PROGRAMMING ASSIGNMENT 4
HUBBM-RACER: A First GUI Application in *Java*

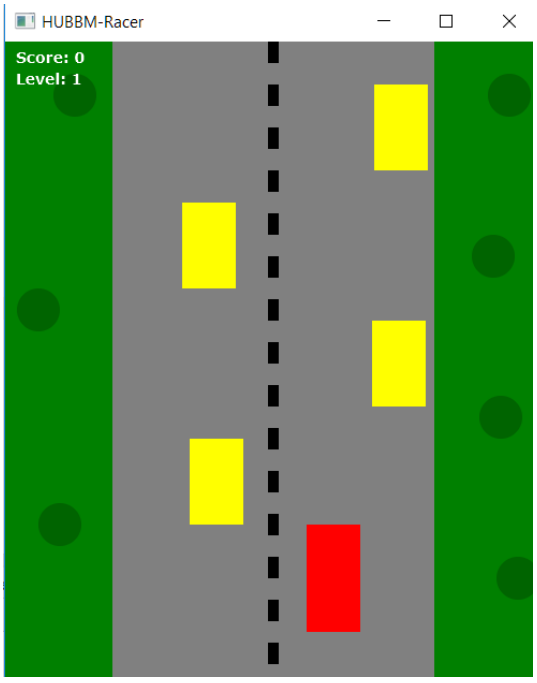
Subject: **Coping with Java GUI framework (JavaFX)**
TAs: Selim Yilmaz, Cemil Zalluhoglu, Bahar Gezici

Release Date: 5/08/2019 (from 23:59:59)
Due Date: 5/31/2019 (until 23:59:59)

1 Introduction

In this assignment, you are expected to gain practice on developing a Graphical User Interface (GUI) application using Java programming Language. As opposed to the *command-line programs* where the interaction between the user and the computer often relies on string of text, a GUI program offers a much richer type of interface where the user uses a mouse and keyboard to interact with GUI components such as windows, menus, buttons, check boxes, text input boxes, scroll bars, and so on. The fact that most people today interact with their computers exclusively through GUI, a developing a GUI-based application has become a must for the new developers.

There are lots of frameworks to develop a GUI application (such as Swing, SWT, AWT, and the like). In this assignment, you are to employ `JavaFX` framework to complete this assignment. JavaFX is a software platform for creating and delivering desktop applications, as well as Rich Internet Applications (RIAs) that can run across a wide variety of devices. JavaFX has support for desktop computers and web browsers on Microsoft Windows, Linux, and macOS.



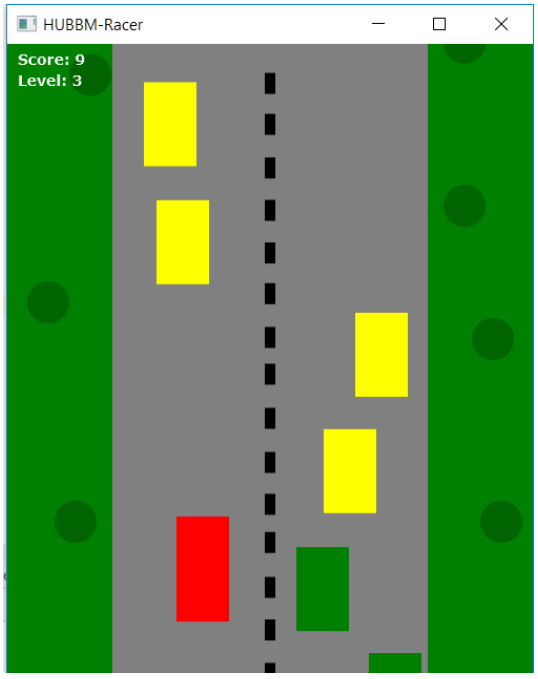
In this assignment, a very simplified version of *Road Fighter*, a very popular Atari game in nineties, is expected from you to develop through JavaFX framework. All details that you need while designing the game is explained in the following section.

