BLG252E Object Oriented Programming Project 1

1. Objective

The objective of this project is to implement a traffic simulator using object oriented programming methods. The simulator will simulate an urban environment with various vehicles moving in traffic.

In this first phase of the project, you are expected to draw the environment and move a single car around the track. You can work in teams of two students.

2. Requirements

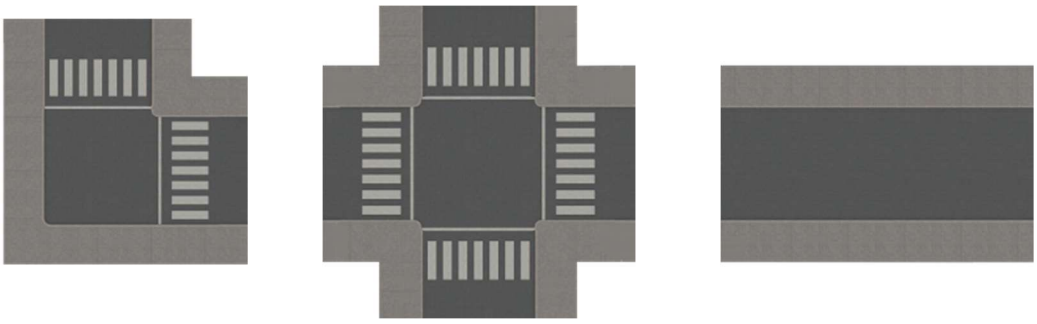
The project will be implemented using SFML Graphics and Multimedia library (see SFML section for more information). The library binary and header files will be provided to you.

If you use a Windows machine, you may use Dev-C++ as your build environment (see compiler section for more information). For Mac and Linux, you can use a suitable C++ compiler such as gcc.

