Overview:

This part is about the implementation of the LPS visualisation. This section consist of 5 part. The second part is about the structure of the project including the link between the front end and back-end side. The third is about the LPS program implement to fit various traffic situation. The Fourth part is about the front side LPS program parsing, data strucre processing and animation. The fifth part will be the server side programming including using technology Express and some user authentication tools. Since the main purpose of this project is to visualise the LPS program in traffic scenario vively. There are severly issue that need to be addressed before designing and implementing the program.

1. How will the LPS program be prased into javascript since there is no interface of write LPS code direct in JavaScript format.

To adding API into LPS interpreter is very time consuming because it needs to understand and modify the LPS.js directly and the interpreter (lps.js) does not encapsulate the key function in (Engine.js) also address by Sam Yong “lps.js does not support the construction of LPS programs by JS without explicitly writing LPS code in order to reduce the amount of code base and API implementation needed.” The idea of formatting LPS directly in javascript was abanded very early. To solve the data parse issue we prase data direct from the LPS code (file with extension .lps) by file proceing in a ascynchornised way.

1. What front-end tool is going to be used to achieve live animation.

There is a animation tool which has been implemented in Node.js Electron called LPS studio however there are variety miserable bugs such as (placing print before action predication causing crash of the program, unable to work with recursion etc). The idea of using LPS studio is abanded. After week of research, PIXI.js was found very suitable to animate the LPS program. This lead to another problem wehter to use pixi direct in HTML or use it in node.js by create another canvas. By some experiement, using it directly in html was more light weighted which was PIXI.js the front-end side animiation side original design to.

1. What information of LPS program need to be prased into the program.

These three question were all tucked down during a long time research.

There is one open interface that is available which is the

const LPS = require('lps');

LPS.loadString('...')

