*Team members:* Ahaz, Destin, Jerry, Timothy, Tung

# Traffic Light Simulator

# Format of document

1. Algorithm
2. Reflection
3. UML diagram
4. Use cases

# Algorithm

**Vehicle moving and spawning algorithm**

Begin

Until every road is examined

Until every lane is examined

Until every car in one lane is examined

Update global variable save from Save algorithm

Update global variable increment from Speedup's algorithm

Call Timer's algorithm with new increment

Update global variable shortest path array for the leading vehicle from Shortest path's algorithm

Update global variable Color from Traffic light color algorithm

Update global variable sensor status from Sensor algorithm

Update global variable PauseFlag from Pause's algorithm

If (save is True)

Keep running the simulation

Else if (PauseFlag is True)

Pause the simulation

Else

If (Color is Red)

Do nothing

Else

Update global variable ChangeLaneFlag from Changing lane's algorithm

Update global variable HideVehicleFlag from Hide vehicle's algorithm

Update global variable SpawnNewVehicle from Spawn new vehicle's algorithm

Update global variable NewLocation through move vehicle's algorithm

End if

Refresh the map according to the flags provided to see new changes.

End Until

End Until

End Until

**Initialization settings' algorithm**

Input: intersection object, total vehicle, traffic timing, sensor

Output: Success flag

Begin

Initialize how many intersections are there by dragging intersections on the map.

Initialize total vehicle.

Initialize traffic light timing for an intersection.

Initialize to turn on sensor option or not.

If either initialization is not successful

Return False

End if

Return True

End

**Save's algorithm**

Input: Button press

Output: Text file

Begin

If (user presses save button), then

Get total car, array of intersection's object, and sensor state

Export a text file to a chosen path

End if

End

**Shortest path's algorithm**

Input: Destination coordinate

Output: Array of shortest path

Begin

Until every path to the destination is explored using Depth First Search

Record the shortest path

End Until

Return the shortest path as list of x and y coordinate to the destination

End

**Changing lane's algorithm**

Input: None

Output: ChangeLaneFlag

Begin

If a vehicle requires lane change

Set change lane flag, either left or right.

Notify the directly opposite vehicle in the lane that you want to change if there are vehicles

End if

return ChangeLaneFlag.

End

**Traffic light color algorithm**

Input: Current colors of the traffic lights of an intersection (current traffic light state)

Output: Next colors of the traffic lights of an intersection (next traffic light state)

Begin

Place the eight traffic lights into a queue

Set the intersection to default (with green lights, allowing North to South and South to North)

IF (Sensor):

When the green light timer has passed:

IF there are no waiting cars in the lanes with red lights

RETURN current traffic light state

ELSE IF there are cars waiting

Set the green lights to yellow lights

When the yellow light timer has passed, turn the yellow lights into red lights

Pop the traffic lights that were green and put them at the end of the queue

FOR EACH traffic light in the queue:

IF there are no cars waiting at this light:

Place it at the end of the queue

ELSE IF there are cars waiting at this light:

IF this is the first light found:

Mark this light as the first in the next sequence

ELSE IF this is the second light found:

IF this light doesn't cause a collision with the first light:

Mark this light as the second in the next sequence

ELSE:

Place it at the end of the queue

IF two lights have been found for the next sequence:

Break out of the FOR loop

Reset the green light timer

ELSE IF (!Sensor):

FOR EACH of the traffic lights that are not red:

IF (Current color == Green && green light timer has passed):

Mark this light to be made yellow

IF (Current color == Yellow && yellow light timer has passed):

Mark this light to be made red

When the yellow light timer has passed: set the yellow lights to red lights

Pop the traffic lights that were green and put them at the end of the queue

FOR EACH traffic light in the queue:

IF this is the first light found:

Mark this light as the first in the next sequence

ELSE IF the first light has already been found

IF this light doesn't cause a collision with the first light:

Mark this light as the second in the next sequence

IF two lights have been found in the next sequence:

Break out of the FOR loop

Reset the green, yellow, and red light timers

{

Light combinations that don’t result in a collision (lights not mentioned are red):