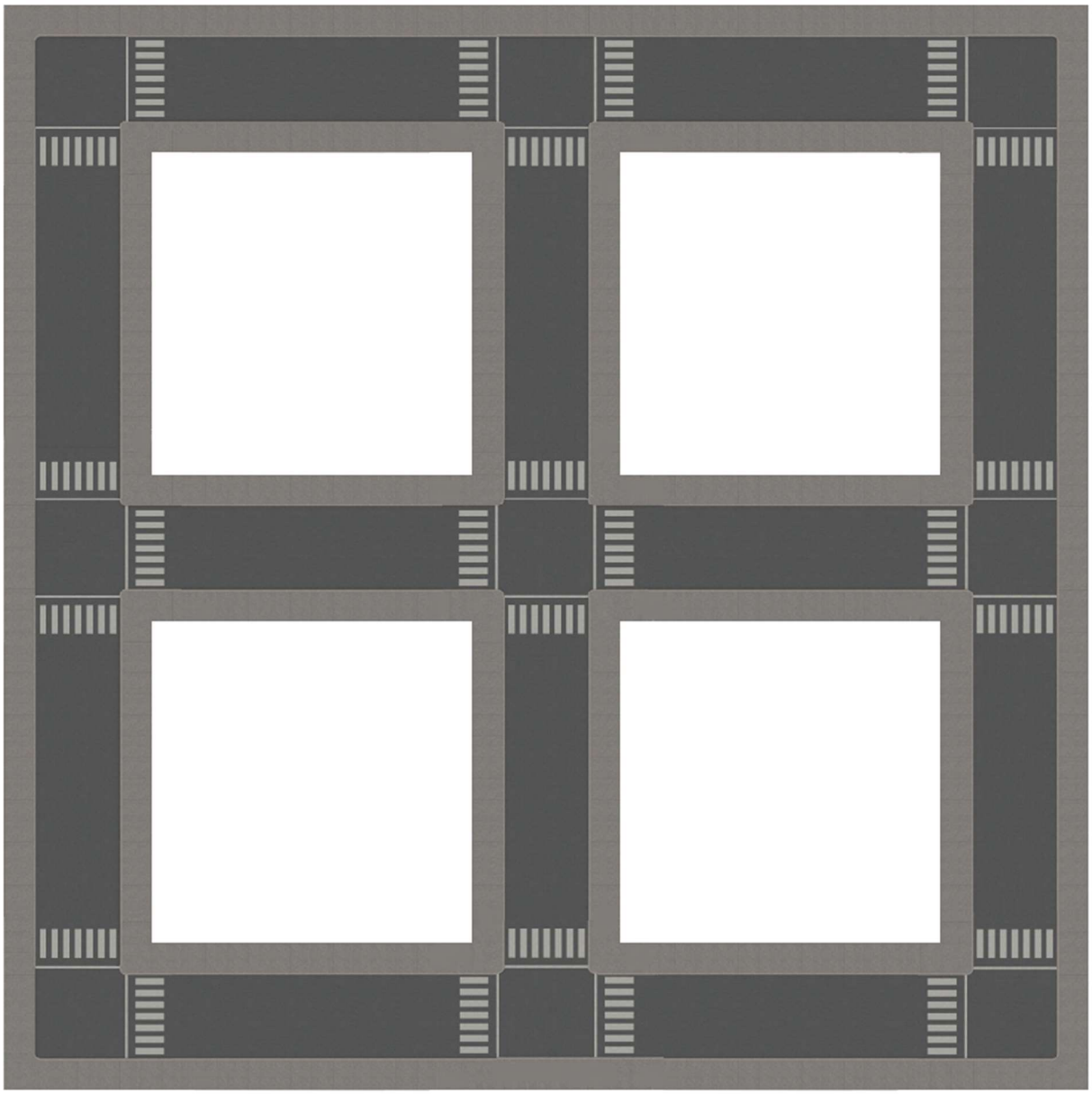
3. Project Description

The simulator graphics will be drawn using a set of sprites. A sprite is a 2D graphic drawn at a particular location on screen. There will be a sprite for the following graphics objects:

RoadTiles



The roads will be drawn by combining road tiles like above in a particular way just like building a jigsaw puzzle. The roads must be continuous and must form a full circuit. See below for a complete circuit example. You are allowed to come up with your own circuit the way you wish, but your circuit should have at least one crossroad where two roads intersect.



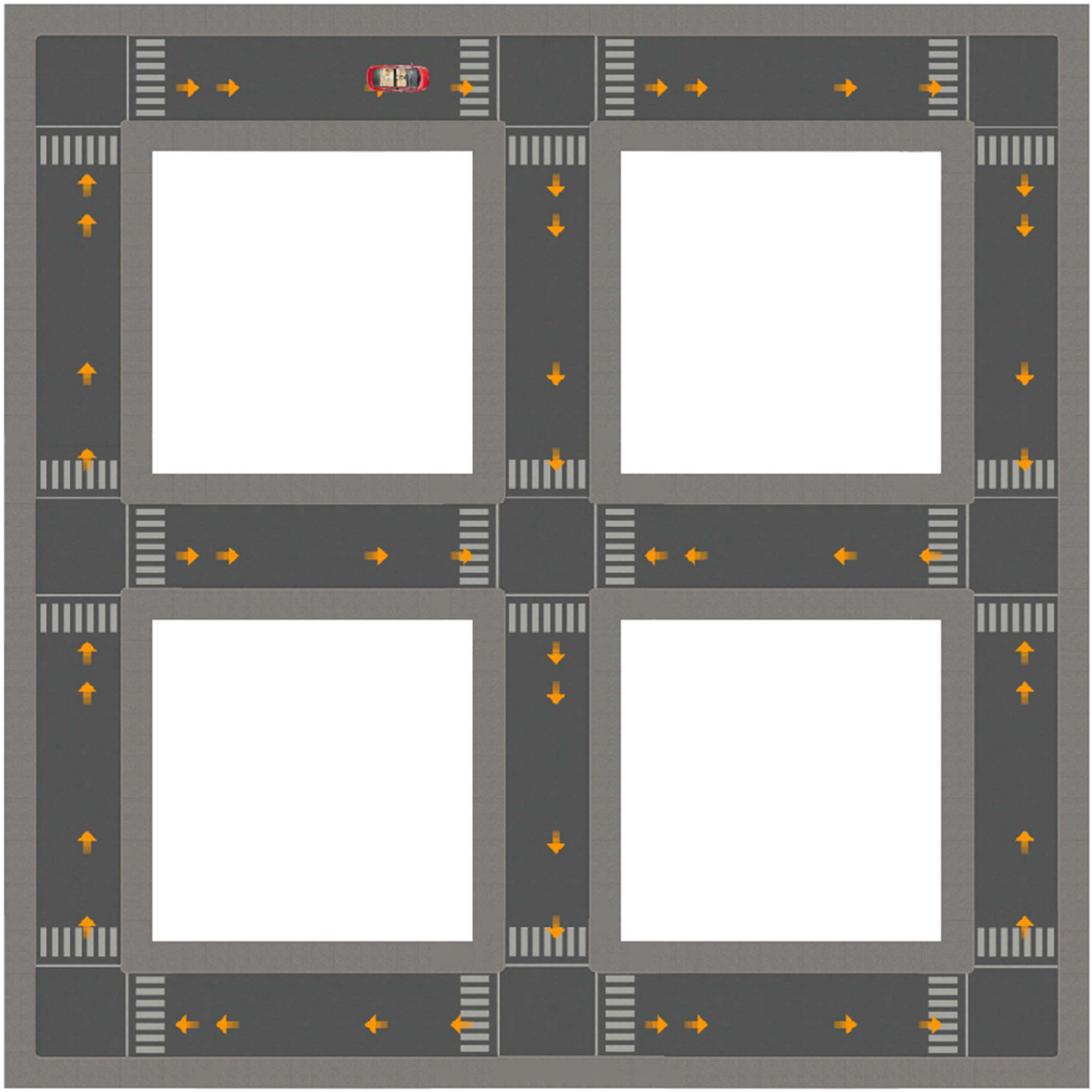
Cars

Your simulation should have one car sprite like below that moves around the circuit.

