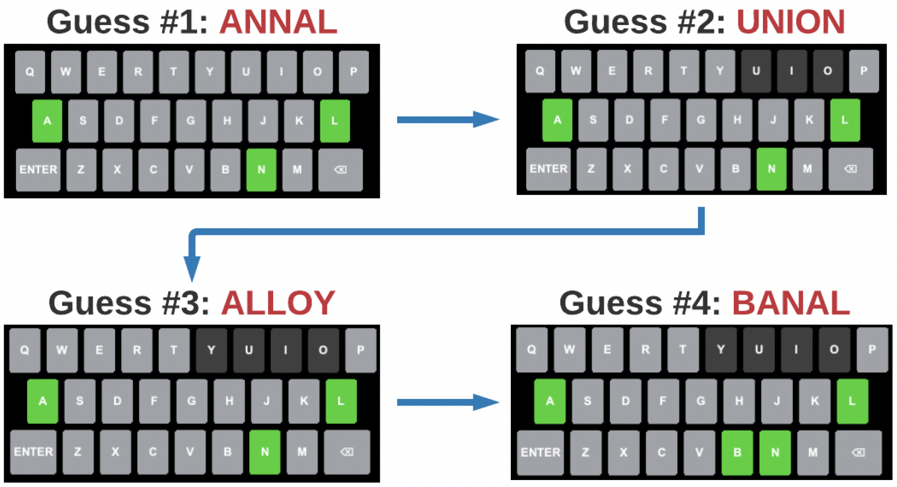
If a player attempts to enter an invalid guess, all of the cells in the active row should "wiggle" back and forth to indicate that there's a problem with the input. The letters themselves **should not** be deleted or changed in any way. There are **two scenarios** where a guess would be invalid – think about it!

3.2.3 Duplicate Letter Handling

The web-based Wordle game has particular rules when evaluating words/guesses with duplicate letters, which your implementation will 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 guesses:



From the previous screenshots, you will be able to reverse-engineer how the duplicate letter cell/keyboard coloring works. Implement this functionality into your game, ensuring it matches the logic demonstrated above.

4 Tips

Below are a few tips to help you in your implementation:

- Though your `JWordleLogic.java` contains only two functions, it is expected that you will add **many more** helper functions in the design of your implementation. You will be graded on the quality of your design; having long, bloated, difficult to read and unintuitive functions will adversely effect your grade!

  Remember your **SOFA** principles (if you haven't covered this in class yet, you will soon!) – keep your functions short and singularly focused. Consider helper functions for small, generic tasks that you need to repeat multiple times in different contexts (ex: searching an array for a value).
- Work and test incrementally. Run your code often. Focus on a single feature or functionality at a time, and test it exhaustively before moving on.
- Make extensive use of your debug toggles; they will be very helpful in your testing and debugging processes.
- Regarding integrating the debug toggles into the JWordle code:
  - one of the toggles is **implemented for you** in `JWordleGUI`
  - one of the toggles you will integrate as part of your **Milestone #1** code
  - one of the toggles you will integrate as part of your **Milestone #2** code
- Lastly, though `String`s can be useful here, remember that they have much greater memory/performance overhead than the primitive `char` type. If you can use a `char` instead of a `String`, you should!

5 Submission

You will submit your code after the completion of each milestone to the appropriate link on your course's Moodle page. For each milestone, upload **only your `JWordleLogic.java`** – do not submit any other code or your `words.txt`.

The milestones are due as follows:

- **Milestone #1:** due **Monday, September 19th** at **10:00PM**. You will additionally meet with your instructor to discuss the design and implementation of your Milestone #1 code – details on this to follow.
- **Milestone #2:** due **Friday, September 30th** at **3:00PM**.