# Road Slider

Submission Date: 18th of November 11:59 PM

## Contents

## 1 Introduction

### 1.1 Submission

Submit a **folder** that is **only** containing your Java source files (HW3.java) to the course's Homework folder. Don't submit GameMap.java, as you are not allowed to change that file.

Full path: **F:\COURSES\UGRADS\COMP130\Homework\**

**Note:** COMP 131 students, please submit to the folder under the COMP 130 directory.

Please use the following naming convention for the submitted folders:

**YourPSLetter_CourseCode_Surname_Name_HWNumber_Semester**

Example folder names:

- **PSA_COMP130_Surname_Name_HW3_F18**
- **PSB_COMP131_Surname_Name_HW3_F18**

Additional notes:

- Using the naming convention properly is important, failing to do so may be **penalized**.
- **Do not** use Turkish characters when naming files or folders.
- Submissions with unidentifiable names will be **disregarded** completely. (ex. "homework1", "project" etc.)
- Please write your name into the Java source file where it is asked for.
- **No late submissions are accepted**.

## 1.2 Academic Honesty

Koç University's *Statement on Academic Honesty* holds for all the homeworks given in this course. Failing to comply with the statement will be penalized accordingly. If you are unsure whether your action violates the code of conduct, please consult with your instructor.

## 1.3 Aim of the Project

This homework is designed for you to feel comfortable creating object oriented graphics applications in Java. As in the previous homework, you are expected to complete or write functions. Moreover, you need to use functions to create or organize graphics objects. For this project, you are expected to implement a game where you try to avoid hitting obstacles and finish the road.

## 1.4 Given Code

When you open the project, you will see that there are parts already implemented in the project. These parts are essential for project to work with your current knowledge. You do not need to understand the given part to be able to do the project. **Do not change** anything in the given code, it will be indicated to you with a comment. **Only** write in the parts in between **'your code starts here'** and **'your code ends here'**. **DO NOT CHANGE ANYTHING** in the **'GameMap.java'** class, you will right code **only to 'HW3.java'**.

### 1.4.1 Given Constants

Constants are given at the bottom of the project. Throughout the explanations of implementation, you will see where you should use these constants. **Do not use** another variable or a static value for something if there is a constant variable defined for that purpose. Feel free to create additional constant and global variables if there is not a defined variable for that purpose.

### 1.5 Further Questions

For further questions **about the project** you may contact

- **Caner Korkmaz** at [ckorkmaz16@ku.edu.tr],
- **Merve Karakas** at [mkarakas16@ku.edu.tr],
- **Ahmet Uysal** at [auysal16@ku.edu.tr].
- **Yusa Omer Altintop** at [yaltintop15@ku.edu.tr],
- **Ayca Tuzmen Yıldırım** at [atuzmen@ku.edu.tr],

Note that it may take up to 24 hours before you receive a response so please ask your questions **before** it is too late. No questions will be answered when there is **less than one day** left for the submission.

## 2 Plan

This project consists of two main parts; in the first part, you need to create the road, draw the strips on the road, create the car, create the obstacles and draw the finish line. In the second part, you must handle the movement of the map on the y-axis, the game result; failure or success. The game fails when the car hits an obstacle, and you need to detect this situation, stop the game and show a failure message. If the player successfully arrives to the finish line without hitting any obstacles, then you stop the game and show the congratulations message.

### 2.1 Creating the Game Map

In this part, you need to create the map of the game. We have created the main methods you need to write. First of all, you need to create the lanes the car will stand on. Then you need to add the line strips (lines on the road) to create a real road like view. After that you need to add a finish line, as we want the players to be able to win the game. But the game also requires a challenge, so you need to create some obstacles. However, a game isn't fun if it is not beatable, so obstacles shouldn't prevent the player from continuing the game. You can find example photos throughout this PDF, and an example video in the folder. Please note that you don't have to change the background color to green, we have already done that for you.

#### 2.1.1 Adding the Road Borders

One of the methods we gave you is **addRoadBorders()**, and in this section, you need to fill in **addRoadBorders()** method. First, you should create road borders on the leftmost and rightmost side of the road. Borders are rectangles and they have width of **ROAD_BORDER_WIDTH** and height of **ROAD_HEIGHT**. Border color should be white. After you created rectangles, you should add them to gameMap. You should not put the rectangles outside of the road e.g. right side of the right border rectangle should be at the same location with right side of the road. The width of the road is equal to **APPLICATION_WIDTH** constant.