Waypoints

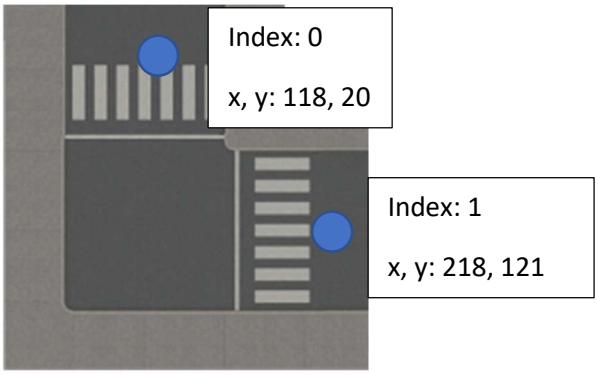
Your simulation should have waypoints that show the points in the track that the car will follow and the direction the car may go. Below are the sprites for the waypoints.

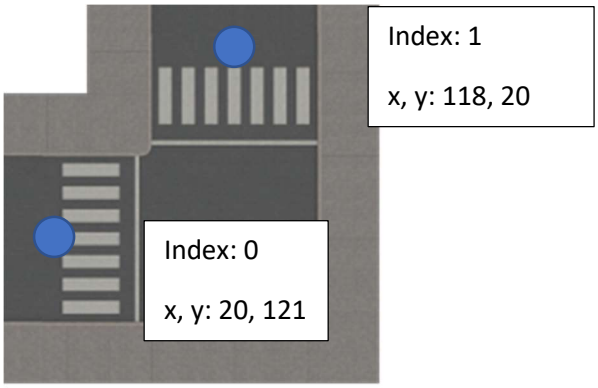
On your final simulation, you should display the waypoints on the track as below.

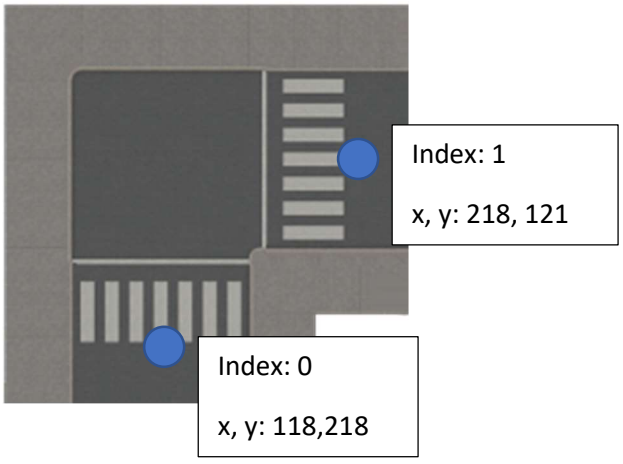


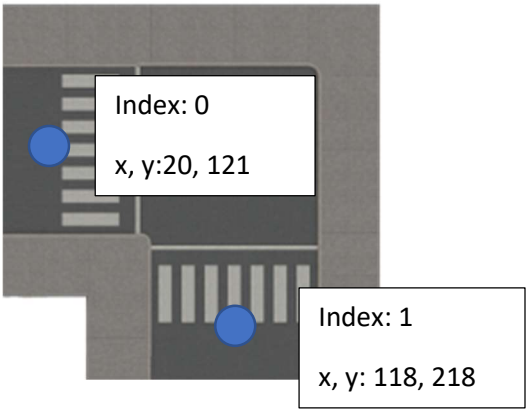
3.1 Road Tiles and Waypoints

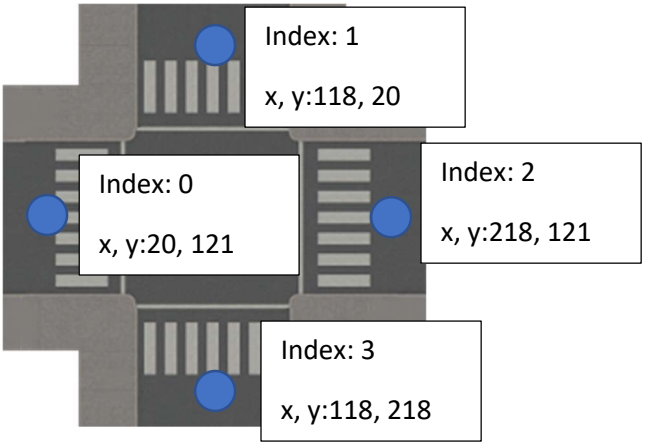
Every road tile is a 239 by 239 pixel graphic sprite. Each tile has a set of two or more waypoints on it. Each waypoint has an internal index starting from 0 within the tile it belongs to as well as internal x-/y- coordinates relative to the top-left corner of the tile. Below are all available tiles and the waypoints belonging to them along with the internal waypoint indexes and their relative coordinates with respect to the top-left corner of the tile.