North traffic light AND South traffic light,

West traffic light AND East traffic light,

North left turn traffic light AND North traffic light,

South left turn traffic light AND South traffic light,

West left turn traffic light AND West traffic light,

East left turn traffic light AND East traffic light,

North left turn traffic light AND South left turn traffic light,

West left turn traffic light AND East left turn traffic light

}

End

**Sensor algorithm**

Input: None

Output: Sensor turns on or not

Begin

When the user designates an intersection as having a sensor, turn the sensor on

When a new car is sensed, place in its intersection lane's queue

IF there are cars on the road(s) of this sensor:

RETURN true

ELSE IF there are no cars sensed on the road(s) of this sensor:

RETURN false

End

**Load pre-existing algorithm**

Input: configuration file

Output: None

Begin

if configurations is valid

open configuration as conf

initialization settings( conf.object, conf.totalcar, conf.traffictiming, conf.sensor)

else

error output << " wrong configuration file"

End

**Spawn new vehicle's algorithm**

Input: None

Output: Spawn new vehicle flag (Can be Boolean value)

Begin

    If((Timer increase) && (Max Vehicles not filled in lane))

        Set SpawnFlag

    End If

End

**Move vehicle's algorithm**

Input: coordinate of vehicle

Output: new coordinate of vehicle

Begin

    Until the vehicle has reached the destination(side of the map)

        If (vehicle has not met intersection)

            If (the front vehicle is a car)

                Increase the vehicle's location by 1 pixel forward if it is a truck

                Increase the vehicle's location by 2 pixels forward if it is a car

            End if

            If (the front vehicle is a truck)

                Increase the front vehicle's next location by 1 pixel forward

                End if

            If (there is no front vehicle)

                Increase the vehicle's location by 1 pixel forward if it is a truck

                Increase the vehicle's location by 2 pixels forward if it is a car

End if

vehicles do nothing to the coordinate

Endif

Elseif (vehicle met intersection), then

If (vehicle is on forward roads)

If (traffic light is red)  or ((traffic light is yellow) && (time passing the intersection < (time until the yellow light change - 2sec)))

Do nothing to the coordinate

Endif

If ((traffic light is yellow) && (time passing the intersection >= (time until the yellow light change - 2sec))) or (traffic light is green)

If (the front vehicle is a car)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

End if

If (the front vehicle is a truck)

Increase the front vehicle's next location by 1 pixel forward

End if

If (there is no front vehicle)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

End if

Endif

vehicles do nothing to the coordinate

Endif

If (vehicle is on turn left roads)

If (vehicle is able to turn left)

if (vehicle's current coordinate == the intersecting coordinate of current LeftTurnRoad and correct road of shortest_path within the intersection)

vehicle change to the correct road of shortest_path

Endif

If (the front vehicle is a car)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

End if

If (the front vehicle is a truck)

Increase the front vehicle's next location by 1 pixel forward

End if

If (there is no front vehicle)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

Endif

Elseif (cars are not able to turn left)

Do nothing to the coordinate

Endif

Endif

If (cars are on turn right roads)

If (cars are able to turn right)

if (vehicle's current coordinate == the intersecting coordinate of current RightTurnRoad and correct road of shortest_path within the intersection)

vehicle change to the correct road of shortest_path

Endif

If (the front vehicle is a car)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

End if

If (the front vehicle is a truck)

Increase the front vehicle's next location by 1 pixel forward

End if

If (there is no front vehicle)

Increase the vehicle's location by 1 pixel forward if it is a truck

Increase the vehicle's location by 2 pixels forward if it is a car

Endif

Elseif (cars are not able to turn right)

Do nothing to the coordinate

Endif

Endif

Endif

Return new coordinate of the vehicle

EndUntil

End

Note: Our group assumes car and truck have the same length and differ in the visual representation. Moreover, 1 pixel means increment in either x,y, or both coordinates depend on whether the vehicle is going straight or turning. 2 pixels means increment in either x, y, or both coordinates.