To add an object to gameMap, you should call add function on gameMap. Let's say you want to add an object named 'myRectangle' to gameMap, then you should write: gameMap.add(myRectangle,x,y) where x and y are coordinates of the added object. Note that gameMap is a GCompound, so this add method is no different than the ones we used when we wanted to add some objects to a GCompound object.

### 2.1.2 Adding the Lane Separation Lines

After adding the road borders, you need to fill in **addLaneSeparationLines()** method. Since our road has three lanes, you should implement this method such that it creates two lane separation lines. As you can see in the below screenshot, one separation line consists of same sized rectangles and there is space between those rectangles. One rectangle has the width **LANE_SEPARATION_WIDTH** and height **LANE_SEPARATION_HEIGHT**, and its color is white. The space between two rectangles is **LANE_SEPARATION_SPACE**. Separation lines should be all along the road. Separation lines should divide the road to 3 equal pieces in the x-axis, to achieve this you should use ROAD_BORDER_WIDTH, LANE_WIDTH, and LANE_SEPARATION_WIDTH to calculate locations of the rectangles in the x-axis. To calculate locations in y-axis, you should use LANE_SEPARATION_SPACE and LANE_SEPARATION_HEIGHT. After you create a line, you should add it to gameMap.

Be aware that after you added the road borders, the new width of the remaining road is equal to $(APPLICATION\_WIDTH - 2 \times ROAD\_BORDER\_WIDTH)$.

### 2.1.3 Adding the Road Lines

After separating the lanes, you should create road lines in the middle of every lane. We gave you **addRoadLines()** to fill in, you should create road lines in this method. Adding road lines to gameMap is similar to adding lane separation lines, there are same sized rectangles for one road line and there is space between two rectangles. One rectangle has the width of **ROAD_LINE_WIDTH**, height of **ROAD_LINE_HEIGHT**, and its color is white. Space between two rectangles is **ROAD_LINE_SEPARATION**. Road line for one lane should be at the center of the lane when we look to x-axis; to place the rectangles to correct locations you should make use of ROAD_BORDER_WIDTH, LANE_WIDTH, ROAD_LINE_WIDTH, and LANE_SEPARATION_WIDTH for determining x-axis, ROAD_LINE_SEPARATION and ROAD_LINE_HEIGHT for the y-axis.

### 2.1.4 Adding the Finish Line

A game should have a goal, otherwise it would be pointless. You are making a game like *Subway Surfers*, however that game is different from yours as it never ends. You will add a finish line, and the players will be able to feel a sense of pride and accomplishment of finishing the game. So, you will implement the **addFinishLine()** method we gave you to indicate end of the game. You