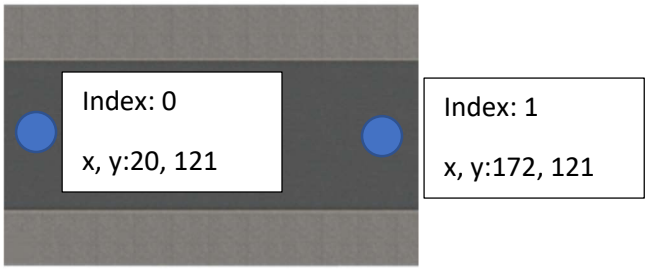


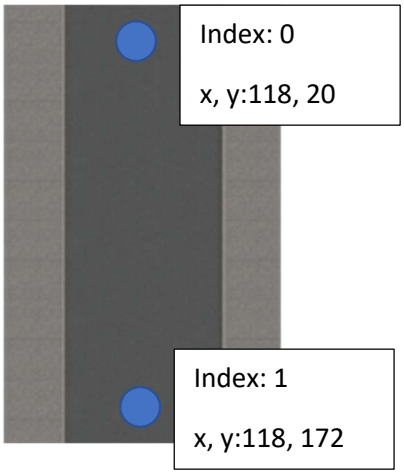


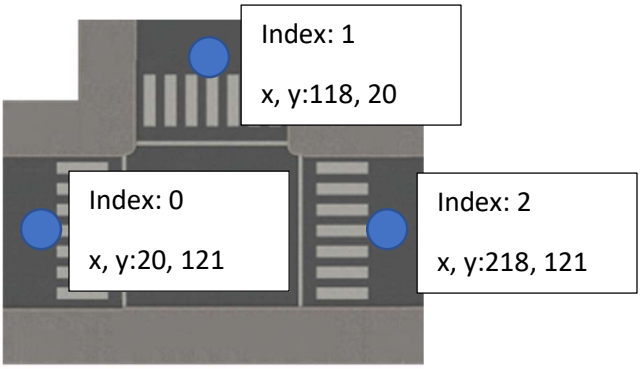


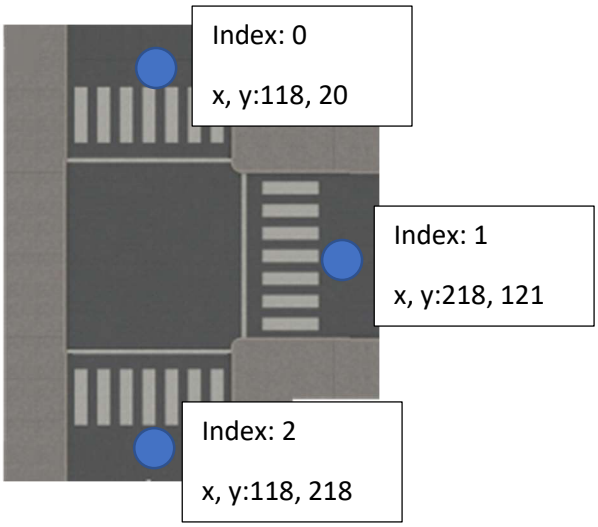

Index: 0
x, y:20, 121
Index: 1
x, y: 118, 218

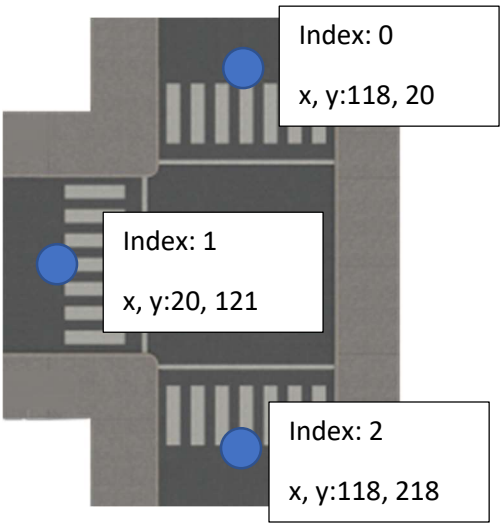

Index: 1
x, y:118, 20
Index: 0
x, y:20, 121
Index: 2
x, y:218, 121
Index: 3
x, y:118, 218