2 HUBBM-Racer

As expected, the goal in HUBBM-Racer game is to score points as much as possible, which depends on the number of competitors that you have overtaken —details regarding

point calculation has been explained later. The top-view of initial scene of the game is given in the figure on the right. As you see, the scene comprises of a road with two lanes (each is divided by lane lines) and green field on both sides in which green circles representing trees (or shrubberies etc.) are located. In addition, there are a number of objects on the road which represent the racing cars. Among them, the red one represents the user's car whereas the yellow ones represent the rivals cars. The scoreboard as well as level information is placed at the top-left corner of the scene. The title of the window is set with the name of the game that is 'HUBBM-Racer'. A short screen capture can be found in the announcement on *Piazza* and you are strictly advised to watch it at least one so that you better grasp how to design the game. The executable jar file has and will not be shared with you as there are lots of decompiler on Internet that converts jar to source code :)

You are to design a scene and feel free to design it with your imagination. However, the base scene of your implementation should be the one given in this paper. User should use arrow keys to move his/her car. Up arrow key enables it to vertical move; while left and right arrow keys enables horizontal move. You should animate all the objects of interest (trees, lane lines, and the cars in ours implementation) when user presses up arrow key so that it makes user feel as if his/her car moved vertically on the road. When user releases keys, however, vertical or horizontal move should be stopped not abruptly but gradually as in our implementation.



User's car is not allowed to leave main road (i.e., passing through left or right green field) nor it moves through a curved road. So you do not need to handle such cases. As you see from the scene capture on left side, the rival's cars are painted with a color (green in out implementation) other than its initial color so that they are marked as overtaken. Every time user's car overtakes his/her rival's car, score should be updated. Score calculation should be as follows:

$$S_n = S_c + L_c$$

where S_n is the next, updated score whereas S_c and L_c represent current score and current level, respectively. That is to say that the next score is equal to the level included current score. Current level L_c simply determines the animation speed —the higher the level is the faster the objects moves. Level should also be updated after every specific interval, which can be determined arbitrarily by you —keep it measurable so that we can observe the update through the evaluation of your work! There should no limitation be introduced for both score and the level, namely they should be increased throughout the game.

The only case where user's car collides with one of the rival's car is the case that end the game.

So you should always check if the user's car 'touches' on other cars throughout the game. Such case is illustrated into the right side capture. As seen from the scene, user's car and the one who user collided with are painted with another color other than red and yellow (black in our implementation). It is very important to denote that there should be initially no collision between rival's cars and between any of rival's car and user's car —see the figure at first page. So, keep that in your mind while creating the initial scene of the game.

Once user has reached the end of the game, a message text into the middle screen will be displayed. The message content should be same as the one in the capture. In addition, user will no more allowed to control the game, namely all the key attempt should be ignored except `ENTER` button. Do mind to ignore `ENTER` button throughout the game. When user pressed it, user should be welcomed with the initial scene as given in the first page. Every time, user switches into the new game, level and score should be set as 0 and 1, respectively.

3 Grading Policy

Task	Point
Report	10
`Graphical design`	10
`Loading game with no collision`	5
`Keyboard control`	15
`Animating objects`	15
`Score calculation`	10
`Level calculation`	10
`Animation speed with respect to level`	10
`Indicating overtaken cars`	5
`Color change in collision`	5
`Enabling new game`	5

4 Design Notes

- Design your implementation in Java 8 in which Lambda expressions are introduced, which considerably simplify the development. So, you are highly encouraged to make use of lambda expressions in this assignment.
- Reporting is mandatory. Report should cover *i*) problem definition (limited to max. 3 sentences), *ii*) solution approach (limited to max. 3 sentences), and *iii*) an explanation per class or objects (limited to a single sentence for each).

5 Submission Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. All the duplicate or Internet works (even if a citation is provided) are both going to be considered as cheating.
- You can ask your questions through Piazza and you are supposed to be aware of everything discussed there.
- You will submit your work from our department's submission system with the file hierarchy as below: This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

→ <student id.zip>
→ src.zip <DIR>

- The class name in which main method belongs should be `Assignment4.java`. All classes should be placed in **src** directory in `src.zip`. Feel free to create subdirectories, corresponding the package(s), but each also should be in `src` directory.