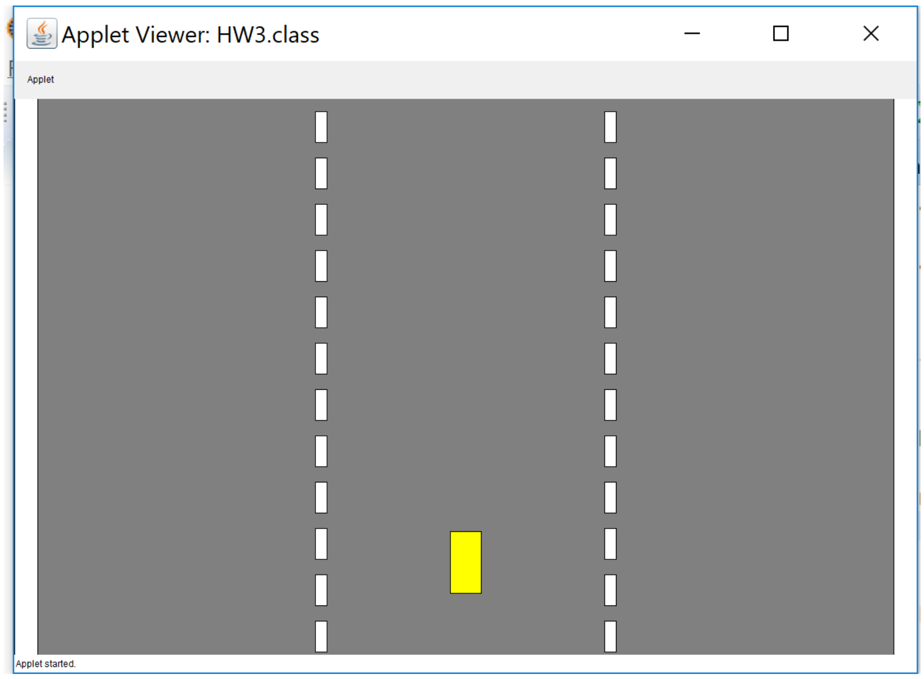


Figure 1: Lane Separation Lines

should create the squares and add them to gameMap. As you can see from the picture below, finish line consists of squares of the same size. There are two rows and **FINISH_LINE_SQUARE_NUM** columns of squares in the finish line. A side of a one square has the length of **FINISH_LINE_SQUARE_SIZE**. Square color is either white or black, and a square's color should be different from the color of the squares that are below, above, right or left to the square. For example, the square at the first row and first column is black, and the square on the right and below are white. You should put the finish line at the end of the road, and all squares should be inside the road, any square should not be outside of the road or on the road borders.

#### 2.1.5 Creating the Obstacles

In this part, you will create random obstacles and add them to gameMap. We divided this part to two methods: **addRandomObstacles()** and **GRect createObstacle(int width, int height, Color fillColor)**. Firstly, you should fill in the createObstacle method using the parameters given in the signature of the method itself. You will create a rectangle using width, height and fillColor. You will use this method when implementing addRandomObstacles(). After implementing createObstacle, you should fill in the addRandomObstacles method. This method should put rows of obstacles throughout the gameMap. In each row there will be exactly two obstacles and their lanes will be chosen at random. Also, you should avoid putting obstacles to the safe zones existing at the first and last part of the map. To be more specific, the y coordinate of the obstacle closest to finish line should be LENGTH_SAFE_ZONE_END. Also, there shouldn't

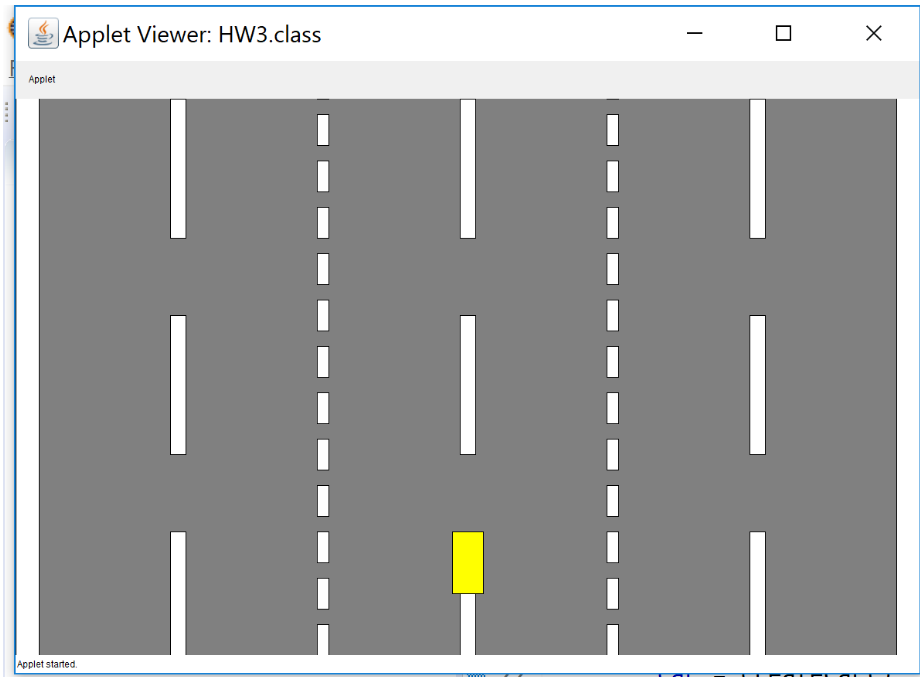


Figure 2: Road Lines

be any obstacles at the first LENGTH_SAFE_ZONE_START units of the map. Width of the obstacles should be exactly equal to width of lane. You should use given constants LANE_WIDTH and OBSTACLE_HEIGHT as sizes of the obstacle and there is $2\times(ROAD\_LINE\_SEPARATION+ROAD\_LINE\_HEIGHT)$ units space between rows of obstacles.

When you add an obstacle to gameMap, you should use addObstacle method instead of add method. For example, if you want to add obstacle named randObstacle to gameMap, you should write:

*gameMap.addObstacle(randObstacle, x, y);*

where x and y values are coordinates of the object.

#### 2.1.6 Creating the race car (Bonus)

Please note that this section is bonus, and we have already given you a simple car. This won't effect your grade out of 100, but we are planning to give bonus points for aesthetic and good looking cars, you can skip this part if you do not want to change design of the car to get bonus points.

