# BLG 252E - Object Oriented Programming - Project #1

Student Name: *Mehmet Fatih Gülakar*

Student ID: *040150015*

Instructor: *Tankut Akgül*

### *A.) Project Goals*

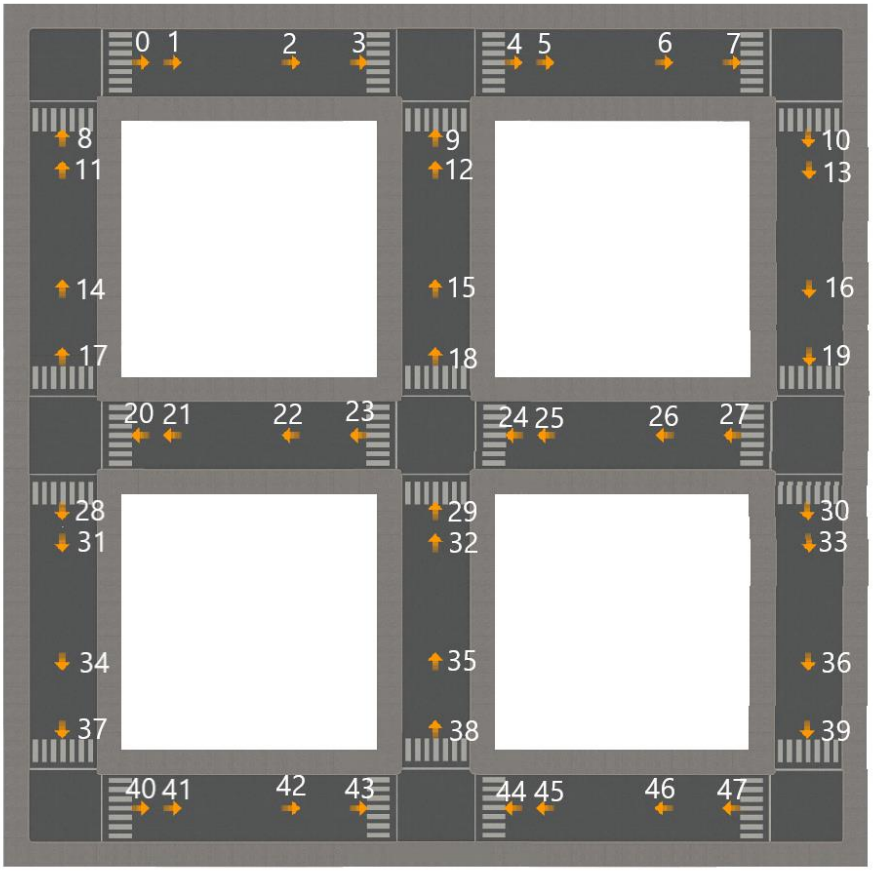
Purpose of this project was implanting a traffic simulator using object oriented concepts. SFML (Simple and Fast Multimedia Library) is used for drawing object to screen. In this first part of project, only a car and moving it around a predefined city is required.

### *B.) Team Members*

Project was entirely implemented by myself.

### *C.) Implementation*

First of all, structure of city is designed. Waypoints and their global indexes are shown below.



Class implementations;

1- **RoadTile**: class has a constructor and a method, which are easy to implement. As size of each tile is known, coordinates of upperleft point of a tile can be calculated easily. To fit whole map in 1600x1200 window, margins are selected as 205 and 3 pixels for X and Y direction, respectively. Coordinates are calculated as:

$$X = x_{margin} + RowNumber * TileSize$$
$$Y = y_{margin} + ColNumber * TileSize$$

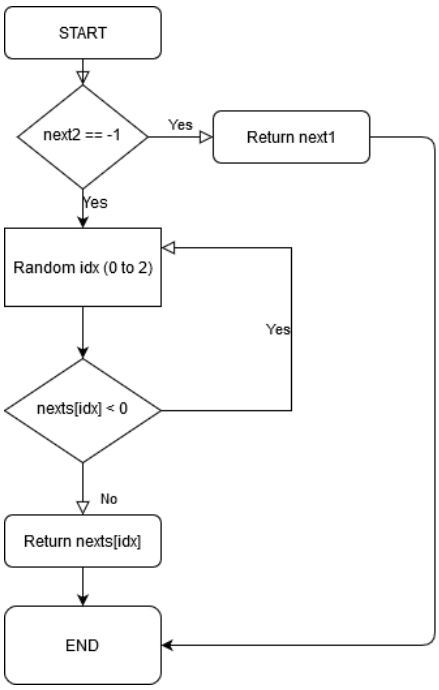
As RowNumber and ColNumber are [0, 4]

Texture and sprite of object are determined according to its type given as constructor parameter. Drawing a road tile to screen is easy since coordinates of it are calculated during initialization.

2- **Waypoint**: most of the member variables of waypoint object are determined by its constructor parameters. Important point in this class is the calculation of global x and y coordinates of it. First, an X and a Y coordinate are calculated based on Road Tile type waypoint belongs to, and the local index of waypoint. Then, global coordinates are calculated since we know row and column of waypoint.

$$X = x_{internal} + x_{margin} + RowNumber * TileSize$$
$$Y = y_{internal} + y_{margin} + ColNumber * TileSize$$

One of the important point is setting origin of sprite as center of texture. If we don't do that, position of waypoint will not be completely accurate.
getNext() method of Waypoint class is very important. For our selection algorithm, default values of next1, next2 and next3 values in constructor is defined as -1. Therefore, during initialization, writing only existing next directions would be enough. So, our goal is randomly selecting a positive index among these "next" values.




