Design Studio #2 Part 2

Ankit Jain
Cory Matthew Cunanan
David Quang Bui
Michael C. Hernandez
Slavyana Naydenova Nedelcheva

TABLE OF CONTENTS		
NO.	**TOPIC**	**PAGE(S)**

Global Design Decisions

Design Requirements

- Lay out roads in pattern of user choosing, accommodate at least 6 separate intersections
- Specify and adjust behavior of traffic lights at each intersection
- Accommodate left hand turns protected by left hand green arrow lights
- Intersections must be 4-way
- Students must be able to add or remove sensors for intersections
- Change road density
- Run multiple simulations side by side to compare
- Two types of traffic: cars and trucks (trucks move slower than cars)
    - Cars could exhibit passing behavior

Design Ideas

- Only allow road closures at the beginning of the road object at an intersection
- Cars move twice as fast as trucks but both will take up the same amount of space for simplicity.

Design Goals

- Create an educational traffic flow simulator program.
- Create a visual map of an area, laying out roads in a pattern
- The visual map should be displayed to the user in real-time as he/she emerges in the situation.
- Specify the interaction mechanism that will control the behavior/pattern of the traffic lights at each of the intersections.
    - A variety of sequences and timing schemes should be allowed.
- The time that people spend at the traffic light should be minimized by the interaction mechanism that applies to that traffic light.
- Create a traffic light pattern that will allow cars to flow through the intersection from each direction in a fluid manner to avoid any crashes
- The visual map should display the current state and of the intersection traffic lights and update the change.
- The simulator would have an option to change the incoming traffic density by controlling how many cars enter the simulation at each of the roads on the edge. There are:
    - Busy roads
    - Seldom used roads
    - And in between those two
- Display any problems with map's timing scheme, be able to change them and display the changes.
- The application aims to simply illustrate the impact that traffic signal timing has on traffic.

- Determine changing the time with a few second what effect has on the next traffic light.

Primary Stakeholders

- **Professor E:** Having commissioned the project for her class, Professor E will likely be teaching the students how to interact with the simulation and draw conclusions based on those interactions.
- **Professor E's Students:** The students will be the audience interacting with the simulation the most, and they will have to be comfortable enough with it so that they can draw general conclusions about traffic based on their experience with the software.

Design Assumptions

- We are assuming the GUI portion of the simulation will correctly be able to give and transmit the information needed. For example, if they were to click on the map to create a road, that same road will be put into our data structure at the corresponding position.
- We also assume that they will handle the animations of the cars changing positions.
- We might assume that we can reuse an existing software package that provides relevant mathematical functionality such as statistical distribution, random number generators, and queuing theory.
- We are assuming that all vehicles slow down at a yellow light relative to the position of the car from the traffic light
- Assume all cars will be following US traffic laws, which means we are assuming the car can turn to the right on a red light.

Design Constraints

- Each simulation should consist of at least six separate intersections.
- Every intersection on the map must have traffic lights.
- There are no stop signs, overpasses, or other variations of these kinds.
- All intersections must be four-way, and no "T" intersections and one-way roads.
- All cars advance at the same speed.
- Pedestrians, crossing roads, bridges, accidents are not considered for this simulator
- All roads must be horizontal or vertical and there are no diagonal or winding roads.

<u>What are we keeping same as Design Studio #2 – Part 1?</u>

Data Structure

- We are maintaining the same data structure, using two-dimensional array to represent the map.
- Simulation classes will contain maps and the objects on maps; while running, the Simulation class will advance by one tick, advancing

Assumptions

- The vehicles can make left and right turns.

What is new in Design Studio #2 – Part 2?

Data Structure

- We are going to keep the same data structure, using two dimensional array, but how its components have been simplified to Intersections, Roads (that have 4 each, 2 going one direction and 2 going the opposite direction), and Left_Turn_Roads(Roads with 5 lanes, with the new 5th lane being in the middle and is the left turn lane).

Lane Closures

- If a road contains a lane closure, cars headed towards that road will automatically calculate a new direction and evaluate if the new road contains a closure as well.

Trucks and Cars

- As trucks move significantly slower than cars, cars behind trucks have the ability to accelerate and move past trucks.