You are just supposed to change the **GCompound createCar()** method, creating a car of your likes using GCompound methods. That method should create the GCompound, fill it with the parts of the car and return that GCompound. (It is a GCompound since you will have the car body and the wheels and etc.) We already called the createCar() in the run, and set the car to the return object of this method.

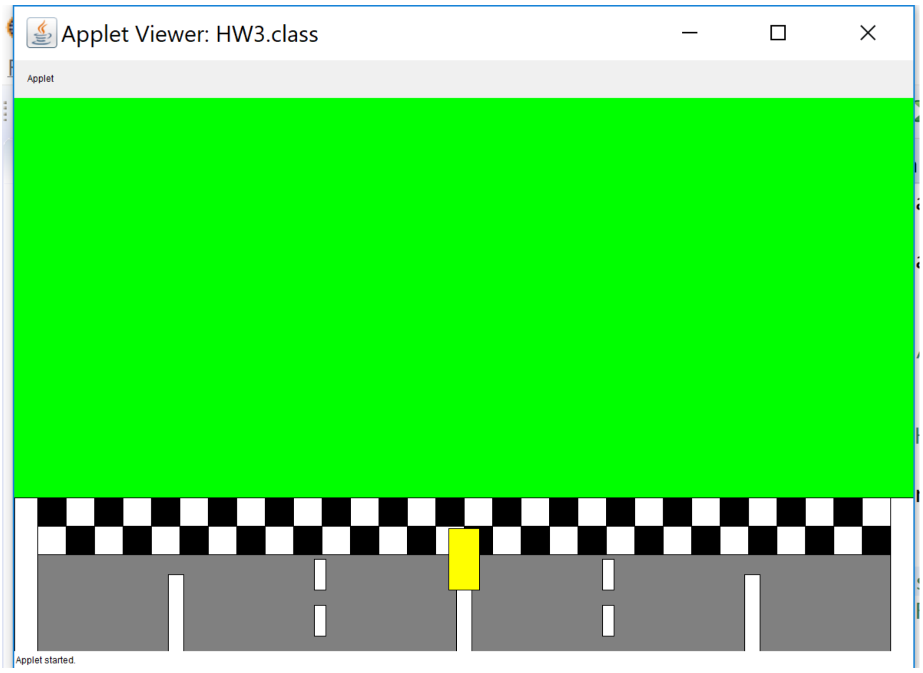


Figure 3: Finish Line

Nice, now you have a very good looking 2D game, you can proceed to the part where you will make the game winnable.

## 2.2 Time to Race

As the goal of the game is to reach the end of the map without hitting any of the obstacles on the way, you are expected to implement the movement and collisions happening throughout the game.

### 2.2.1 Movement

Imagine you are playing that famous game, *Road Fighter*, and struggling with that annoying red cars. This project does not have *horizontally moving obstacles*; however, movement mechanics are very similar to that. Instead of using move method for car, you will move the gameMap using move method. This will create an illusion that your car is moving upwards the road. Because gameMap is a GCompound that contains all elements except the car object, when you call move method on gameMap, you will see that all elements are moving at the same time. To observe the movement, after moving the gameMap one time, you should pause for PAUSE_TIME before moving again.

Note that car's movement to right and left has already been handled in the given code. When you run the project, you can move the car using left and right keys of the keyboard. That's why in this part, you only implement the movement on y-axis.

