Lab 6: Event Handling and Thread

Instruction

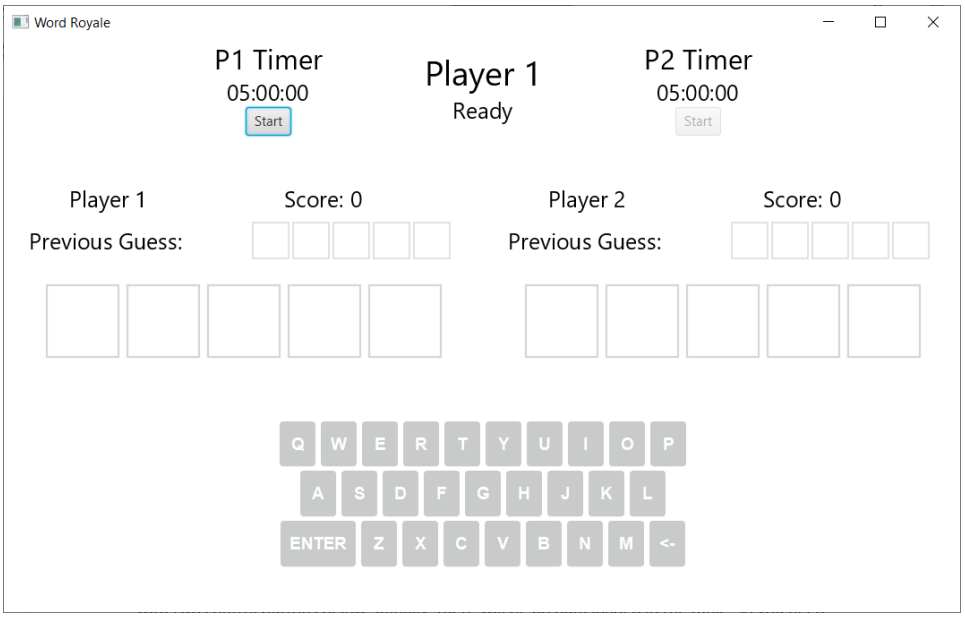
1. Click the provided link on CourseVille to create your own repository.
2. Open Eclipse and then “File > new > Java Project” and set project name in this format **2110215_Lab6_2021_2_{ID}_{FIRSTNAME}**
    - Example: **2110215_Lab6_2021_2_6131234521_Josh**.
3. Initialize git in your project directory
    - **Add .gitignore.**
    - Commit and push initial codes to your GitHub repository.
4. Implement all the classes and methods following the details given in the problem statement file which you can download from CourseVille.
    - **The provided files contain one folder:** `src` **make sure to add it into your project.**
    - You should create commits with meaningful messages when you finish each part of your program.
    - Don’t wait until you finish all features to create a commit.
5. Export your project into a jar file called **Lab6_2021_2_{ID}** and place it at the root directory of your project.
    - Example: **Lab6_2021_2_6131234521.jar**
6. Push all other commits to your GitHub repository

1. Problem Statement : WORD royaLE

This game is based on a famous daily word-based puzzle game “Wordle” by Josh Wardle, modified to fit 2 Player format. Each player takes turn guessing their own Word under the time limit. The score can be obtained by getting the correct guess at the word.

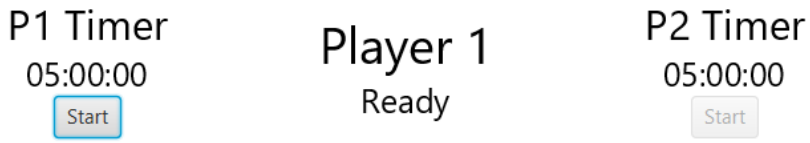
If the time runs out, the game ends. The player who lets the time runs out get -1 score penalty as well.

1.1 Overview



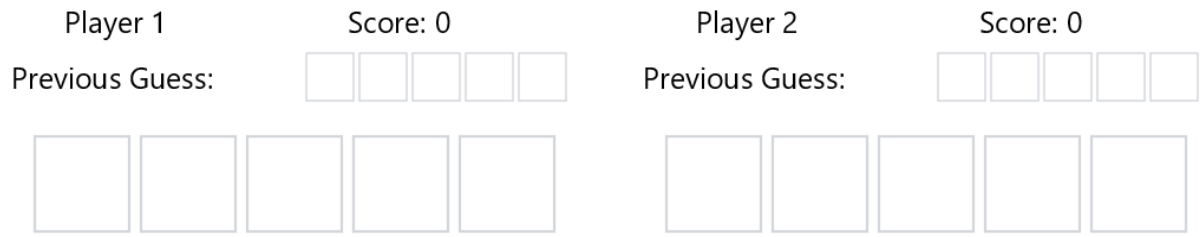
This application can be separated into 3 different sections, each section built from one or more pane.