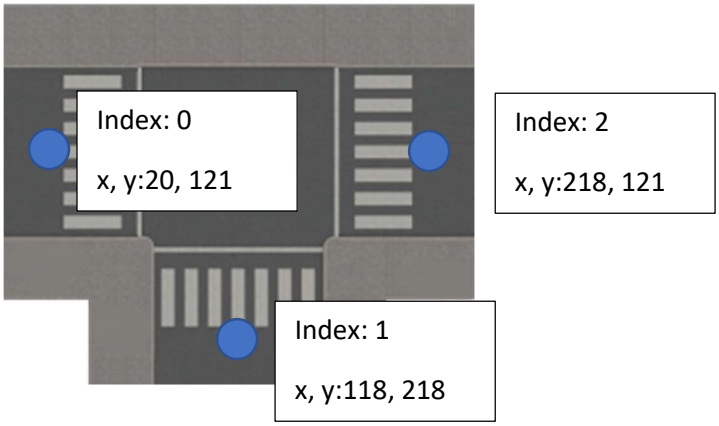

Index: 0
x, y:20, 121
Index: 1
x, y:172, 121


Index: 0
x, y:118, 20
Index: 1
x, y:118, 172


Index: 1
x, y:118, 20
Index: 0
x, y:20, 121
Index: 2
x, y:218, 121


Index: 0
x, y:118, 20
Index: 1
x, y:218, 121
Index: 2
x, y:118, 218


Index: 0
x, y:118, 20
Index: 1
x, y:20, 121
Index: 2
x, y:118, 218


Index: 0
x, y:20, 121
Index: 2
x, y:218, 121
Index: 1
x, y:118, 218

3.2 Classes

You should implement the following classes:

1. RoadTile

Private Members:

- float x; //x coordinate of the top left corner of the road tile
- float y; //y coordinate of the top left corner of the road tile
- sf::Texture texture; //tile texture object
- sf::Sprite sprite; //tile sprite object

Public Methods:

- RoadTile (tRoadTileType t, int row, int col);
  Constructor for the RoadTile class
  parameters:
  - row: Row number for the tile
  - col: Column number for the tile
  - t: Tile type (one of the available tile types, such as left t-junction, cross-junction, top-left corner, etc.)
- void draw();
  Draws the tile to screen

2. Vehicle

Private Members:

- float x;
- float y;
- float angle;
- bool origin_set;
- sf::Texture texture; //vehicle texture object
- sf::Sprite sprite; //vehicle sprite object

Public Methods:

Vehicle(tVehicleType t, float x, float y, float angle);
Constructor for the Vehicle class
parameters:
- t: Vehicle type (one of the available vehicle types)
- x: x coordinate of the current vehicle position
- y: y coordinate of the current vehicle position
- angle: vehicle heading angle, i.e., orientation

void move(float &x, float &y, float &angle);
Moves the vehicle and draws it to screen at the new location
parameters:
- x: x coordinate to move the vehicle to
- y: y coordinate to move the vehicle to
- angle: new vehicle heading angle after the move

3. Waypoint

Private Members:

- float x; //Global x coordinate of the waypoint
- float y; //Global y coordinate of the waypoint
- int dir; //direction of the waypoint (one of the 4 available directions)
- int next1; //Global index of the first alternative next waypoint
- int next2; //Global index of the second alternative next waypoint
- int next3; //Global index of the third alternative next waypoint
- sf::Texture texture; //waypoint texture object
- sf::Sprite sprite; //waypoint sprite object

Public Methods:

- Waypoint(tWaypointDir dir, tRoadTileType type, int row, int col, int idx, int next1, int next2, int next3);
  Waypoint constructor
  parameters:
  - dir: Waypoint direction
  - type: Road tile type this waypoint belongs to
  - row: Row number of the tile this waypoint belongs to
  - col: Column number of the tile this waypoint belongs to
  - idx: Internal index of the waypoint within the tile it belongs to
  - next1: Global index of the first alternative next waypoint
  - next2: Global index of the second alternative next waypoint
  - next3: Global index of the third alternative next waypoint

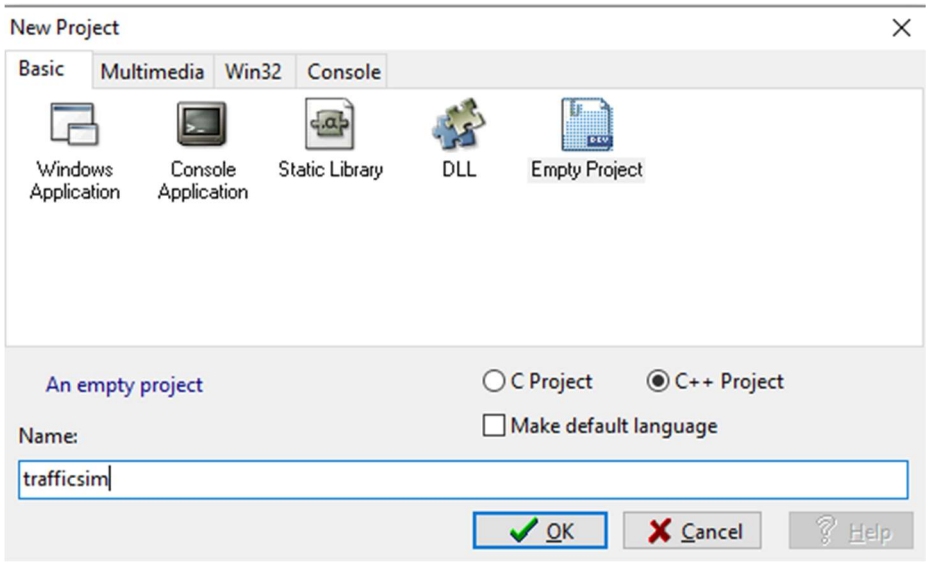
- int getNext();
  Returns the index of the next waypoint. If there are alternative waypoints, returns the index of one of them randomly
- void getPosition(float &x, float &y, float &dir);
  Returns the x, y coordinates and the direction of the waypoint
- void draw();
  Draws the waypoint to the screen