**Turn left algorithm**

Input: intersection object

Output: ability to turn left (bool)

Begin

if (vehicle meets intersection)&&(vehicle is on the turn left road)

timePassIntersection = time passing the intersection(length of intersection pixels/vehicle's speed) * 2

If ((left)traffic light is red) or (((left)traffic light is yellow) && (time passing the intersection < (time until the yellow light change - 5sec)))

Return false(stop)

Endif

Return true(go)

Endif

End

**Turn right algorithm**

Input: intersection object

Output: ability to turn right(bool)

Begin

if (vehicle meets intersection)&&(vehicle is on the turn right road)

timePassIntersection = time passing the intersection(length of intersection pixels/vehicle's speed) * 2

If (traffic light is red) or ((traffic light is yellow) && (time passing the intersection < (time until the yellow light change - 5sec)))

Return false(stop)

Endif

Return true(go)

endif

End

**Hide vehicle's algorithm**

Input: None

Output: Success flag

Begin

if vehicle_location_out_of_boundary

remove vehicle object from the map

End

**Speedup's algorithm**

Input: Current increment, multiplicity

Output: New increment

Begin

Multiply the current increment by the multiplier, and return that new increment

End

**Timer's algorithm**

Input: Increment

Output: None

Begin

Increment by default is set to 1

Each second, increase the total time of the simulation by the increment

End

**Pause's algorithm**

Input: Stop flag (Boolean value)

Output: Success flag

Begin:

if stop flag == true

stop car

stop traffic light

if sensor == on

stop sensor

cout << "user paused" << endl

return Success flag

End

# Reflection

**Reflection on multiple types of vehicles**

- **How our original design helped**
    - The previous design helps with the accommodating new type of vehicle which is a truck by leaving the generation of total vehicles per lane to the random number generator. Thus, we don't needed to know specific number for trucks or cars in the run session.
    - The previous design helps with accommodating new type of vehicle which is a truck since it allows users to specify the number of vehicles per run session.

- **How our original design didn't help**
    - The previous design does not help with accommodating new type of vehicle since we now need to modify our setTotalCar to setTotalVehicle.
    - The previous design does not help since we need to make truck and car the children's classes for vehicle class.

**Reflection on left turn arrow**

- **How our original design helped:**
    - Three lanes: In our original design, we had streets consist of 3 lanes. At intersections, the leftmost lane of this street would direct cars left, the center lane would direct cars straight, and the rightmost lane would direct cars right. We can still uphold this design with the addition of a left turn arrow. Adding a left turn arrow would mean that we would have to consider the possibility that traffic from a street could be directed to only go left, and not straight or right. In this situation, if there isn't a lane dedicated solely to left turns, a traffic jam could occur if frontmost cars want to go straight (preventing cars behind it from turning left). Since our original design has a lane dedicated solely to left turns, it supports this change.
    - Lights that correspond to lanes: In our original design, traffic lights would manage specific lanes. This meant that each intersection would have 4 traffic lights, which each light managing the 3 lanes coming from one of the 4 possible directions. In our new design, we split the management of the 3 lanes of a street into 2 different lights, the left turn light and the normal light (red, green, yellow). Thus, there is not much change in our data structure.
    - Sensors: The functionality of the traffic light sensors remains the same. In our original design, each traffic light object would have access to the information of a sensor object managing 3 lanes (a whole street). With the addition of the left arrow light, we can still uphold this design of having a sensor correspond to a traffic light; we just have to change how a street is sensed. The left turn traffic light would be able to sense cars on the leftmost lane it manages, and the normal traffic light would be able to sense cars in the center and rightmost lanes.