The first section is the Timer and Turns detail

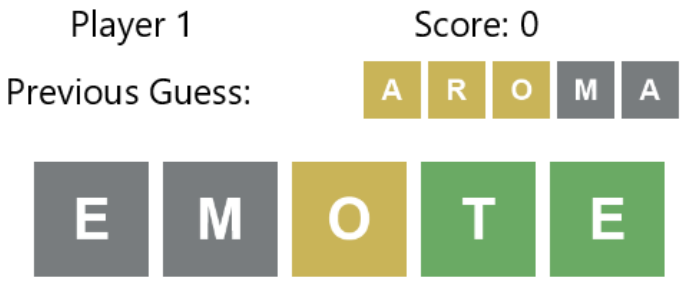


The player can click on the respective “Start” button to start their turn, and the status text will change accordingly, as well as enabling the keyboard and handle other respective logics.

The second section is the Word Canvas



Once the player presses the button on the keypad, the letter will be filled accordingly. After the player submitted the word, the main panel slowly flips to reveal the letter status. (Each letter will gradually start the flipping animation independently without waiting for the previous one to finish)



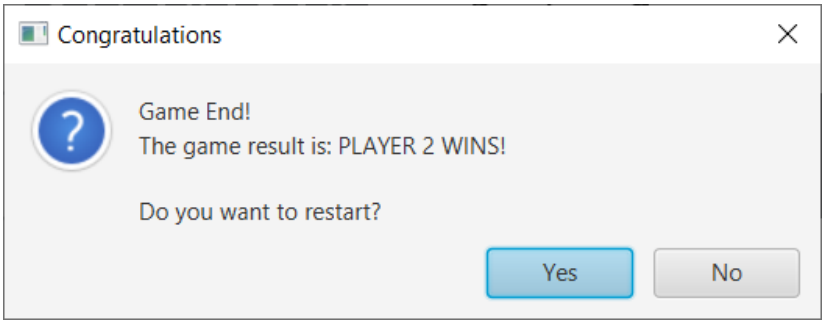
If the box is green, it means that the letter is in the answer and is in the correct position

If the box is yellow, it means that the letter is in the answer, but is in the wrong position

If the box is gray, it means that the letter is not in the answer at all.

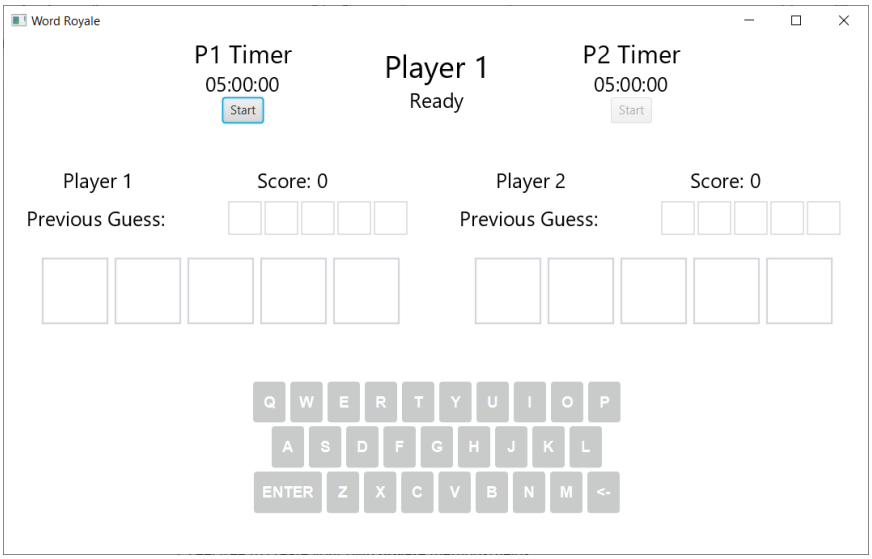
The “Previous Guess” box updates at the beginning of each respective player’s turn.

Once the game ends, the player who lose the time got their score deducted by one, then this dialogue box should be displayed to denote the winner.



The player can either choose to restart the game by pressing “Yes”, or close the program by pressing “No”.

