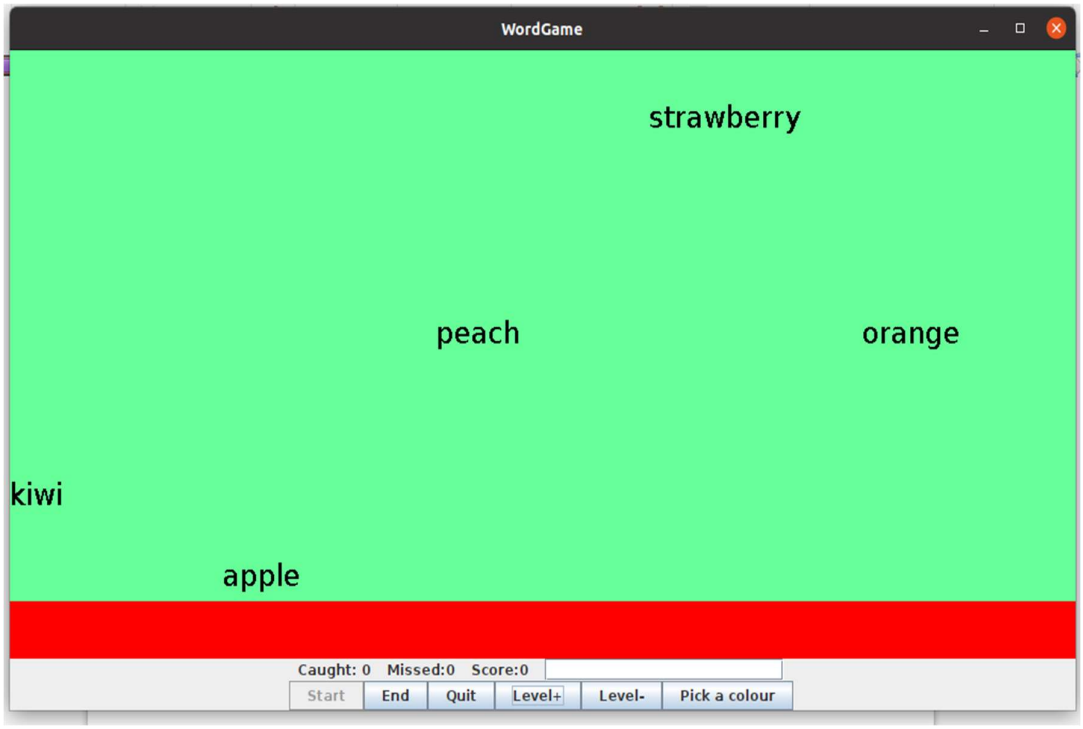
LIBHONGO MKO MKXLIB001

Computer Science : CSC2002S

Assignment 2: Parallel Programming and Concurrency

Type Falling Words Game implemented using java swing



Introduction

Computer Science students doing the CSC2002S course in 2021 were required to design a multithreaded java program. The Assignment is under the Parallel and Concurrency section. The program has to ensure thread safety and concurrency in order for it to work well.

The program is a typing game whereby the user is provided a Textbox for typing the words. The game had to follow the Model-View-Controller pattern . The Model-View-Controller pattern is an architectural pattern that helps in separating the system into three layers mainly the Model, View and Controller. The Model handles all the data logic meaning the source code, while the View is what the user sees which is the User interface. The Controller controls the flow of data to the objects that are created in the model, so as to update the user interface. The document will provide a description of all classes used; description of the java concurrency features used. Explanation of how the system was validated and debugged and also additional features that are used.

