Batandwa Mgutsi
MGTBAT001
CSC2002S - Assignment 2

Introduction

This report aims to detail the implementation of the WordApp application written by Batandwa Mgutsi. The Application is a Java AWT GUI app that drops words from the top of the screen and the user has to type the words as fast as they can to get a larger score.

Architecture

The application has two views of interest: the WordPanel and the score HUD. To animate the words in the WordPanel a second thread is used. In the source code this thread is referred to as the Animation Thread and runs as an EventLoop. That is, it runs in a loop polling events from the Main Thread and rendering the words at approximately 30Hz. To allow for this “push-poll” communication a ConcurrentLinkedQueue, from the java standard library, is used. The reason this data structure was chosen is because it is non blocking, all push and poll operations return immediately.

The Main Thread pushes events to the Queue, and the Animation Thread gets these events and processes them accordingly. These events are:

1. On Start - The animation thread should begin animation
2. On Stop - The animation thread should stop animation, reset the already falling words, and reset the score.
3. On Quit - The animation thread should break (i.e the user pressed quit)
4. On Data - The user has entered text and the animation thread should pass this to the controller and update the score accordingly.

The logic for the animation thread is implemented in the WordPanel class. This class also acts as the main controller for the game as it is the one that updates the user’s score. The Score class is implemented such that it is listenable. When the animation loop updates the score model in the Score.currentScore instance, the score View created in the Main Thread is updated in real time.

In total the program only has three threads: the Main Thread, the Java Swing Thread, and the Animation Thread. The Main Thread is mostly idle, as all it does is initialise the app and pass events to the animation thread. Browsing through the code, one will notice that the Score class is only used by the Main Thread during initialisation, and the Animation Thread during animation, therefore it does not need the synchronized methods as seen in the code. These methods were synchronized simply for planned future extensions to the project.

The Java Swing Thread and the Animation Thread are the only threads accessing the same data at the same time (i.e WordRecords). The Java Swing thread, however, only reads the data, and reads it so frequently that it always draws the latest state from the Animation Thread. Therefore, even here, concurrency features were unecessary, but the author (Batandwa Mgutsi) was afraid to remove them because the assignment is about Concurrency. In this case, the only thing the features do is prevent the two threads from accessing the same WordRecord at the same time.

Batandwa Mgutsi
MGTBAT001
CSC2002S - Assignment 2

The program has the following source files, as can be seen from the docs:

File	Job
EventLoop	Decodes events from the Main Thread for the Animation Thread. It does this with a ConcurrentLinkedQueue.
Score	Keeps a static Score instance that keeps the current score for the user. When the score is changed the Score View is updated.
TimeService	Provides a delta time service for smooth animations.
WordDictionary	Keeps all the words read from the dictionary file.
WordPanel	The actual game controller. Animates the words, handles Animation Thread events, and updates the score.
WordRecord	A state for a single word. WordRecords are reused. See code.

Notable Code features

1. Clean build and run commands for make
2. Clear symbol names (e.g variables). The symbol names make comments unnecessary
3. Simple architecture based on event loop rather than bare shared memory which would have led to hard to debug race conditions
4. Descending method complexity. Cleaner, english-like methods make use of private complicated methods. Rather than one big complicated method, the complexity gets more as you go one method call lower.

Conclusion

As a side note, this assignment has taught me how to write a proper event loop in Java. That’s a skill I’ll never lose.