In getNext() method, next2 value is checked first. It is -1, that means next3 is also -1. So, there is no need for further processing, next1 is returned. If next2 is not -1, that means at least two values of "nexts" are non-negative.

To select among them, next1, next2 and next3 values are stored in an array. An index is selected randomly until corresponding value is non-negative. To select a random index, C++11's feature <random> library for pseudo-random number generator is used. It may be redundant for this project, but it is recommended more than rand(), since rand() has some flaws such as its bias and it contains division, which is not performance-friendly.

To encapsulate RoadTile and Waypoint classes, I created a class called City, which basically contains all RoadTile and Waypoint objects and some methods.

3- **City**: city classes has two private vector object called r and w, which hold pointers to RoadTile and Waypoint objects, respectively. Also, for the waypoints, global index of them are the same as index of vectors in class. Since they are private, programmer should access their methods through City's methods. To do that, there is a method called getNextPosition(), which returns the position and index of destination, given the current index.

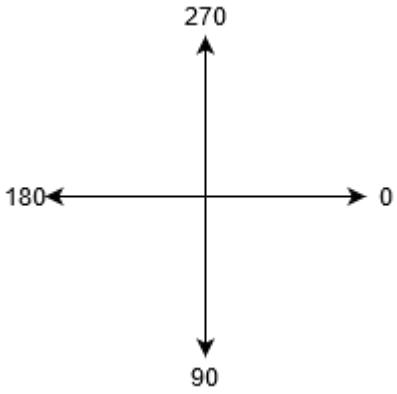
4- **Vehicle**: Constructor of Vehicle class is nearly same as other classes.

For the non-smooth version of move() method, SFML's setPosition() and setRotation() are enough.

In smooth version of it, there are two different cases. These are going straight and turning around corner.

Straight movement of car object is simple. Unless objects's X coordinate is close enough to destination, vehicle goes X_STEP right or left, depending on whether X coordinate is lower or higher than destination's X. Same goes for Y coordinate. For each step, sprite's position is updated.

Turning corner is more complicated than going straight. During circular movement, all X and Y displacements are calculated using angle difference, called ANGLE_STEP. Therefore, vehicle's angle is altered depending on whether it is lower or higher than destination's direction. SFML's coordinate system is clockwise.

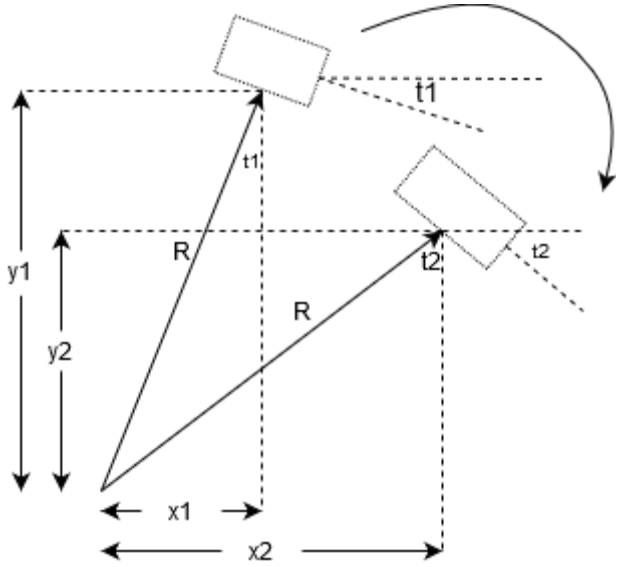



For angles between 0 and 270. Same procedure as X/Y can be followed. For others, there are two different cases.

- Vehicle's direction was 270 degrees; it will turn to 0 degree: In that case, using the same procedure as other angles will be wrong, since for that procedure, car will turn 270 degrees, instead of 90 degrees. So, instead of decreasing it, we increase the angle of until it becomes 360 degrees, it means car's direction is right and angle is adjusted by subtracting 360 degrees from it.
- Vehicle's direction was 0 degree; it will return to 270 degrees: In that case, instead of increasing the angle, we decrease it until it become -90, which is 270 degrees. When it becomes -90, it is adjusted by adding 360.

After calculating the angle_temp, which is higher/lower than angle of vehicle by ANGLE_STEP, is used to calculate the circular displacement. Displacement equations are different for turning right or turning left.

For turning back, x1, x2 and y1, y2 can be written as follows:




$$x1 = Rsin(t1), \qquad x2 = Rsin(t2)$$

$$y1 = Rcos(t1), \qquad y2 = Rcos(t1)$$

Therefore, difference between them, called offsetX and offsetY are equal to:

$$offsetX = x2 - x1 = R(\sin(t2) - \sin(t1))$$

$$offsetY = y1 - y2 = R(\cos(t1) - \cos(t2))$$

For other case, which is turning left, equations become exact opposite, since only direction of the displacement vector is changed:

$$offsetX = x1 - x2 = R(\sin(t1) - \sin(t2))$$

$$offsetY = x2 - x1 = R(\cos(t2) - \cos(t1))$$

After calculation of offsetX and offsetY, sprite's move method called to draw the car with its new position.

### *D.) Discussion*

Toughest problems I encountered was the angle rollover problem while calculating new angle value. I sorted it out by checking different cases of turn. Also, calculation of offsetX and offsetY was difficult. By keeping the last angle before turn (turn_direction class member), I solved it too.

I think project can be improved in a way that, instead of calculating initial position of car for its constructor, constructor only takes on which waypoint car will be created. Moreover, instead of using Vehicle's checkDest() method, a more suitable way could be used for better object oriented design.