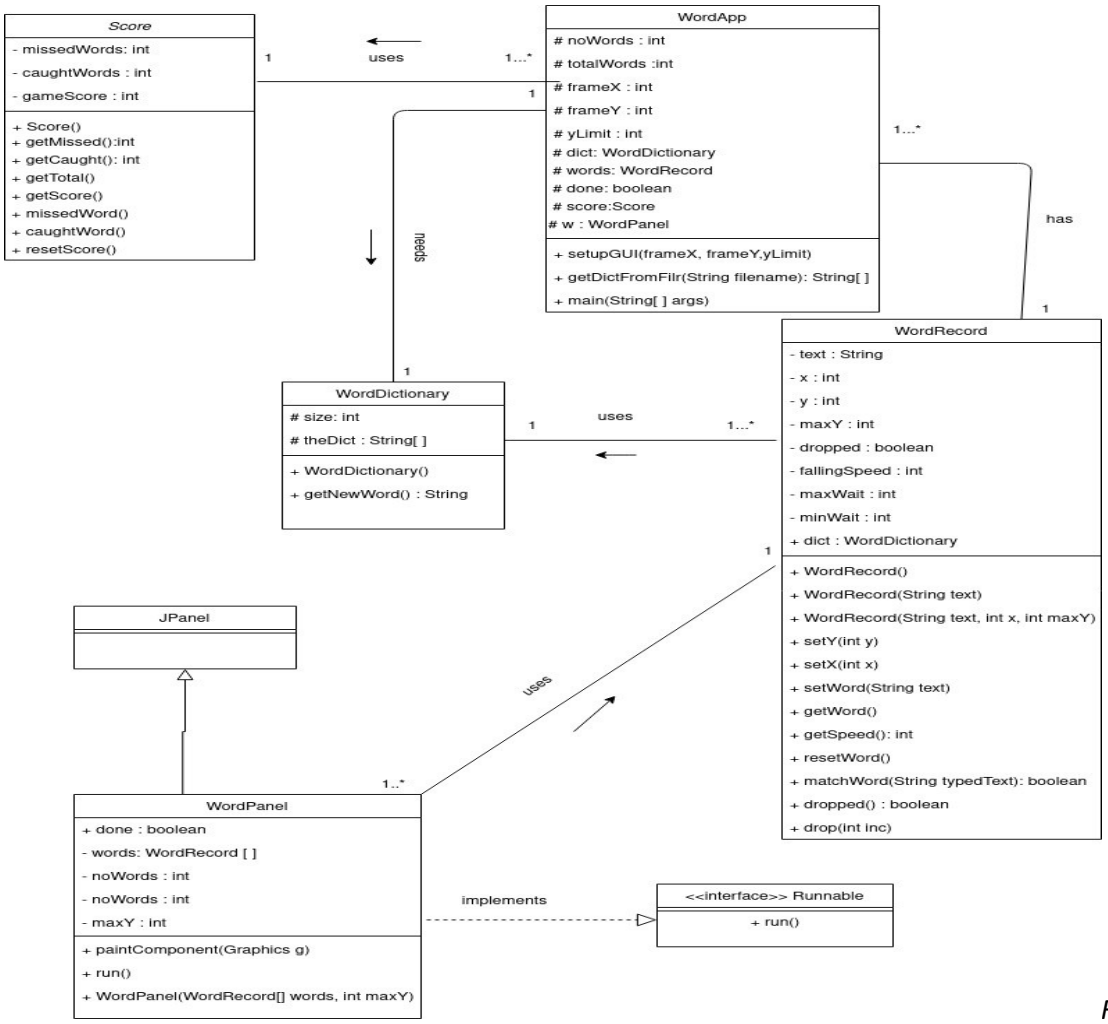


Figure 1: Class diagram

Class Description

The program has 5 classes mainly WordApp.class, WordDictionary.class, WordPanel.class, WordRecord.class and Score.class. The classes were provided as a starting point for the game, they were all stored in a package called skeletonCodeAssignmt2 repository. All source codes are found in the src directory and all the compiled versions are in the bin directory.

WordApp class

This class has the main method for handling the creation of the Graphical User Interface. The class was modified to cater for the creation of the threads for running the program.