Comparing Simulations

- We are going to allow for the Main Class to contain multiple Simulation classes and add more Simulation classes. In doing so, the user will be able to run multiple Simulation classes at the same time.

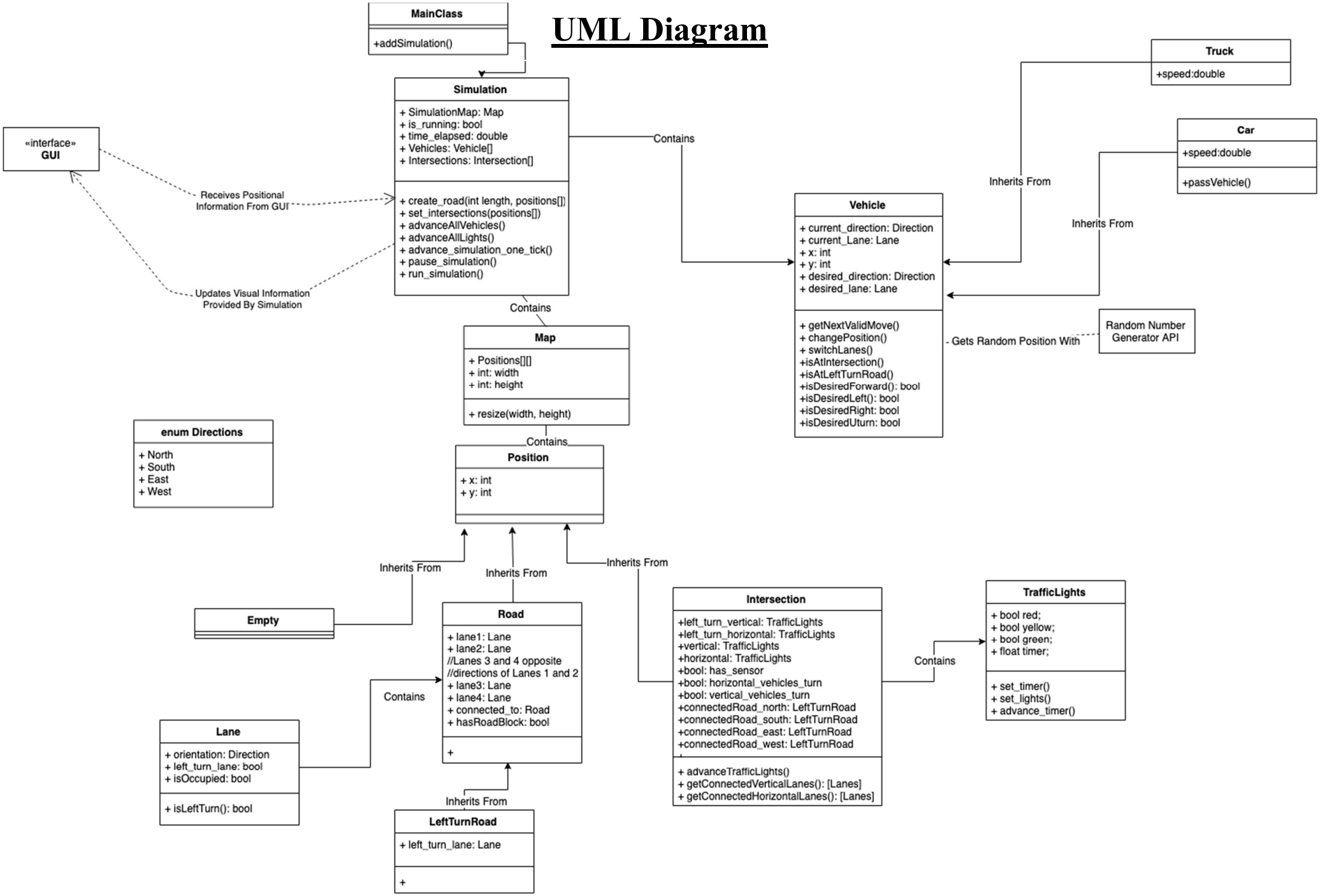
Saving Simulations

- Simulation progress can be paused at any given time by changing the is_running attribute of the Simulation class to False. However, the user can rerun a paused Simulation at any given time, since its state and attributes remain frozen.