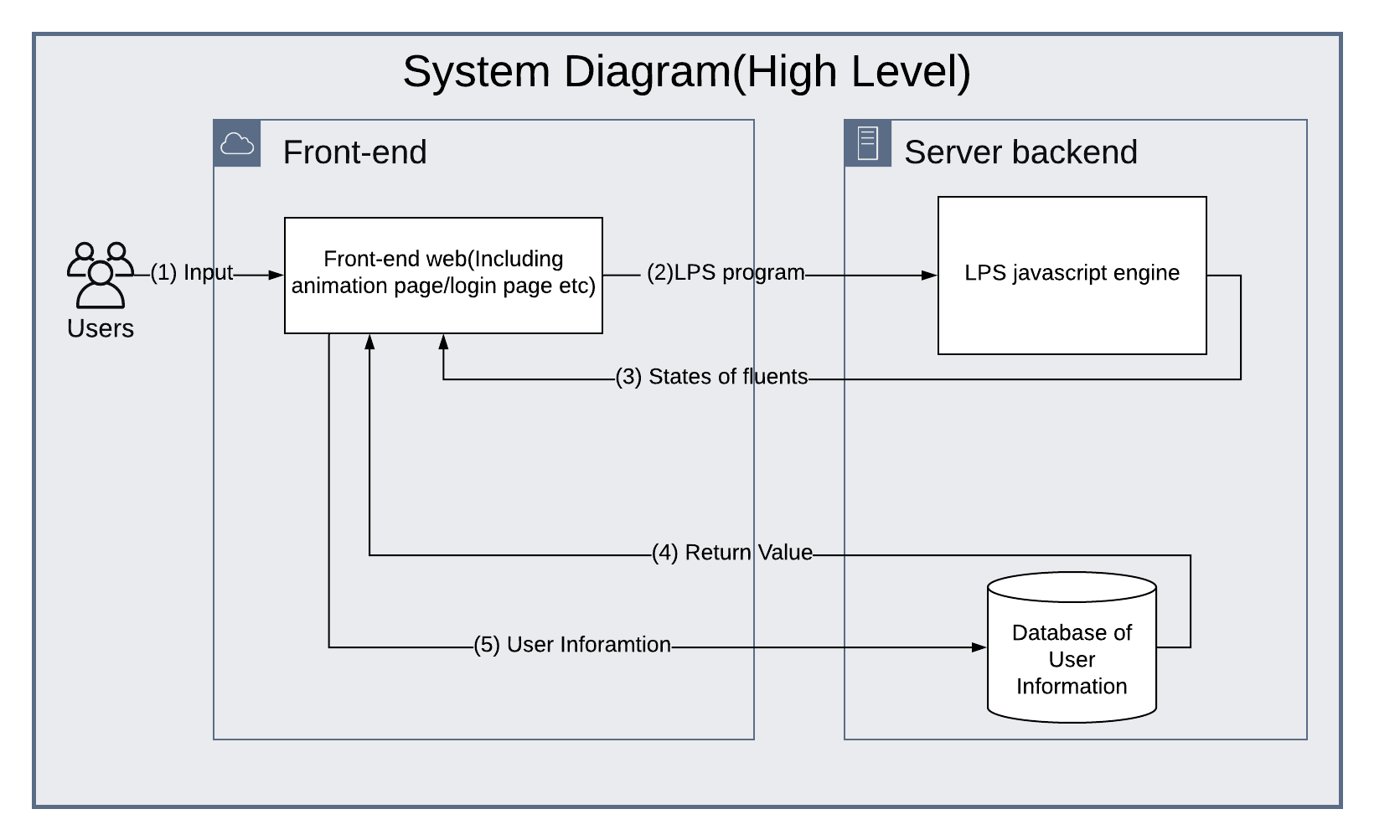
.then((engine) => {

engine.run();

});

The data that we are parse onto the web is then solved. The only issue we need to solve is to formatting different fluents into different data structure and capture the change of fluent in each running cycle, detail will be illustrated in the following subsection.

Structure Design:

the high-level system diagram of LPS visualiser is shown as follow: 

1. Means any input to the system such as typed in user information, loaded LPS code, mouse dragging event, typing LPS program in text area and so on. After the input event happened the corresponding page need to handle these responses or input accordingly. that the LPS program should be in JavaScript syntax.
2. The main job of the LPS visualiser is to pass the LPS program into interpreter (The LPS.js) running as a JavaScript library. After that the changing state of fluent(as No.3 shown) will be passing onto the webpage while the LPS program is running benefited to the JavaScript concurrency model.

(4) The interaction of the back-end program and the also include the login and register part. User need to either register as a new user or login. After this part user information in the database will be retrieved (as No.5 shown), and REST API will comes to play. REST API will return to the main visualization page if user information is valid or stay in the login page if user information is not valid.

LPS program

Overview：

The LPS program is the input program. For the traffic the LPS program should not only follow the structure of the interpreter understandable way as we discussed in section (?) also it should be easy to be parsed into the PIXI animation engine. Hence the format is defined as below (User can replace any Upper-case text to real instance number or predicate:

% we assume the destination is reachable

maxTime(Number).

cycleInterval(Number).

fluents([

Set of fluent predicate defined

]).

events ([

Set of event predicate

]).

Observe([

Set of observation predicate (happed at a particular time)

]).

actions ([

Set of actions predicate defined

]).

initially([

Initial set of fluent predicate (true from cycle 1)

]).

Fact predicate (true forever)

Reactive Rules …

……..

Predicate Rules

Action (change fluent)

Define street(fact) as: street(StreetName, coordinate(X, Y), Width, Height, Number\_of\_lane) ie street(mainStreet , coordinate(100, 200), 900, 50, 1). Meaning there is a street called mainStreet with the top left corner (100,200). Width 900 height 50 and the number of lane is 1.

Define location of vehicle(fluent) as: location(Name\_of\_car, coordinate(X, Y), Direction) ie: location(car0, coordinate(150, 225), eastward) meaning car0 is located at (150,225) and facing eastward.