- **How our original design didn't help:**
    - Number of traffic lights: In our original design, we had each intersection utilize 4 traffic lights, one for each direction of traffic. This is because we could only signal traffic to go through a green light, meaning we only needed one traffic light for each street. With the addition of green left arrow lights, we decided to change our design so that each intersection would now have 2 traffic lights for each street. One light would have the green, yellow, and red lights, and the other light would have the green left arrow, yellow, and red lights.
    - New type of traffic light: In our original design, we only considered one type of traffic light: a traffic light with green, yellow, and red lights. In order to accommodate for the addition of a left green light, we had to modify our design. Traffic lights would now have to be split into 2 different types: a left turn light and a normal light. A normal traffic light would be the same as our original, and be used to manage the center and rightmost lane. A left turn light would have its green light replaced with a left green light, and be used to manage the leftmost lane.
    - More possible light combinations: In our original design, with the use of only green, yellow, and red lights, and using 1 traffic light per street, we had a very limited number of possible light combinations. Only one of the four traffic lights could be green at a single time, since cars would be able to go left, straight, and right, and we had to ensure that collisions wouldn't result. With the modification of adding a second left turn traffic light for each street, the number of possible flows of traffic at an intersection increased. For example, two opposite green left turn lights could be on, and a collision wouldn't occur. In our original design, we had 4 possible combinations (not including combinations where all lights are red), but by adding green left arrow lights, we now have 8 possible combinations of traffic lights.

**Reflection on road closure functionality**

Vehicles will have destination saved within themselves. They will follow the path toward the destination

- **How our original design helped**
    - In our original design, we have roads with multiple lanes. Each lane has its own functionality, either left-turn, right turn, go forward. Vehicles can change lanes in the middle of the stimulation to prevent the problem of traveling in circles. Since if a vehicle is not able to change lane, it will always on left-turn or right-turn lane when it was already on it. Since our original design has the functionality of changing lanes for vehicles, it helps vehicles to run on the path toward destination.
    - We already designed the functionality of lanes for left-turn or right-turn. This meant our original design has the functionality to direct vehicles. The arrow lights can help decrease the complexity of turning at the intersection. In our new design, traffic lights will work with lanes together to help vehicles to follow path.
    - We have a simulator timer for vehicles reaching their destination. This meant our original design can provide a visual support for students learning. In our new design, the

timer can demonstrate more information for student because the different paths may create more result which help students learn more.

- **How our original design didn't help**
    - In the new design, vehicles will have destination saved within themselves. This meant we need to find the path toward the destination instead of reaching the side of the map. In our original design, vehicle's destination is not determined, so the vehicles will finish their travel once they reach the side of the map. Although the turn-left/right functionality is exist in our original design, vehicles are still moving toward the sides randomly. Therefore, the path toward the destination is necessary in our new design, as well as changing lane and turn left/right functionalities.
    - In our original design, we have designed turning availability of both lanes and traffic lights. However, those are two separate functionality. This meant in our original design, vehicles' moving functionality is directed by a single type traffic light or lane functionality. Therefore, in our new design, we combine them into one functionality. Vehicles now need to follow the traffic light and be in the correspond lane to move.
    - Since this change affects all vehicles not only the car, we need to consider how the effect propagate through the design of moving traffic. Trucks moves slower than the cars, so the numbers of different types of cars affect the final report of the simulation. The trucks will decrease the number of vehicles which can pass through the intersection per round of traffic lights. This may cause the change of shortest path based on the density of roads. Therefore, our new design is more realistic and help students more.