4. Compiler

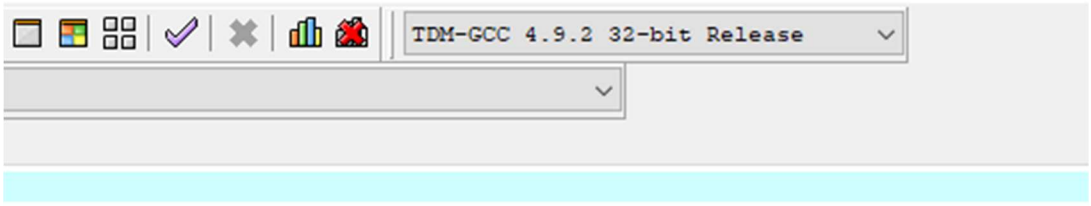
This section describes Dev-C++ setup for building the project.

1. From File menu of Dev-C++, select New and then Project…

2. In the New Project dialog box, select “Empty Project” and give “trafficsim” name to your project. Make sure C++ Project type is selected on the dialog box. Then, save the project to a folder of your choice.



3. SFML requires a 32-bit compiler, it won’t work with a 64-bit compiler. Therefore, in the main window, make sure **TDM-GCC 4.9.2 32-bit Release** version is selected from the dropdown box.



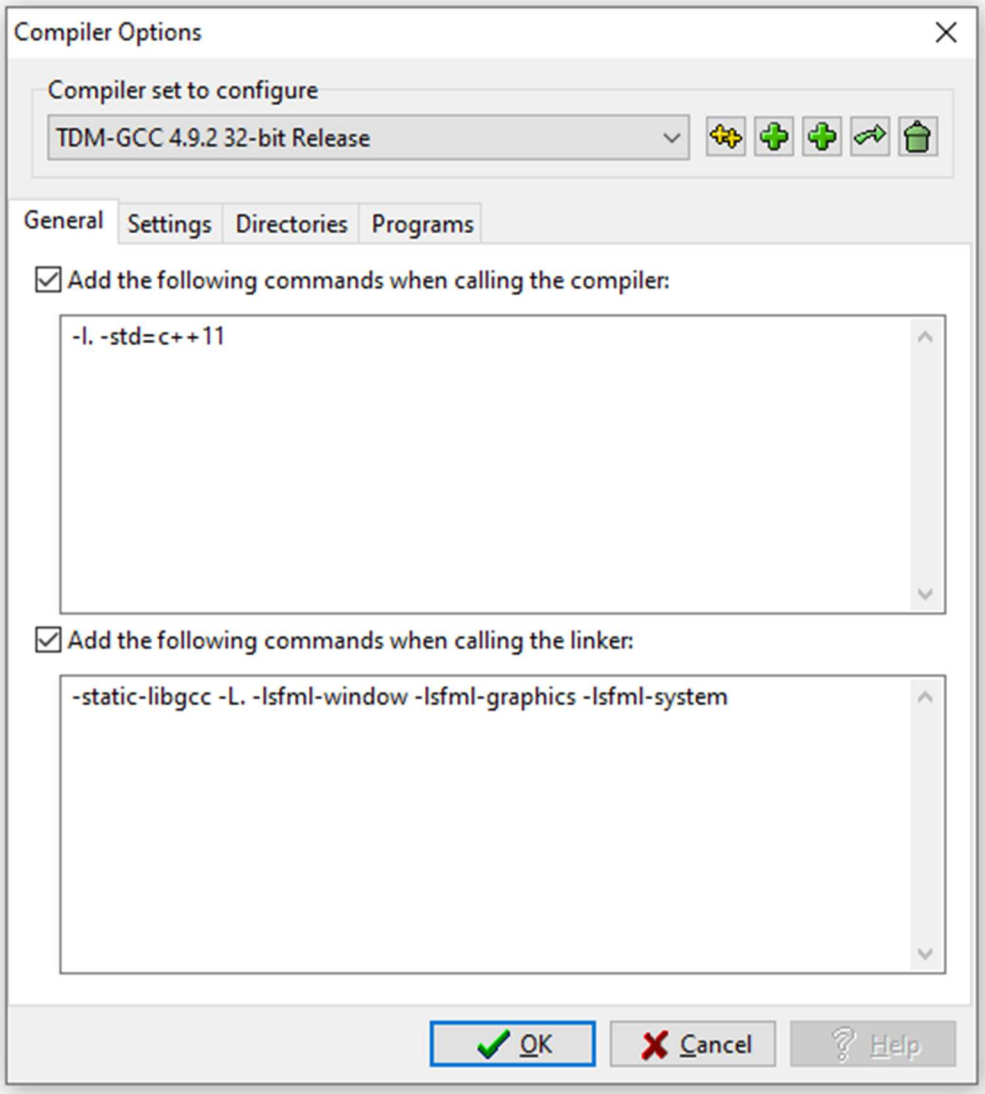
4. From Tools menu, select Compiler Options. Make sure **TDM-GCC 4.9.2 32-bit Release** is also selected as the Compiler set to configure in the dropdown box in Compiler Options dialog. In the dialog box, select the checkbox next to “Add the following commands when calling the compiler”. Enter the following line to the text box below this option:

```
-I. -std=c++11
```

