BLG 252E - Object-Oriented Programming - Project #3

Student Name: *Mehmet Fatih Gülakar*

Student ID: *040150015*

Instructor: *Tankut Akgül*

A.) Project Goals

In the third and last project, the goal was implementing bus and bus stop classes while making Vehicle class as an abstract class.

B.) Team Members

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

C.) Implementation

The first phase of the project was changing the classes so that Vehicle class becomes an abstract class and the car we used in previous projects is separated as Car class. In previous projects, calculation of the next waypoint has been done by City class. Firstly, variables to hold destinations are put on Vehicle class instead of main.cpp. Secondly, Vehicles class is converted to an abstract class by changing the move method as

```
virtual void move(sf::RenderWindow& window) = 0;
```

Car class' members are same as Vehicle class implemented in previous projects. Unlike car, bus class has different attributes and methods from vehicle class.

- std::vector<int> stops : Vector of global indexes where a bus will stop
- std::vector<int> route: Vector of global indexes where buss will go through
- int currentStop: current Stop of the car
- int routePointer: pointer to select next_index from route vector
- std::vector<int> getStop: return stops vector of the object.
- Void setStop(): assign this->stops to vector of bus stop indexes given as a parameter.
- Void addPath(): push the index to route vector of Bus
- Void printPath(): prints route of the bus, used for debugging purposes.

Texture and sprite of vehicle object are determined on derived Car and Bus class constructors.

In move method of Car and Bus classes, followings are done:

- Check if the current waypoint allows being passed through. If yes, slow down.
- Check if the car has arrived at its destination waypoint. If yes, the following are done:
  For Car class:
  i) Take possible next indexes as a std::vector using getNexts() method. Choose a random one among them. Repeat this process until the chosen index is different from -1.
  j) Use the next index to change destination variables of Car object.

For Bus class:

i) Take possible next indexes as the same way as the Car class. Then, store the current index as a variable.
j) Obtain the next waypoint's global index by using route vector and route pointer. Increase the route pointer for next waypoint
k) If the obtained waypoint index is no different from the stored current index, decrease the route pointer by one, since the car has not arrived at its destination yet.
l) If routerPointer points to the last element of the route vector, assign it to zero. That means the bus completed its circular route.
m) Use the index to determine destination variables.

- After obtaining the destination, bus and car class move to the destination in the same way.

Determining the route of Bus:

First of all, the route of Bus is not predetermined. That means at first, the program calculates the shortest path between bus stops using the Breadth First Search algorithm. To implement that, I used some research on graph data structures.

An adjacency list, which is needed for finding the shortest path, is a vector of vectors. It stores from an index, which other indexes are possible to go. Constructing it was simple, thanks to the City object.

In the city object's constructor. Destination indexes are pushed to the corresponding index of the adjacency list. For example, the waypoints which are possible to go from the 24th waypoint are 18th and 23rd waypoints. Therefore,

AdjList[24] = {18, 24}

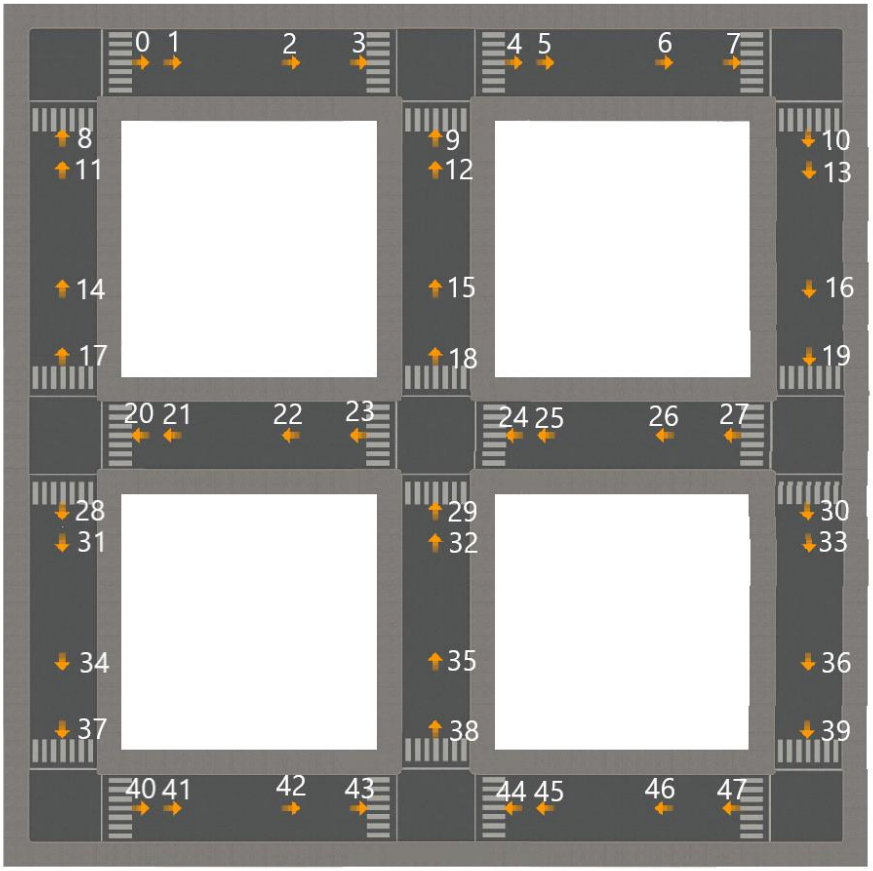
Then, I wrote the BFS method (with help, check resources), which is a helper function to find the shortest path. It is used for calculating the shortest path between two indexes, and works as follows:

- Initialize a queue, and three different arrays for storing distances, predecessors, and visited waypoints.
- Push the first index to queue, label start index as visited.
- While the queue is not empty:
  i) Pop x from the queue.
  j) Else, for all edges from x, if any edge is not visited, label it as visited. Change the destination array so that the xth index of it is the destination from the source to the xth index. Also, change the pred array so that the xth index of it is xth predecessor.
  k) Push the neighbor of x to queue. If it is the destination, return true.

As a result, bfs() method simply returns the array of distances from the start node to the destination node. Assigning this path to a bus is done in findShortestPath method. This method only takes a bus object pointer as a parameter. Then, gets its stops as a vector. The algorithm works as follows:

- For all consecutive elements of stops, find the shortest path.
- Since bfs() returns distances goes higher to as path goes to the destination, we store the subroute from waypointX to waypointY, and append it to bus' route in reverse order.
- Also find the shortest path from end waypoint to start waypoint, to form a cyclic path.

You can see the indexes of each waypoint to check if the bus goes on the correct path



Please note that buses do not stop at busStops but goes always on same route.

D.)Discussion

The toughest part I encountered was implementing the shortest path algorithm. But overcome it adapting a method on the Internet to the project. Standard requirements of the project were easy to implement since it uses already implemented methods. Moreover, I think this whole simulator we designed is not only a great way to understand OOP concepts, but also a great platform to implement algorithms we have learned in previous years of study.

BFS algorithm is implemented using the link below:

https://www.geeksforgeeks.org/shortest-path-unweighted-graph/