**Reflection on Co-simulation functionality**

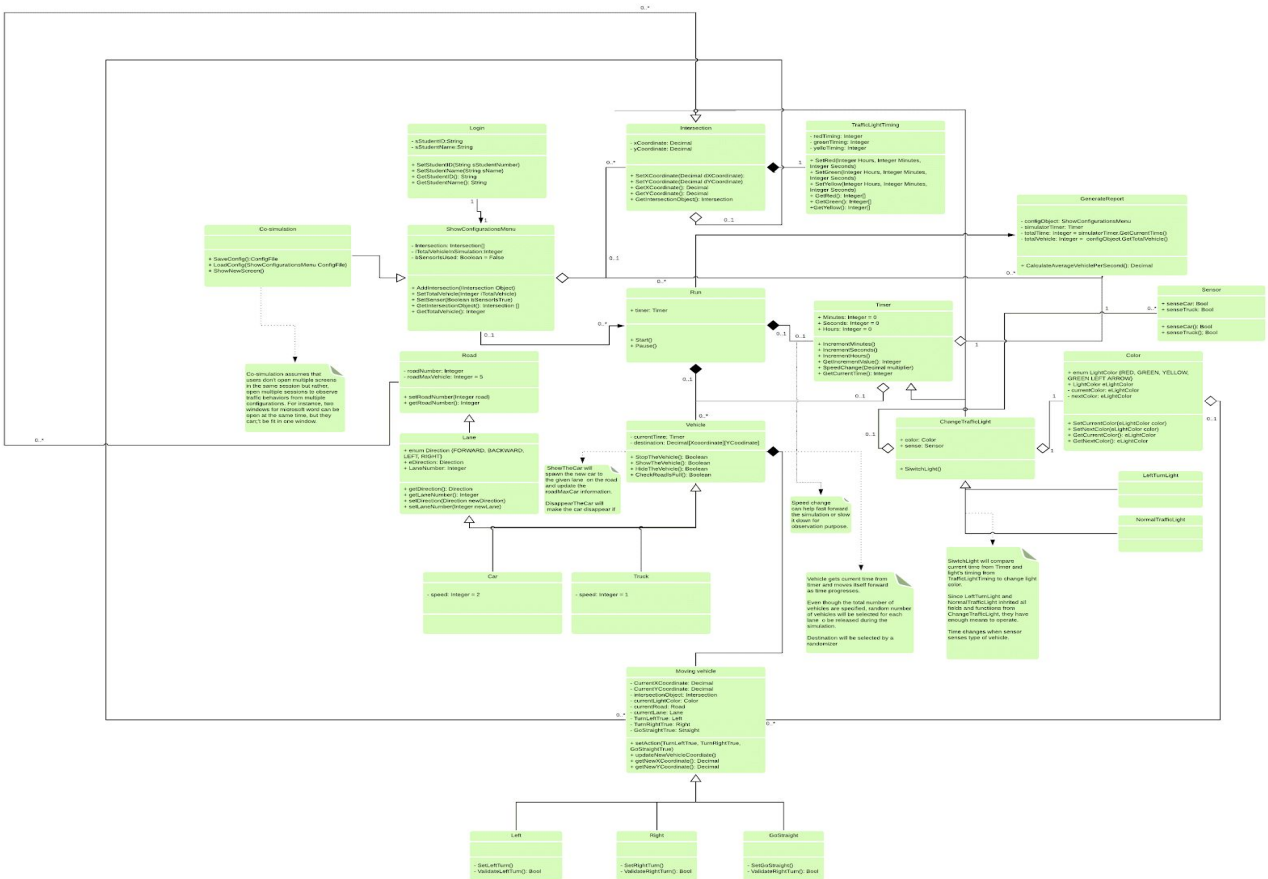
- **How our original design help**
    - Original design helped to design new changes because it gave basement of new changes such as the structure of the original design, such as settings, login, run, etc.. are connected to Co-simulation. Without the basement, it could have been a totally different design of what it is right now in terms of elements, structures. Also could have been confusing design for us, the developers, and clients, student and professors.
    - Original design contained variable settings such as Intersections, total car, and sensor used, which made it easier to implement the save configuration function as well as load configuration function where elements can be simply retrieved from the original setting and saved into the file or retrieve from a file and load it onto the system.
    - Premade program itself, original design, helped in designing co-simulation because basically, opening a new window only requires to make a new tab as a window and run the same exact program on them. Which makes it easier to make multiple and run at the same time because it does not require to build a whole program again, nor make changes.

- **How our original design didn't help**
    - Had to decide what kind of configuration file would be. As in our original design, there was no preset for configuration file. Having save and load required a new method to create the configuration file and follow it's pattern and algorithm to figure out each element's value and save or load into a corresponding setting element in the system.
    - Basic new screen implementation needed an isolated design for itself. Whenever people try to open new screen or simulator, It needed to be able to run duplicate program as the original was only set to 1 simulator at a time.
    - Loading and saving needed accurate value of each setting. But our original design did not have accessor for the sensor, a boolean, and needed to directly access it by calling the element from the setting. It could have been better and safer to have an access method because by mistake the original value has a chance of being changed by the user since it is not a constant variable.