Similarly, into the bottom text box, enter the following line:

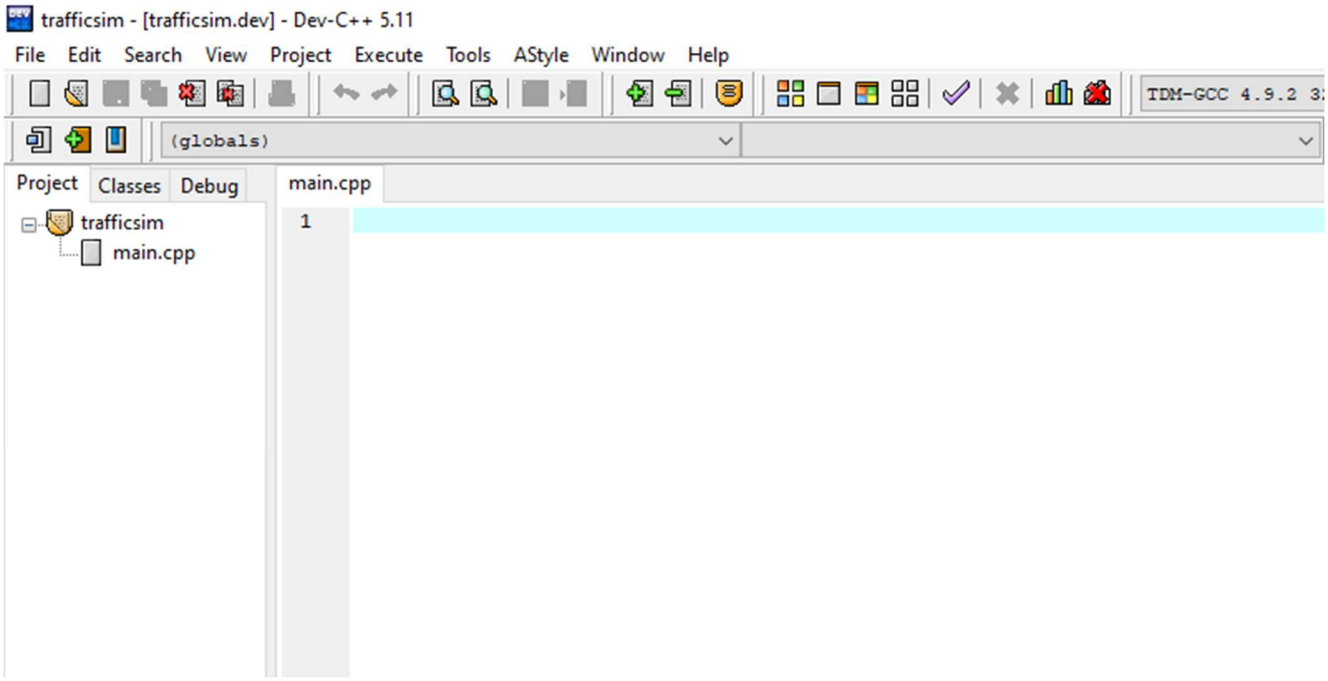
-static-libgcc -L. -lsfml-window -lsfml-graphics -lsfml-system

See blow figure for more information.

