BLG 252E - Object-Oriented Programming - Project #2

Student Name: *Mehmet Fatih Gülakar*

Student ID: *040150015*

Instructor: *Tankut Akgül*

A.) Project Goals

In the second project, the goal was adding traffic lights and a data structure to control them easily. Moreover, adding at least 6 vehicles, while avoiding collision between them was required. SFML library is for drawing objects as we did on the first project.

B.) Team Members

As in the previous project, all codes are written by myself.

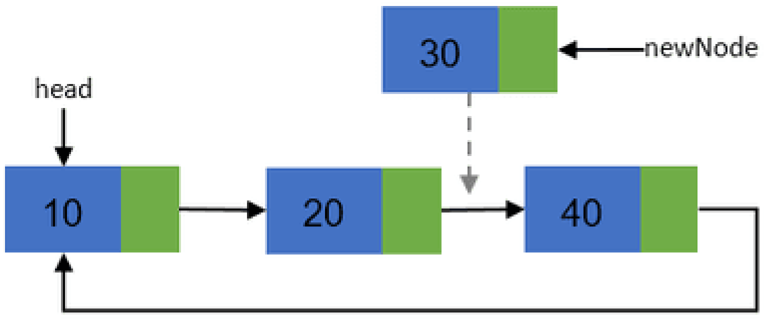
C.) Implementation

Different from the previous project, the random number generation method is changed. This project is developed using Visual Studio 2019. There was not any randomness problem while debugging on VS. But when I tested it with TDM-GCC compiler, I realized getNext() method was not random. Therefore, instead of using C++11's random_device & mt19937, I am now using rand() function to generate a random index.

First of all, TrafficLight & TrafficLight group classes are added to the project.

1- TrafficLight: It has x, y and dir attributes as RoadTile & Waypoint objects have. After loading textures for green and red light, its state is determined by the parameter given in the constructor. Also, each traffic object has pointers to another TrafficLight object and a Waypoint object. <u>TrafficLight pointer</u>: It is used for switching to green-light state to another TrafficLight in TrafficLightGroup. The process is explained in detail in TrafficLightGroup. <u>Waypoint pointer:</u> For the second project, I added an attribute to Waypoint class called "canDrive", which determines whether car on the waypoint can leave, and get/set methods for it. Each TrafficLight object is bound with a waypoint object in its constructor. When a traffic light's state is red, it makes its waypoint's canDrive attribute false, allowing to stop the car at red light. Similarly, when the state is green, canDrive becomes true, so the car can pass over it easily.
   The default value of canDrive is true for all waypoints, so it can be changed only with traffic lights.

2- TrafficLightGroup: It is a simple circular linked list that holds TrafficLight objects. add() method is working as follows (inserting to circular linked list):
   - Check if the list is empty (head is NULL). If yes head becomes object to be added, and its next pointer is pointing to itself.
   - If the list is not empty (head is not NULL), create an iterator, then advance it until its next pointer is the head pointer. So iterator points to the last node (related to head pointer). Add it to the circular linked list

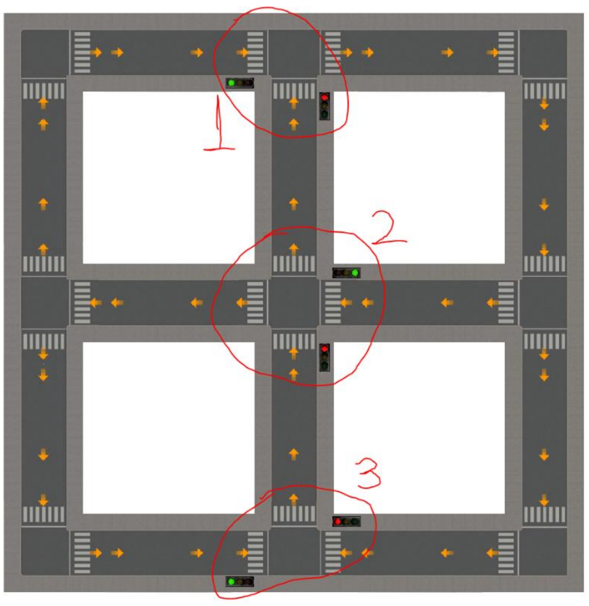


Before simulate() method of TrafficLightGroup, simulateAllLight method of the class I implement City works as follows:

- City class has a city_time attribute whose type is sf::Clock. City::simulateAllLights() takes deltaClock variable (created in main.cpp) as parameter. We first use deltaClock.getElapsedTime() to obtain how much time passed, called currTime. Then we calculate a deltaTime (float) variable by subtracting city's time from currTime. Therefore, we obtained a float variable to be passed to all TrafficLightGroup's simulate methods.
- Now, we have deltaTime, which is the time between two main game loops. simulate() method takes this value as a parameter, and adds it to TrafficLightGroup's "time" attribute. If "time" attribute is greater than "duration" attribute (which is switching time of TrafficLightGroup), the light pointed by greenLight pointer becomes red, then greenLight pointer points to next light. Then, new light's state is set as green. We do not need to bother with what to do when greenLight's next pointer is null since it cannot be null for the circular linked list.

Traffic Lights are declared in City class' constructor. Their coordinates are by determined by trial and error. Also, they are bound with waypoints just left of them (relative to the road).