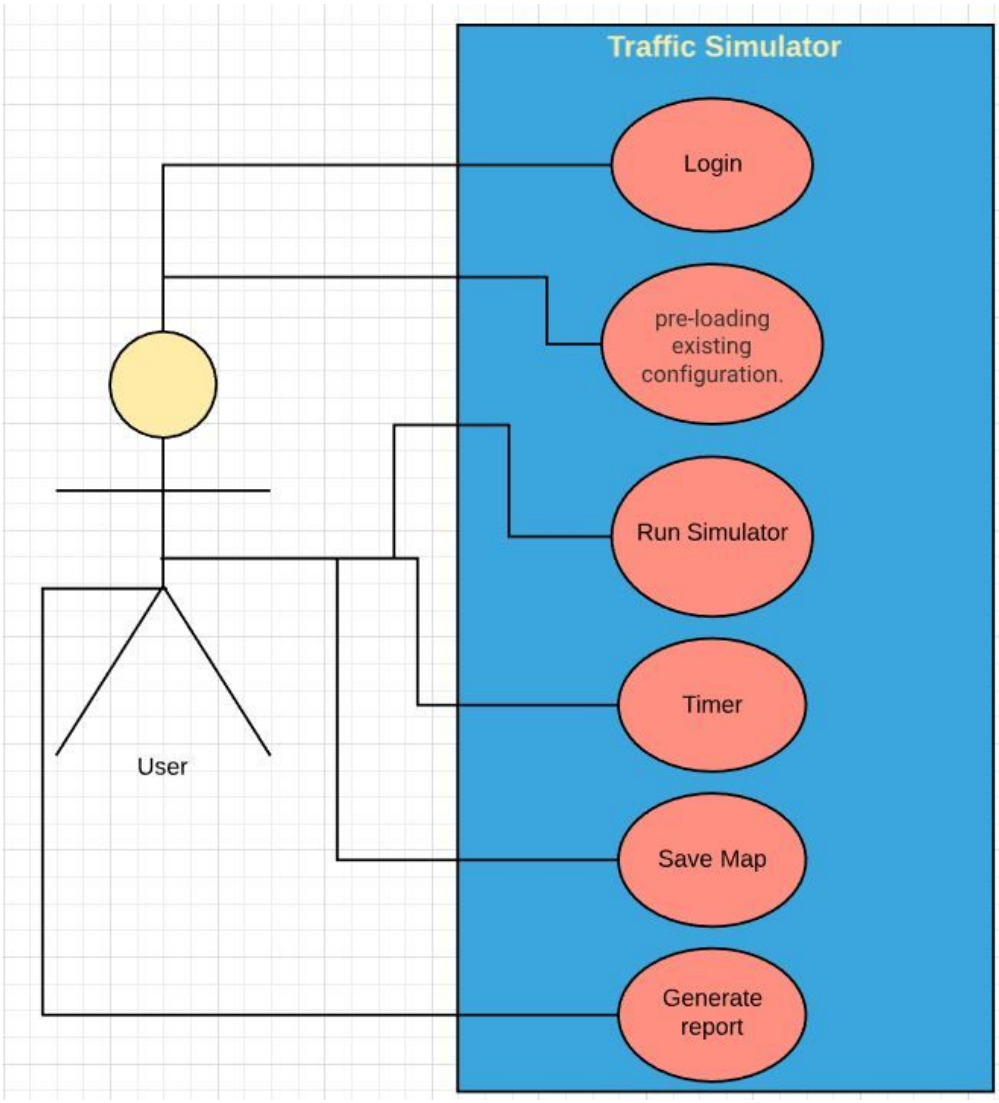
# UML Diagram

**Please use the link to access the document for better visual representation of the UML.**

https://www.lucidchart.com/documents/view/f082984f-c773-4f16-b5d9-d5bb6fedd082/0



## Use cases


Traffic Simulator
Login
pre-loading existing configuration.
Run Simulator
Timer
Save Map
Generate report
User