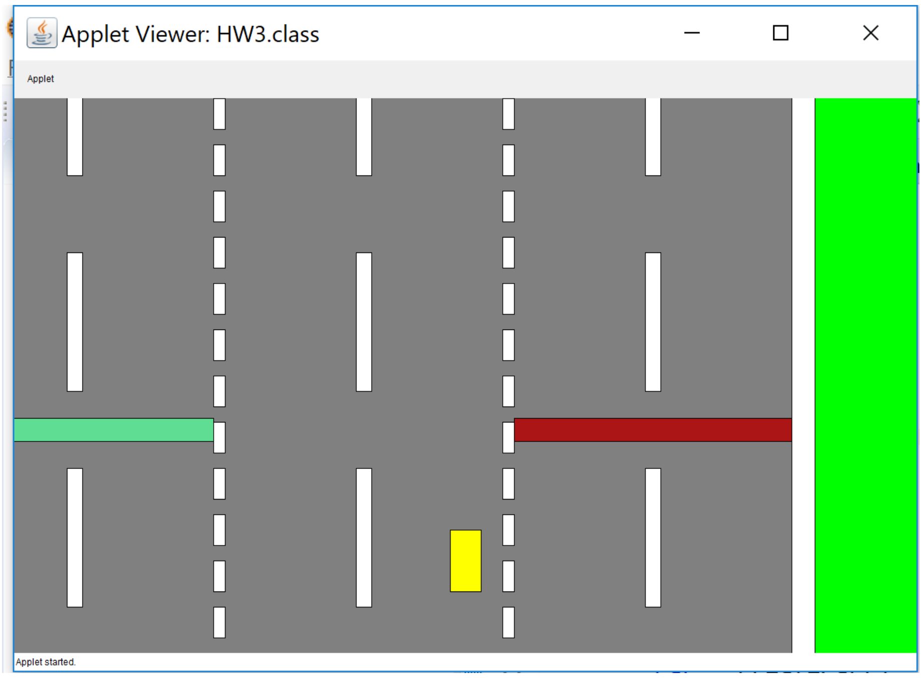


Figure 4: Game Play with Obstacles

### 2.2.2 Hit or Pass?

After you achieved the proper movement of the map, you must focus on checking whether the car is colliding with any obstacle. There are considerable amount of different ways to check what is present at a given coordinate on canvas, however, we strongly recommend you to follow given implementation technique carefully.

Thanks to implementation of the moving map, now we know that map's location is changing continuously to downward. This implies one critical fact that, car's relative position to any object on the gameMap is changing as well. That's why you must continuously check for the current position of the car to see if there is another object on the map that could cause a crash. This continuous control process can be done with the following steps:

1) Since gameMap does not contain the car, you must convert car's location to a *local point* that gameMap can understand and work on.

**Hint:** Knowing the usage of **getLocalPoint(GPoint gp)** for a GCompound would be an advantage for getting a local coordinate.

2) After getting the *local point*, you must get the object, or simply the element, on the map on that local point .

**Hint:** Knowing the usage of **getElementAt(GPoint gp)** would be helpful for getting the element for the specified coordinate of GCompound.

3) After getting the object that has the same coordinates as the car's on the

gameMap, you must check whether this object is an obstacle or another object that would not cause any problem for us. This control can be done with the following code trick:

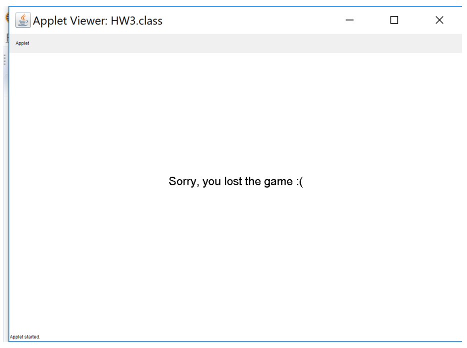
```
if (gameMap.getAllObstacles().contains(object)) {
    // Your code here
}
```

4) You must repeat previous steps until either reaching finish line, or finding an object with car's coordinate on the map, or simply hitting an obstacle. For both cases, race must be terminated, and the program must be able to show proper results as explained in next part.

***Now you are ready to race!***

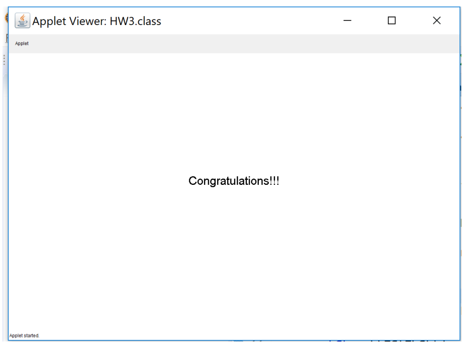
### 2.2.3 Finish

Any kind of race must be followed by a mighty end. To win the game, the only thing player must do is not to hit any of stationary obstacles before the finish line. If this is achieved, your program must congratulate the user. However, if player fails to avoid an obstacle, the game must be terminated, and indicate the loss. In this part, you should fill in the **finishGame()** method. You should create a message using GLabel. If player wins the game your message should be "Congratulations!!!", if player loses the game it should be "Sorry, you lost the game :(". You should check if player has finished the game successfully to decide which message to show. You should remove all elements from the application window using **removeAll()** method, then add the message to the application window. Message should be at the center of the application window.



Sorry, you lost the game :(

(a) Fail Message



Congratulations!!!

(b) Congratulation Message

## 2.3 End of Project

Your project ends here. You may continue to tinker with the code to implement any desired features and discuss them with your section leader. However, do not include any additional features that you implement after this point in to your submission.