Methods

setupGUI:

this is a method that returns no value and has four parameters:

(int frameX, int frameY, int yLimit). These parameters are responsible for setting the dimensions of the window that will be displayed to the user.

The user interface created is as follows:

Figure 2: User interface

Description of buttons in the interface:

The user presses the Start button and the words start falling, activating the End button. The Start button creates thread of the WordPanel class, so as to invoke the method, causing the game to start.The user can stop the game by pressing the End button which resets the current running counters and the words in the GUI. Pressing the Quit button closes the game entirely.

getDictFromFile:

This method is for generating the dictionary of words from the file provided by the user that will be used in the game as the words that are falling in the user interface.

This method has only one parameter which is the filename and returns an array of the words taken from the file.

WordDictionary class

This class is for generating the dictionary of words that will be used by the game. There were no modifications that were made in this class, only javadocs were added.

Methods:

WordDictonary:

This is the constructor of the class. There are two types of constructors that were provided. The default constructor(no argument) and a constructor that takes in an array that will store the words.

getNewWord:

This method returns a newly generated word by using the random generator for returning the index of the word to be used.

WordPanel class

This class inherits the JPanel component which is part of the swing package. This class is responsible for creating the GUI components such as the buttons. The class implements the Runnable interface for parallelizing the game. The change that I made is implementing the run method.

Methods:

paintComponent:

This method is responsible for initializing the GUI components in the window. The colours of the interface and fonts are set in this void method.

run:

This method is responsible for the work that will be done by the threads. The method iterates through the words list and drop each word using a specified speed. It then checks whether the word is dropped or not. If it is dropped the missed counter is incremented or if it is not dropped the caught counter is incremented and then the user interface becomes updated in runtime.

WordRecord class

WordRecord class for updating the counters of missed, caught and score. No changes were made in this class.

Methods:

The class has a set of setters and get methods for updating the record of a word such as the matchWord method which checks whether a typed word matches one in the wordDictionary array

Score class

This class contains atomic class variables that record the number of words caught and missed. They are used to store the scores according to their categories such as caught and missed.

Concurrency features

In ensuring that concurrency in the program is met, the words fell at different speeds so as to ensure that the user does not type all the words at the same time. Further concurrency is discussed below:

WordApp class

The class has a volatile variable called done. This variable is volatile because it is accessed in main memory by all the threads that will be executing. Making the variable volatile ensured protection against other threads from modifying it. All modifications made by the threads have to be updated in main memory or else there will be runtime errors in the program making the game to not stop when required.

WordDictionary class

The class has a synchronized getNewWord method, the reason why the method is synchronized is to block the next thread's call to the method as long as the previous thread's execution is not finished. If this method is not synchronized, a racecondition may be faced.

WordPanel class

This class has a volatile variable called done. It is volatile so as to modify the value of the variable by different threads. This ensures that the variable is thread safe. It means that multiple threads can use the variable at the same time without any problem.

WordRecord class

All the methods in this class are synchronised. This allows only one thread to use the method and edit the values of the class instances at any time. This is to prevent any data lost due to incorrect order of execution due to methods not being locked.

Code assurance

Thread safety

Thread safety refers to data that can safely be shared in a multithreaded program while preventing data leaks and deadlocks. In ensuring thread safety I used immutable instance variables and declaring them as **final** and also made variables that can be shared amongst the threads to be private. I also used volatile variables, and synchronized methods.

Thread synchronisation

Thread synchronization ensures reliable communication between threads. My program has synchronized methods which prevent threads from executing the same method at the same time which may result to a race condition down the line. A race condition occurs when two threads operate on the same object without proper synchronization and the operations interleaves each other. Without synchronisation, two threads can access the score class and one might change

the value before another but the other one will change the value again without including the changes done by the first thread, deadlock. This problem could change the recording of the score program which could lead to problems such as falling more than the expected number of words.