Methods and UML

- We changed the main methods for moving the simulation one tick forward as the car behavior is now inherently different
    - All updated methods are shown in the pseudo code
- The UML has been updated to accommodate the new requirements

UML Diagram


MainClass
+addSimulation()
Simulation
+ SimulationMap: Map
+ is_running: bool
+ time_elapsed: double
+ Vehicles: Vehicle[]
+ Intersections: Intersection[]
+ create_road(int length, positions[])
+ set_intersections(positions[])
+ advanceAllVehicles()
+ advanceAllLights()
+ advance_simulation_one_tick()
+ pause_simulation()
+ run_simulation()
«interface»
GUI
Receives Positional
Information From GUI
Updates Visual Information
Provided By Simulation
Contains
Truck
+speed:double
Car
+speed:double
+passVehicle()
Inherits From
Inherits From
Vehicle
+ current_direction: Direction
+ current_Lane: Lane
+ x: int
+ y: int
+ desired_direction: Direction
+ desired_lane: Lane
+ getNextValidMove()
+ changePosition()
+ switchLanes()
+isAtIntersection()
+isAtLeftTurnRoad()
+isDesiredForward(): bool
+isDesiredLeft(): bool
+isDesiredRight: bool
+isDesiredUturn: bool
- Gets Random Position With
Random Number
Generator API
Map
+ Positions[][]
+ int: width
+ int: height
+ resize(width, height)
Contains
enum Directions
+ North
+ South
+ East
+ West
Position
+ x: int
+ y: int
Inherits From
Inherits From
Inherits From
Empty
Road
+ lane1: Lane
+ lane2: Lane
//Lanes 3 and 4 opposite
//directions of Lanes 1 and 2
+ lane3: Lane
+ lane4: Lane
+ connected_to: Road
+ hasRoadBlock: bool
+
Intersection
+left_turn_vertical: TrafficLights
+left_turn_horizontal: TrafficLights
+vertical: TrafficLights
+horizontal: TrafficLights
+bool: has_sensor
+bool: horizontal_vehicles_turn
+bool: vertical_vehicles_turn
+connectedRoad_north: LeftTurnRoad
+connectedRoad_south: LeftTurnRoad
+connectedRoad_east: LeftTurnRoad
+connectedRoad_west: LeftTurnRoad
+
+ advanceTrafficLights()
+ getConnectedVerticalLanes(): [Lanes]
+ getConnectedHorizontalLanes(): [Lanes]
Contains
TrafficLights
+ bool red;
+ bool yellow;
+ bool green;
+ float timer;
+ set_timer()
+ set_lights()
+ advance_timer()
Contains
Lane
+ orientation: Direction
+ left_turn_lane: bool
+ isOccupied: bool
+ isLeftTurn(): bool
Inherits From
LeftTurnRoad
+ left_turn_lane: Lane
+

