**CSC2002S – ASSIGNMENT 2**

**PCP2 Typing Tutor & Hungry Word Mover**

**MYZMBA002 | August 2022**

**Duplicate Words**

To handle duplicates within the Typing Tutor game, modifications were made to the *CatchingWord.java* in the *TypingTutor* Package.

This stemmed from the initial understanding of the *CatchingWord* class’s design to loop through the *words* array and to find words matching to the target word typed in by the user/player. On finding a match to the target word, the *matchWord* method of the *FallingWord* class was called on the particular target, which then reset the word and incremented the score board.

Modifications made to the class included adding an array to store the duplicate words which was called within the existing iterative *while* loop. On collecting all possible duplicates, outside of the while loop a *for-each* loop was used to iterate and compare the y-coordinate of each duplicate word before calling the *matchWord* method on the one with the highest score (i.e. lowest duplicate on screen)

**Hungry Word Mover Class**

This class was created a s a combination of the *FallingWord* class and the *run* method of the *WordMover* class and extended java *Thread*. This was done in an effort to avoid possible errors that would arise from making too many modifications to the existing classes to accommodate the function of *HungryWordMover.*

Of the *FallingWord* class all the existing constructors were copied with modification’s to be used as constructors of the *HungryWordMover class.* In contrast to the fact that the *FallingWord* manipulated the y-axis to move the words vertically across the screen, within the *HungryWordMover class* the maxY value was changed to a maxX value and set to 300.

Also, within the set methods the setY was changed to keep the Y value constant whilst the setX method incremented the x value. Then finally within the drop method, the setX method was called and the x value incremented.

All methods were declared as synchronized methods just as in the original class.

In addition, the run method of the *WordMover* class was copied with very little changes made only to calling the correct class names within the loop. The reason for creating this run method of its own instead of inheriting the existing one was because of the movement of the words horizontally instead of vertically, which could pose possible other race conditions had we utilized the existing methods, this also helped with the randomization of the *HungryWord* by using different fruit names each time and allowing a period for which the thread was *Sleeping* too. Apart from that, the class thread ran similarly to the other *FallingWord* Threads, like being created at the same time and moving at the same speed.

To display the hungry word on the screen, modifications were made to the JPanel *GamePanel* class (Line 58 and 59).

Within the createWordMoverThreads method of the *TypingTutorApp* an instance thread of the HungryWordMover was created and started. In the main method the array size was set to zero, keeping in mind that we wanted to display one word at a given time.

**Race Conditions**

Threads are not killed with each new press of the Start Button; this was shown by the fact that the counter still increases after multiple calls of start.

Also, the Quit Game button functions the similarly to the Pause Button, as one can “re”-start the game from the same point even after having pressed quit.