2. Implementation Details:



Most of the Interface structures and game logics have been provided. You only need to implement the functionality of the Program. Those spots are marked with **/*FILL CODE*/** or **/*FIX CODE*/**

NOTE:

1. The interface of the program **must be responsive all the time**. You can interact with the game while the timer is running, and the flipping animation can be seen without freezing.
2. There should never be any exception raised if the program is implemented correctly.
3. There is a possibility to implement the entire program without using thread (Eg. AnimationTimer). You will get the score, **but you will not get any point from thread part.**
4. Only **relevant** methods/fields/classes will be shown below. (There are a lot more methods/fields/classes, but you do not have to touch them to finish the assignment)
5. Feel free to create your own **private** methods/fields.

** Noted that Access Modifier Notations can be listed below*

+ (public), # (protected), - (private), underlined (static), ALL_CAPS (final)

2.1 package `component`

2.1.1 Class `KeyButton` `extends Button`

This class is a component that is a single keypad button. It contains everything related to the keypad button itself, such as the appearance. You do not have to modify this class.

2.1.2 Class `KeyboardPane` `extends VBox`

This class is a component that contains the pane with the Keypad to input the word.

2.1.2.1 Fields

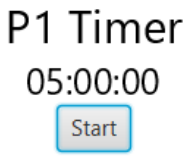
- Hashtable<String, KeyButton> btnDict	A hash table that maps from a String denoting the key, and the KeyButton object itself.
- String[][] keyArray	An array of array of String denotes each keyboard row.

2.1.2.2 Constructor

+ KeyboardPane()	Set up the KeyboardPane. **/*FILL CODE*/** Clicking each respective KeyButton should trigger a static method addLetterToCurrentPlayer(String s) from the class GameLogic.

2.1.3 Class `TimerPane` `extends VBox`

This class is a component that contains the respective Player label, a Timer, and a button that starts the Timer countdown.



2.1.3.1 Fields

- Button button	The button with the text “Start”

2.1.3.2 Constructor

+ TimerPane(int pl)	Initialize all fields and layout for the pane of the player pl+1 (0 for Player 1, 1 for Player 2) **/*FILL CODE*/** Clicking the “Start” button should calls the static method beginTurns(int pl) from GameLogic.

2.1.4 Class `WordCanvas` `extends Canvas`

This class is a component that contains the word box itself.

2.3.1.3 Methods

+ void flipLetter(int index, Status newStatus)	Plays the flip animation of the specified letter, after halfway the letter change its coloring to match the specified Status.

	/*FIX CODE*/ The following part of this code will freeze the program when run. Please turn this into a separate thread instead. **CAUTION:** Some part of this method contains UI update, which can cause an error if running in a different thread. You need to alter the code to make it works, too! *Please see the comment in the file for more information.*
+ flipWord(Status[] newStatus)	Plays the flip animation of the entire word, this method calls the method flipLetter above. **/*FIX CODE*/** The following part of this code will freeze the program when run. Please turn this into a separate thread instead. **CAUTION:** Some part of this method contains UI update, which can cause an error if running in a different thread. You need to alter the code to make it works, too! *Please see the comment in the file for more information.*

2.1 package `logic`

2.1.1 Class `GameLogic`

This class is the main logic of the game. It contains many methods related to the game state.

2.1.2.2 Methods

+ void startCountDownTimer(int pl)	Starts the countdown timer **/*FILL CODE*/** The following code will make the timer starts counting down. But it will not work if got called by the main application thread.

+ void runCountDownTimer(int pl) throws InterruptedException	Handling the timer counting down itself. **/*FIX CODE*/** The following code contains UI update, which can cause an error if running in a different thread. You need to alter the code to make this works.
+ void endGame()	This method is called when the game is finished (eg. The timer has run out). **/*FILL CODE*/** Create an Alert here, set all proper details and show it. **Do note that the detail must exactly match the example above to get the full score.** When closed, check for the result, and do the following: - If the player has click YES: call resetGame(); method - Otherwise, call Platform.exit(); to close the game.

3. Criteria

Main UI

- Start button works = 1 point
- Each Keypad button works = 1 point

Thread

- Individual Letter Flip animation **plays successfully** = 1.5 point
- Word Flip animation **plays correctly** = 1.5 point

(no freeze, each letter starts spinning without needing to wait for previous letter)

- Countdown Timer works = 1 point

(no freezing the entire program or crash)

- No Exception has been raised throughout the program = 1 point

(From trying to update UI on non-JavaFX thread)

Alert

- The Alert is raised after the timer ends = 1 point
- The Alert Detail is correct = 1 point
- The results have been handling correctly = 0.5 X 2 = 1 point total

Total **10 points**