Pseudo Code/Algorithm

Vehicles: changeLanes Method ()

```
changeLanes()
{
//loop method until current_lane == desired_lane

If(vehicle.current_lane == lane 1 or vehicle.current_lane == lane 2)
{
If(vehicle.is_inLeft_Turn_road)
{
        if(vehicle.desired_lane == lane 1)
{
        If(vehicle.current_lane == lane 2)
        {
                if(lane 1 is empty)
                {
                        Vehicle.current_lane = lane 1
                }
        }
}
Else if(vehicle.desired_lane == lane 2)
{
        If(lane 2 is empty)
        {
                Vehicle.current_lane = lane 2
        }
}
Else if(vehicle.desired_lane == road.left_turn_lane)
{
        If(vehicle.current_lane == lane 1)
        {
                If(lane 2 is empty)
                {
                        Vehicle.current_lane = lane 2
                }
        Else if(vehicle.current_lane == lane 2)
        {
                if(vehicle.road.left_turn_lane.is_empty())
                {
```

```
                    Vehicle.current_lane = road.left_turn_lane
                }
        }
}

}
else
{
if(vehicle.desired_lane == lane 1)
{
        If(vehicle.current_lane == lane 2)
        {
                if(lane 1 is empty)
                {
                        Vehicle.current_lane = lane 1
                }
        }
}
Else if(vehicle.desired_lane == lane 2)
{
        If(lane 2 is empty)
        {
                Vehicle.current_lane = lane 2
        }
}
}
}
If(vehicle.current_lane == lane 3 or vehicle.current_lane == lane 4)
{
If(vehicle.is_inLeft_Turn_road)
{
        if(vehicle.desired_lane == lane 4)
{
        If(vehicle.current_lane == lane_3)
        {
                if(lane 4 is empty)
                {
                        Vehicle.current_lane = lane 4
                }
        }
```

```
}
Else if(vehicle.desired_lane == lane 3)
{
	If(lane 3 is empty)
	{
		Vehicle.current_lane = lane 3
	}
}
Else if(vehicle.desired_lane == left_turn_lane)
{
	If(vehicle.current_lane == lane 4)
	{
		If(vehicle.lane_3.is_empty())
		{
			Vehicle.current_lane = lane 3
		}
	Else if(vehicle.current_lane == lane 3)
	{
		if(vehicle.road.left_turn_lane.is_empty())
		{
			Vehicle.current_lane = road.left_turn_lane
		}
	}
}

}
else
{
if(vehicle.desired_lane == lane 4)
{
	If(vehicle.current_lane == lane 3)
	{
		if(lane 4 is empty)
		{
			Vehicle.current_lane = lane_4
		}
	}
}
Else if(vehicle.desired_lane == lane 3)
{
```

```
        If(lane 3 is empty)
        {
                Vehicle.current_lane = lane_3
        }
}
}
}
}
```

Vehicles: passVehicle Method()

```
passVehicle()
//can only run on vehicles that are not of type truck
If(vehicle is not a truck)
{
If(next position has a vehicle that is not at max speed)
{
        if (in Lane 1)
        {
                desired_lane = Lane 2
        }
        Else if (in Lane 2)
        {
                 desired_lane = Lane 1
        }
        Else if (in Lane 3)
        {
                 desired_lane = Lane 4
        }
        Else if (in Lane 4)
        {
                 desired_lane = Lane 3
        }

vehicle.change_lanes()
}
}
```

Vehicles: getNextValidMove Method()

```
getNextValidMove()
{
```

```
//updates the desired lane for the vehicle
//makes sure that the vehicle won't go into a road closure in the highlighted section

valid_moves = Lane []

        if(vehicle is at an intersection)
        {
                If (desired_direction == North)
                {
                        Next_Road = road north of intersection
                        For (Lanes in Next Road)
                        {
                                if(Lane.direction == desired_direction)
                                Add Lane to valid_moves
                        }
                }
                Else if(desired_direction == South)
                {
                        Next_Road = road south of intersection
                        For (Lanes in Next Road)
                        {
                                if(Lane.direction == desired_direction)
                                Add Lane to valid_moves
                        }
                }
Else if(desired_direction == East)
                {
                        Next_Road = road east of intersection
                        For (Lanes in Next Road)
                        {
                                if(Lane.direction == desired_direction)
                                Add Lane to valid_moves
                        }
                }
                Else if(desired_direction == West)
                {
                        Next_Road = road west of intersection
                        For (Lanes in Next Road)
                        {
                                if(Lane.direction == desired_direction)
```

```
                                Add Lane to valid_moves
                        }
                }

        }
        Else if (vehicle is in the left turn road)

                //This section will specifically handle road closures
                //The last else will occur if there is a road closure, causing the car to randomly
                //pick another direction, then recursively call this function again so it recalculates
//with a new direction in mind

If ((desired direction is relative left || desired direction is a Uturn) && left turn lane
is same direction as current lane && Road in the relative left of
intersection is not closed off)
        {
                add left turn lane to valid moves
        }
else if (desired direction is relative right && Road in the relative right of
intersection is not closed off)
        {
                if (in Lane 1 or Lane 2)
                {
                        add Lane 1 to valid moves
                }
                else
                {
                        add Lane 4 to valid moves
                }
         }
        Else if (desired direction is forward and && Road in the relative forward of
intersection is not closed off)
{
        Add Intersection to Valid moves
}
Else
{
        Other directions = [All the directions except current desired direction]
        Randomly pick another direction
        getNextValidMove()
```

```
        break;
}
        Else
        {
                Else If (at Lane 1)
                {
                        Next Road = Connected Road at the current direction of(i.e West of, East
of, North of, South of) the current Road
                        Add the Next Roads Lane 1 to valid moves
                }
                Else If (at Lane 2)
                {
                        Next Road = Connected Road at the current direction of(i.e West of, East
of, North of, South of) the current Road
                        Add the next road Lane 2 to valid moves
                }
                Else If (at Lane 3)
                {
                        Next Road = Connected Road at the current direction of(i.e West of, East
of, North of, South of) the current Road
                        Add Lane 3 to valid moves
                }
                Else If (at Lane 4)
                {
                        Next Road = Connected Road at the current direction of(i.e West of, East
of, North of, South of) the current Road
                        Add Lane 4 to valid moves
                }
        }

//use random number generator to get a random index from valid moves
desired_lane = valid_moves[random index]
}
```

