TAKUDZWA SHERPHERD MASEVA

MSVTAK002

ASSIGNMENT2

Task1: Ensure that, when duplicate words appear on the screen that the lowest word disappears first (this is not currently supported by the game).

How: For this task, l made changes to the CatchWord class. Inside the run method of this class and in the second while loop, l modified the if statement that was checking if the word falling matches the word that has been typed by the user by calling the matches method which checks if the words are equal and returns a Boolean value and making the word disappear depending on the Boolean value returned by the matches method. If the if condition held then the score was incremented and the while loop broke, so there was nowhere where duplicates where being checked for. To implement this l removed the if statement that was calling the match method from the FallingWord class and replaced it with an equals method and made sure l loop through all the falling words by removing the break statement. As the words are falling now, I am checking if the falling word is equal to the word that a user has typed and if it is, it is made the current falling word that is closest to the pink panel which is the word that needs to be reset. As l keep on iterating through the other falling words, if there is another falling word that is equal to the word typed by the user, then l compare the height of the current falling word, which is the y coordinate of that word and the height of the duplicate word and if the duplicate has a greater height than the current falling word, then l reassign my falling word which is closest to the pink region and need to be reset equal to the duplicate word. Outside of the while loop and after finding the word closest to the pink region, l then call the match method which will reset the falling word object and return true then the player's caught score is incremented.

Task2: A green word – Hungry Word - that is selected randomly from the dictionary, appears randomly and moves horizontally across the middle of the screen. Any words that Hungry Word bumps into disappear (and their total is added the Missed counter). If the user types the Hungry Word then it disappears and the total is added to the Caught counter. If the Hungry Word exits before being typed, the total is added the Missed counter.

The HungryWordMover class is like a direct copy of the FallingWord class but with some modifications to it. In the Falling Word class, words move from the top to bottom but the HungryWordMover class l implemented in such a way that words move from left to right

horizontally. Now to begin with, since the two classes are similar I made the HungryWordMover.java class extend the Falling Word.java class so that I inherit the methods and attributes of FallingWord class. I also changed the maxX, maxY, x and y instance variables from the FallingWord class protected so that the HungryWordMover class can access them. In the HungryWordMover class, I override the drop method of the FallingWord class adding the parameter that's passed to this method to the current x value this time instead of the y value and then calling the setX method again this time instead of the setY method as in the FallingWord method, so that the word move to the right since its increasing the x value.

Colour of the HungryWord

Since the HungryWordMoverClass is a child of the FallingWord class and the GamePanel's constructor takes in an array of words of type FallingWord it means now this array can now also have words of type HungryWordMover as well as words of type FallingWord. In the method paint Component of the GamePanel class and inside the else-if statement where we are looping through the array, I added another if statement that checks if the falling word in the array is an instance of HungryWordMover class and if so, it sets the graphic colour of the object to green.

HungryWord and FallingWord bump

I created another class, called Intersection which extends the GamePanel class which will be used to find the width of the words that are being drawn on the panel and get their coordinates. When I have the width and the starting coordinates of a word, I could find the ending coordinates of a word and also the intersection of two words. I also have a for loop that loops through all the FallingWords that are being run and compare their coordinates with the coordinates of the hungry word and if it is within the region, the drop value of the falling object is set to true so that it can be dropped and disappear on the screen.

Threads to Move words

Inside the TypingTutorApp Class and inside the CreatWordMoverThreads methods, where word objects are created and added to the words array, after all the FallingWord objects are created, I added another object which is an instance of the HungryWordMover which will be passed and used in the game panel class to render words on the screen. After adding the falling objects and a HungryWordMover object into the words array, each object in the array is added to and individual wordMover object using a for loop and these wordMover objects are the threads and are started all at once.

Race conditions identified

1. In the Score Updater class, there is a possibility of a race condition when the text for the JLabels scoreView, caught and missed is being set. More than one thread can attempt to set the text for these JLabels leading to a race condition because the method being used to set the text for the JLabels, the set Text method is not thread hence not synchronized.