Wiki

**1. What is the problem to be solved.**

Desktop TypeRacer is a game that is designed to improve the speed of typing on the keyboard, while having fun.

**2. What are the objectives of the project and how it solves the problem.**

With the help of computer logic, application (Desktop TypeRacer) shows how many words per minute you type (for example, 45wpm, 50wpm, 70wpm, etc.) in - this way you can compare your achievements, as this was a special option to review the last 10 entries that you made and 10 most - your best achievements are recorded in the "Ranking list". Through these comparisons you will be able to obtain very accurate an idea of ​​how fast you write and if you need while having fun with the game will improve your speed / performance of writing.

The application has a simple design that is very easy and convenient to use. You have three levels of difficulty as follows - "Easy (150)", "Medium (250), "Hard (350)". Number in the brackets indicate how many words is a text that you have to write.

**3. Who are the authors of the project and what is the rough division of labor between them.**

The author is George Krachkov and Hristo Georgiev. Distribution of tasks is as follows: Hristo Georgiev - 1, 3, 5, 7, George Krachkov - 2, 4, 6

1. When you run the program is loaded window from which they can select the following functions:
   1. Start Game
   2. Ranking list
   3. About
2. When selecting the "Start Game" reload window, this time choose the game's difficulty as follows:
   1. Easy (150)
   2. Medium (250)
   3. Hard (350)
3. Upon selection of difficulty reload window again, and it was charged in a random file sheet, depending on the difficulty. Upon discharge of the sheet on the display monitor of the length of words, and when necessary, the words are transferred to the new line.
4. When entering a word check, checking whether it is spelled correctly, if so, by pressing the "space" to go to the next word.
5. Specific information is displayed to the right of the text is:
   1. displays the speed with which the user writes (wpm), if the speed is below the window 30 is green, between 30 and 60 - 60 above, and yellow - red.
   2. Displays clock that counts down the seconds as you type.
   3. Write out the length of the text + level.
   4. Displays the number of letters in the text.
6. Upon the completion of the level is reloaded chart of the last 10 times, the syntax is:

username -> wpm -> Level -> the number of letters in the text

1. At - performance is directly recorded in the Ranking list - a, where the maximum is 10 people.

**4. How to run a project from another developer familiar with Java, Mercurial and Eclipse.**

Programmer familiar with a little Eclipse, you can launch the application simple and import - not the project and then run (compiled). Most - probably just need to fix the Build Path - it. As you enter it you just need to delete everything and then add on "Add Library" system libraries, which is the default. Then from the "Add Variable", you must add-ons for "swt", and for this purpose you should go to "Extend", \* swt \*, in the text box and from there to select the last 3 plugin from "plugins".

**5. How the application works. What are the main points of his work, more interesting**

**code fragments.**

Functional design is written without the help of any label - I. Thus in a simple design of the set program. The whole "draw" part of the program is redefined, ranging from tables and get to textarea.

public int wpmCounter(**int** numberOfLetters, **long** timeSinceNow) {

**int** wordsCount = (**int**) (numberOfLetters / 5);

**float** minute = (**float**) timeSinceNow / 60;

**int** newWPM = (**int**) (wordsCount / minute);

**return** newWPM;

}

This function takes, elapsed time and the written characters to this point. Code allows easily add penalty points if use "backspace".

**public** ArrayList<SaveUserScore> getTheScourFromBestToWorst(

ArrayList<SaveUserScore> userData) {

ArrayList<SaveUserScore> scoreRating = **new** ArrayList<SaveUserScore>();

**for** (SaveUserScore s : userData) {

scoreRating.add(s);

}

Collections.*sort*(scoreRating, **new** Comparator<SaveUserScore>() {

**public** **int** compare(SaveUserScore s1, SaveUserScore s2) {

**return** s2.getWpm() - s1.getWpm();

}

});

**return** scoreRating;

}

6.  Tutorial video:   <http://www.youtube.com/watch?v=RwNYDZQjurk>