Vehicles: changePosition Method()

```
changePosition()
{
        getNextValidMove()
If (desired direction is the same as current lane's direction && in desired lane)
{
```

```
        If (next position is empty)
        {
                Move car to next position
        }
        If (next position is not empty && next position has a vehicle not at max speed && this
vehicle is a car)
        {
                passVehicle()
        }
}
Else if(not in desired lane)
{
        change_lanes()
}
Else if(next position is an intersection)
{
        If (currently in left turn lane)
        {
                if((direction == East || direction == West) && horizontal left turn light is green
&& Next position is empty)
                {
                        Move car to next position
                }
                Else if((direction == North || direction == South) && vertical left turn light is
green
&& Next position is empty)
                {
                        Move car to next position
                }
                Else
                {
                        \\this means that the lights are red so don't go or that the space is
\\occupied so it can't move
                        pass
                }

        }
        Else
        {
                if((direction == East || direction == West) && horizontal light is green &&
```

```
Next position is empty)
                {
                        Move car to next position
                }
                Else if((direction == North || direction == South) && vertical light is green
&& Next position is empty)
                {
                        Move car to next position
                }
                Else
                {
                        \\this means that the lights are red so don't go or that the space is
\\occupied so it can't move
                        pass
                }
        }
}
Else if(Vehicle is at an intersection)
{
        If (desired lane is unoccupied)
        {
                Move Car to next position
                Vehicles current direction = this Lanes direction
        }
        Else
        {
                //Space is occupied so it can't move
                Pass
        }
}
}
```

Vehicles: isDesiredLeft Method()

```
isDesiredLeft()
{
        Boolean Value = false
        If (current direction is North)
        {
                if(desired direction is West)
                {
```

```
                Boolean Value = true
            }
        }
        Else if (current direction is South)
        {
            if(desired direction is East)
            {
                Boolean Value = true
            }
        }
        Else if (current direction is East)
        {
            if(desired direction is North)
            {
                Boolean Value = true
            }
        }
        Else if (current direction is West)
        {
            if(desired direction is South)
            {
                Boolean Value = true
            }
Return value
}
```

Vehicles: isDesiredRight Method()

```
isDesiredRight()
{
        Boolean Value = false
        If (current direction is North)
        {
            if(desired direction is East)
            {
                Boolean Value = true
            }
        }
        Else if (current direction is South)
        {
            if(desired direction is West)
```

```
                {
                        Boolean Value = true
                }
        }
        Else if (current direction is East)
        {
                if(desired direction is South)
                {
                        Boolean Value = true
                }
        }
        Else if (current direction is West)
        {
                if(desired direction is North)
                {
                        Boolean Value = true
                }
Return value
}
```