Traffic Light Groups also declared in the same constructor. Their durations are decided as 5, 6 and 7 seconds respectively.

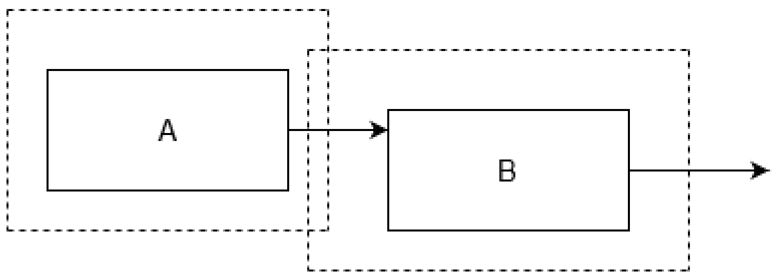


Adding Cars, Collision Detection and Smooth Movement

Additional cars could be added to my first project. However, main.cpp could be messy for a large number of cars. To write more clean code, I wrote a method called City::moveAllCars(). This method does all movement calculations in a for loop. Therefore, code can be organized for a large number of cars. A 1D array of waypoint indexes and 2D vector of x, y and direction variables of cars should be passed to method also. Moreover, City class has a vector of Vehicle pointers to use at moveAllCars & checkAllCollision methods.

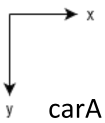
Collision detection was the hardest of the project for me. I wrote a method called Vehicle::detectedCollision(), which takes a pointer to another Vehicle object. It works as follow:

- Take sprites of carA & carB and store them in separate variables. Then scale them for COLLISION_SCALE (defined in utils.h)
- Use SFML's intersect method. It returns true if two sprites has collided with each other. Since we enlarged the sprites before that, we detected collision before cars collided.



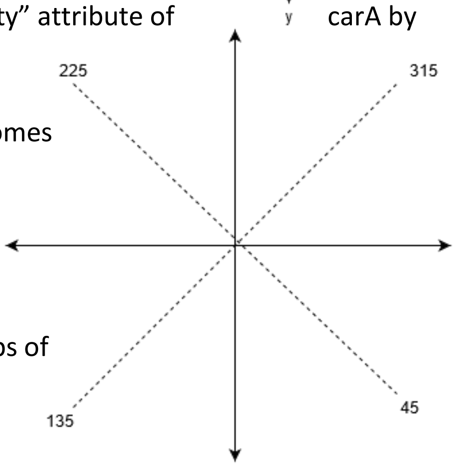
- If we detect a collision, we must determine which car to slow down. There are several cases:
  - If the angle is between [315, 45](right) and carA's x is smaller than carB's x.
  - If the angle is between [225, 315](up) and carA's x is greater than carB's x.
  - If the angle is between [135, 225](left) and carA's y is smaller than carB's y.
  - If the angle is between [45, 135](down) and carA's y is greater than carB's y.

  If any of these conditions are satisfied, that means carA(current object) is behind carB(passed object). So we need to decelerate carA. Now we should check if cars are going in a straight way or they are turning around a corner. We are doing this by std::fmod(), if the angle is multiple of 90 degrees, cars are going straight.



  If cars are going straight we decrease the "straight_velocity" attribute of carA by some multiple of ACCELERATION (declared in utils.h). To prevent the velocity to be negative and resulting in movement in the reverse direction, straight_velocity becomes 0 after a certain threshold MIN_SPEED.

  If cars are turning around a corner and they are going to collide. carA's "circular_velocity" is halved (Its maximum value is 1.0). Unlike the straight movement, circular_velocity does not become 0, to avoid sudden stops of cars.



If any of the conditions above are not satisfied (angle conditions should still be satisfied, but position conditions must be the negation of the first case), carB becomes car to be decelerated. Same processes above are done by the passed object.

Also, if we do not detect a collision, we reset the circular_velocity to 1.0.

City::checkAllCollision() method iterates over all cars pushed back to city's car vector. And it checks if a car collides with any other car.

- For regular acceleration/deceleration, we use the following way:
  - If the car was waiting at red light. Its "isStopped" attribute becomes true. So, when it starts to move, its velocity becomes 0.1, which results in acceleration.
  - Also, there is a MAX_SPEED, that limits the straight speed of the vehicle.
  - Instead of updating the current position with fixed X_STEP & Y_STEP, we update them with car's straight_velocity. So a car's step becomes smaller when it knows it will collide with another car. And gets larger for any other case.

Please note that there is no circular acceleration, but deceleration. A car's circular velocity becomes 1, which is max. value, when it does not collide with any other car object.

Also, when a car's next waypoint is bound with a red light. Its straight_velocity is divided by 1.25 until it arrives at the red light stop. This method allows us to decelerate cars smoothly.

Finally, there could be some uncovered cases for collisions. So you can come across bugs.

D.) Discussion

The toughest part I encountered was implementing the collision detection algorithm. I sorted it out by merging cases for circular and straight movement in one if condition. Also, designing a method that checks for both objects was difficult for me.

I think it can be improved by eliminating its movement bugs. This involves determining cases in which bugs occur and do something to cover it. Moreover, adding circular acceleration, which makes sense when a car was stopped at red light and will turn around the corner after the light goes green, is another feature to be added. Although, it may require some major changes in collision detection and handling algorithm.