Liveness

Liveness refers to the ability of a concurrent program to execute in a timely manner. The program is running as expected because when the counter(number of words) that is specified by the user is reached the programs stops executing and displays a message to the user. Additionally, when the user presses the End button the program stops executing and all counters are restarted at the same time.

No deadlock

Deadlock is a situation whereby two or more threads are blocked forever, waiting for each other to release locks. The program uses swing which uses multiple threads, preventing deadlocks was one of the major challenges. I made sure that my program does not execute foreign code while holding a lock, by that I mean objects of other classes that are created to fulfil a particular purpose were not created while there were active locks. I also used encapsulation to prevent any deadlocks by making instance variable private and ensuring that **this** is not locked.

System validity

Testing my system validity I first ensured that the code runs smoothly and the graphics can be clearly drawn to the screen by using the command :

```
System.setProperty("sun.java2d.opengl","true");
```

I further tested my counters whether they update correctly when each word was missed, caught and score update. The score update had trouble at first because it was incremented by 1 instead of the length of the word. I fixed that. I also tested whether the events associated with each button was working correctly. I adjusted the speed so that each word cannot fall at the same time. My code runs efficiently on JGrasp installed in both Linux and Windows 10.

I tried crashing the system but not providing the required arguments and as expected my program did not run. I varied the arguments to ensure that all words can be displayed and move

at different paces. Through all this testing I can confirm that my program works. Bugs could still appear in occasional cases, since the program is multithreaded, and bugs appear once in a while and hard to track but further tests are required to find such bugs.

Model-View-Controller

The Model-View-Controller pattern is an architectural pattern that helps in separating the system into three layers mainly the Model, View and Controller. The Model handles all the data logic meaning the source code, while the View is what the user sees which is the User interface. The Controller controls the flow of data to the objects that are created in the model, so as to update the user interface.

The View of my program is the Graphical User Interface which the user interacts with. The controller in my perspective was the event listeners for the buttons and the text editor where by whatever interaction the user makes, the controller triggers a certain method in the logic layer of my program. Model of my program are all the classes that are dependent on each other, which are responsible for the execution of statements and operations such as generating the GUI.

Additional features

Added two buttons called Level+ and Level - which are used for increasing and decreasing the speed of the game. Level + increases and Level - decreases the speed. The feature was implemented to cater for users with different typing skills. Some users are slow typers while others are fast at typing. Increasing and decreasing the speed promotes user friendliness.

I also added a button called Pick a colour that allows the user to pick a colour that they want the background to be. When the button is clicked a colour window is displayed in the screen, providing the user with a variety of colours to pick from.

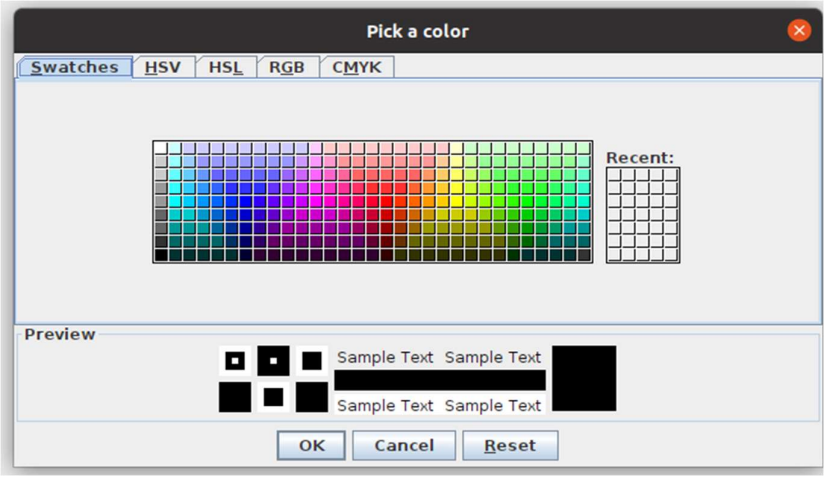


Figure 3: Colour window