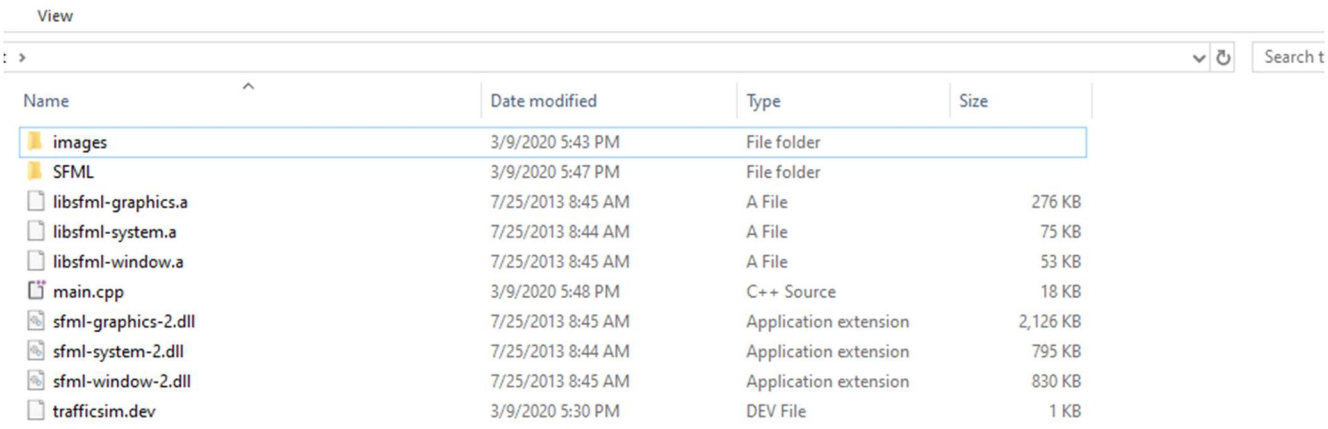


Click Ok to close the Compiler Options dialog box.

5. Save the source code as main.cpp as shown below.



6. Copy the contents of the shared project files at Ninova to the project folder you created at step 1. Your project directory should look as follows:



7. If you click Compile from the Execute menu, Dev-C++ will try to compile your code. Of course, we haven't written any code yet, and we need at least a main() function to get the code compile without errors. See next section for a minimalist SFML application.

5. Simple SFML Application

```
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <iostream>
using namespace std;

int main()
{
   sf::RenderWindow window(sf::VideoMode(800, 600), "Traffic Simulator");
   sf::Texture texture;
   if (!texture.loadFromFile("images/vehicles/car1.png"))
   {
      cout << "Could not find the image file" << endl;
   }
   sf::Sprite sprite;
   sprite.setTexture(texture);
   int x = 150;
   int y = 150;
   int increment = 1;

   //Normally sprite rotation reference is the sprite's top left corner. To rotate the sprite around its center point, we need to set the
   //rotation origin to the sprite's center. This is how we do it:
   //Get the bounding box coordinates for the sprite
   sf::FloatRect boundingBox = sprite.getGlobalBounds();
   //Set the sprite rotation origin to the center of the bounding box
   sprite.setOrigin(sf::Vector2f(boundingBox.width / 2, boundingBox.height / 2));

   while (window.isOpen())    //This is the main loop, the simulation should take place within this loop
   {
      // check all the window's events that were triggered since the last iteration of the loop
      sf::Event event;
      while (window.pollEvent(event))
      {
         // "close requested" event: we close the window
         if (event.type == sf::Event::Closed)
            window.close();
      }
      //Clear window
      window.clear(sf::Color::White);

      //Move car sprite to x,y position
      sprite.setPosition(x, y);

      //Draw the car sprite to screen
      window.draw(sprite);

      //Update the display
      window.display();

      //Change x-coordinate of the car sprite
      x+= increment;
      if (x == 250) {
         //if the car has reached x position of 250, rotate it by 180 degrees (car faces to the left now)
         sprite.setRotation(180);
         //the sprite now moves to the left
         increment = -1;
      }
      else if (x == 150)
      {
         //if sprite has reached back to x position of 150, rotate it back to 0 degrees (car faces to the right now)
         sprite.setRotation(0);
         //the sprite now moves to the right
         increment = 1;
      }

      //Wait 0.01 seconds before looping again
      sf::sleep(sf::seconds(0.01f));
   }
   return 0;
}
```

The above code moves a car horizontally back and forth between x-coordinates 150 and 250 on screen. You should see the following window if you compile and run this code.

5. Project Deliverables

Your submission should include a zip file containing a project report and the contents of your project folder.

1. You need to add comments to your code. Uncommented code will get partial credit. Be reasonable with the number of comments you add. Do not try to comment every line.
2. DO NOT submit files individually. Put them into a compressed zip archive and submit the zip archive. Name your zip archive with full name of your team members such as ahmet_bilir_and_veli_yapar.zip
3. Submit your homework zip file as an attachment to Ninova.

Below is the rubric for the project report.

Introduction

Briefly describe the project goals here

Team Members

Name the project team members and specify their roles in the implementation of the project.

Implementation

Here, you should describe the classes you implemented, how you calculated the position for the road tiles as well as waypoints, and how you move the car along those waypoints.

Discussion

Briefly describe the problems you faced in the implementation of your project and how you could improve it.