Vehicles: isDesiredForward Method()

```
isDesiredForward()
{
        Boolean Value = false
        If (current direction is North)
        {
                if(desired direction is North)
                {
                        Boolean Value = true
                }
        }
        Else if (current direction is South)
        {
                if(desired direction is South)
                {
                        Boolean Value = true
                }
        }
        Else if (current direction is East)
        {
```

```
            if(desired direction is East)
            {
                  Boolean Value = true
            }
      }
      Else if (current direction is West)
      {
            if(desired direction is West)
            {
                  Boolean Value = true
            }
Return value
}
```

Vehicles: isDesiredUturn Method()

```
isDesiredUturn()
{
      Boolean Value = false
      If (current direction is North)
      {
            if(desired direction is South)
            {
                  Boolean Value = true
            }
      }
      Else if (current direction is South)
      {
            if(desired direction is North)
            {
                  Boolean Value = true
            }
      }
      Else if (current direction is East)
      {
            if(desired direction in West)
            {
                  Boolean Value = true
            }
      }
      Else if (current direction is West)
```

```
        {
                if(desired direction is East)
                {
                        Boolean Value = true
                }
Return value
}
```

Simulation: advanceTrafficLights Method()

```
advanceAllTrafficLights()
{
For (All Intersections)
{
If (isVerticalLanesTurn)
{

if(Vertical Left Lane Turn Traffic Light Timer > 0)
        {
                Decrease time for left lane turn traffic light timer one tick
                If (Timer == max_timer/3)
                {
                        Change Vertical left turn light to yellow
                }
                Else if (Timer == 0)
                {
                        Change Vertical left turn light to red
                        Change Vertical light to green
                }
        }
        Else if(Vertical Left Lane Turn Traffic Light Timer == 0)
        {
if(has sensor)
        {
                Bool decrease timer = false
                For (all connected horizontal lanes)
                {
                        If (lane is occupied)
                        {
                                Decrease timer = true
                        }
```

```
                }
                if(decrease timer == true)
                {
                        Decrease time for vertical traffic light
                        If (Timer == max_timer/3)
                {
                        Change Vertical light to yellow
                }
                                }
                Else
                {
                        pass
//Dont decrease time for vertical light since no one is there
                }
        }
        Else
        {
                Decrease time for vertical light
        }
        If(the vertical light time == 0)
        {
                VerticalLaneTurn = false;
                        HorizontalLaneTurn = true;
                        Reset Horizontal Left Turn Traffic Light timer
                        Reset Horizontal Traffic Light timer
                        Change Horizontal Left Turn Light Green
                }
}
}
Else
{
if(Horizontal Left Lane Turn Traffic Light Timer > 0)
        {
                Decrease time for left lane turn traffic light timer one tick
                If (Timer == max_timer/3)
                {
                        Change Horizontal left turn light to yellow
                }
                Else if (Timer == 0)
                {
```

```
                    Change Horizontal left turn light to red
                    Change Horizontal light to green
                }

        }
        Else if(Horizontal Left Lane Turn Traffic Light Timer == 0)
        {
if(has sensor)
        {
                Bool decrease timer = false
                For (all connected vertical lanes)
                {
                        If (lane is occupied)
                        {
                                Decrease timer = true
                        }
                }
                if(decrease timer == true)
                {
                        Decrease time for horizontal traffic light
                        if (Timer == max_timer/3)
                                {
                                Change Horizontal light to yellow
                                }

                }
                Else
                {
                        Pass
                                //Don't decrease time for horizontal light since no one is there
                }
        }
        Else
        {
                Decrease time for horizontal light
        }
        If(the horizontal light time == 0)
        {
                HorizontalLaneTurn = false;
                        VeritcallLaneTurn = true;
```

```
                    Reset Vertical Left Turn Traffic Light timer
                    Reset Vertical Light timer
                    Change Vertical Left Turn Light Green
                }
}
}
}
}
```

Simulation: advanceAllVehicles Method()

```
advanceAllVehicles()
{
For (all Vehicles)
{
        vehicles.changePosition();
}
}
```

Simulation: advanceOneTick Method()

```
advanceOneTick
{
        //advance the simulation one tick
        moveAllCars()
        AdvanceTrafficLights()
        Time_elapsed++
}
```

Simulation: Main Method()

```
main()
{
        While (simulation is running)
        {
                advanceOneTick
        }
}
```