TSHTHA094_CSC2002S_PCP2

Thabelo Tshikalange

August 2022

1 HungryWordMover

The hungryWordMover is similar to the word mover.The difference are as follows:

1.1 The Constructor:

an additional constructor is used that has a list of all the falling words that are set from the Typing tutor class.

1.2 The run method:

The method was adjusted to first find the hungry word in the given list so that it can use it when dropping words because it drops word horizontally. Then it goes on to use the hungry words position and compares it to the other words to see if they are in contact if so, it resets that word and sets it as a miss.

```
1  while (!myWord.droppedx() && !done.get()) {
2      if (myWord.equals(TypingTutorApp.hungrywWord)){
3          myWord.dropx(10);
4          for (int i = 0 ; i <words.length; i++){
5              if(!words[i].equals(TypingTutorApp.hungrywWord)){
6                  if (words[i].getY() == myWord.getY()){
7                      if ((myWord.getX()+myWord.getWord().length
                           ()/2 -words[i].getX()) < 1){
8                          score.missedWord();
9                          words[i].resetWord();
10                     }
11                     if ((myWord.getWord().length()/2- myWord.
                           getX() -words[i].getX()) < 1){
12                         score.missedWord();
13                         words[i].resetWord();
14                     }
15                 }
16             }
17         }
18     }
```

2 Changed Classes

2.1 TypingTutorApp

The Typing tutor App was changed as follows:

A random number generator was made that gives a number between 0 and the number of words. The using that number the hungry word was chosen if the falling words index matched that of the random word. The after it was set to be the hungry word that all the other classes will call reference to.

Furthermore the addition of an instance of the HungryWordMover class was made, this will store all the falling words then, form there the hungry word will be run different.

Then the wordMover is started.

```
20 static FallingWord hungrywWord;
21 static int random;
22 static HungryWordMover[] wordMovers;
23
24 random = (int)Math.floor(Math.random()*(noWords-1));
25
26 for (int i=0;i<noWords;i++) {
27             if(i == random){
28                 words[i]=new FallingWord(dict.getNewWord());
29                 words[i].setPos(0, gameWindow.getHeight()/2+1);
30                 hungrywWord = words[random];
31             }
32             else{
33             words[i]=new FallingWord(dict.getNewWord(),
                   gameWindow.getValidXpos(),yLimit);
34             }
35         }
36         //create threads to move them
37         for (int i=0;i<noWords;i++) {
38                 wrdShft[i] = new WordMover(words[i],dict,score,
                       startLatch,done,pause);
39                 wordMovers[i] = new HungryWordMover(words[i],
                       dict,score,startLatch,done,pause,words);
40         }
41         //word movers waiting on starting line
42         for (int i=0;i<noWords;i++) {
43             wrdShft[i].start();
44             wordMovers[i].start();
45         }
46     }
```

2.2 FallingWord

Changes made To the falling word class were to account for the hungry word to allow it to move vertically instead of vertically and for the word to reset to the

start when it reaches the end of the frame.

To make the game fair and not have a hungry word that always pops up after it is typed It was set to restart at -200 this is to for some type of delay before the next word comes up.

The match word class was changed to have the hungry word accounted for when it gets typed.

```
47 public synchronized boolean matchWord(String typedText) {
48         //System.out.println("Matching against: "+text);
49         if (typedText.equals(this.word)) {
50             resetWord();
51             resetWordx();
52             return true;
53         }
54         else
55             return false;
56     }
57 
58 public synchronized  void setX(int x) {
59         if (x > TypingTutorApp.gameWindow.getWidth()){
60             x=TypingTutorApp.gameWindow.getWidth();
61             droppedx=true;
62         }
63         this.x=x;
64     }
65 public synchronized void resetPosx() {
66         setX(-200);
67         setY(260);
68     }
69 public synchronized void resetWordx() {
70         resetPosx();
71         word=dict.getNewWord();
72         droppedx=false;
73         fallingSpeed=(int)(Math.random() * (maxWait-minWait)+
               minWait);
74         //System.out.println(getWord() + " falling speed = " +
               getSpeed());
75     }
76 public synchronized  void dropx(int inc) {
77     setX(x+inc);
78 }
79 
80 public synchronized  boolean droppedx() {
81     return droppedx;
82 }
```

2.3 CatchWords

The changes to the Catchwords class was to account for having duplicate words and removing the lowest one. we first find the word that is targeted then form there we go over the words that are already on the screen to see if the match the targeted word if so then we check the lowest one and that will be the one that gets removed.

```
83  while(pause.get()) {};
84      int counter = 0;
85      int Y_value = 0;
86      if(words[i].getWord().equals(target)){
87          for(int j = 0; j < noWords; j++ ){
88              if (words[j].getWord().equals(target)){
89                  if (words[j].getY()>Y_value){
90                      Y_value = words[j].getY();
91                      counter = j;
92                  }
93              }
94          }
95          if (words[counter].matchWord(target)) {
96              System.out.println( " score! '" + target); //for
                    checking
97              score.caughtWord(target.length());
98              //FallingWord.increaseSpeed();
99              break;
100         }
101     }
102     i++;
103 }
```

2.4 Wordmover

The changes here were to not have the selected hungry word be set to move like all the other words.

```
104 if (!myWord.equals(TypingTutorApp.hungrywWord)){
105         myWord.drop(10);
106     }
```

2.5 GamePanel

Using the selected hungry word it was set to be drawn with a different color from the other ones

```
107 //Check if word is hungry
108 if(words[i].equals(TypingTutorApp.hungrywWord)){
109     g.setColor(Color.green);
110     g.setFont(new Font("Arial", Font.PLAIN, 26));
111     g.drawString(words[i].getWord(),words[i].getX()+borderWidth
            ,words[i].getY());
112 }else{
113     g.setColor(Color.black);
114     g.setFont(new Font("Arial", Font.PLAIN, 26));
115     g.drawString(words[i].getWord(),words[i].getX()+borderWidth
            ,words[i].getY());
116 }
```

3 Things noted

- Some race conditions that arise where two words have the same X coordinates and the same speed making it hard to see the words.
- some words are set to be close to the edge of the screen and sometimes the word will be to long and fall out of the screen.
- The distances of the limit of Y does not account for when the window is maximised or reduces.