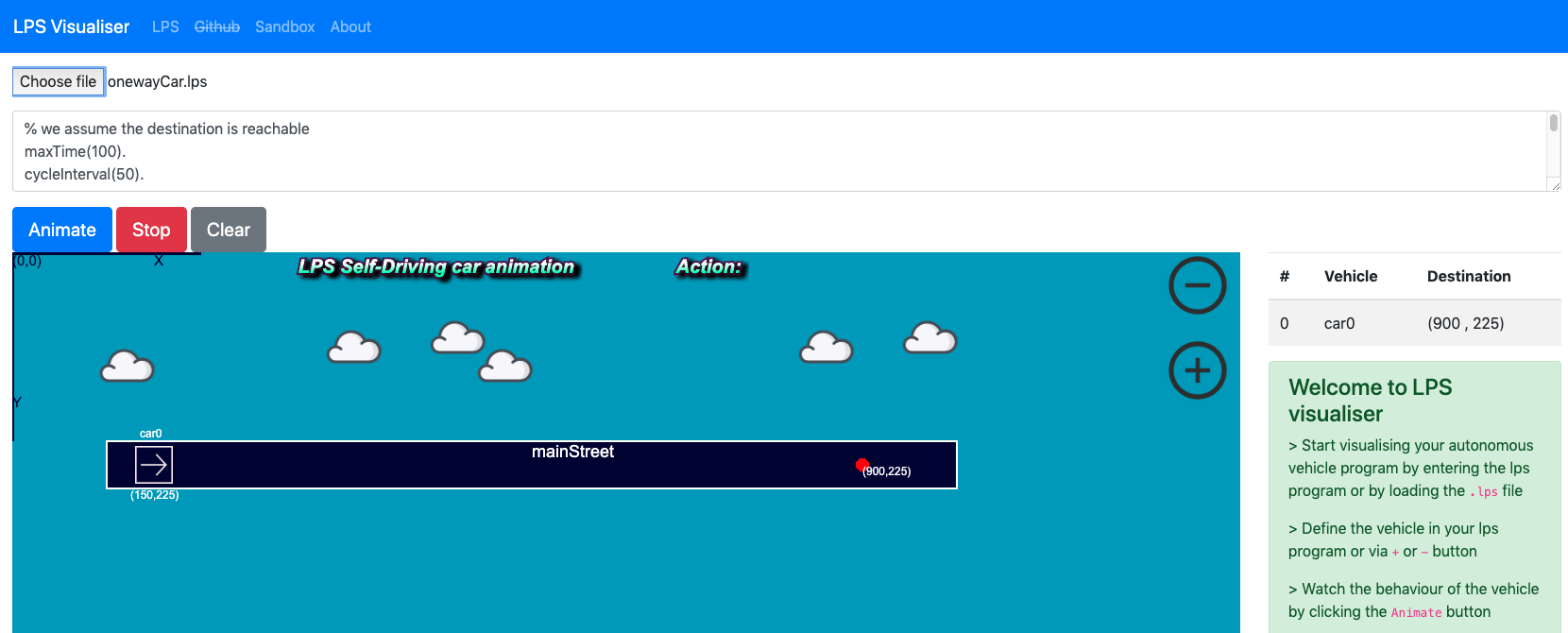
Define traffic light(fluent) as: trafficLight(coordinate(X, Y), Working\_status, Color, FacingDirection) ie: trafficLight(coordinate(430, 475), red, westward, mainStreet) meaning there is a traffic light located at coordinate(430, 475) and it is showing red color at the moment, the traffic is facing

Define the goal of the Vehicle(fluent) as: goal(VehicleName, coordinate(X, Y)) ie: goal(car0,coordinate(980, 475)), meaning the goal of car0 is at (980 475)

Each vehicle can have multiple state(fluent) eg stopped(VehicleName) or moving(VehicleName) ie: moving(car0) meaning car0 is at state of moving.

Define the junction(fact) as: eg junction(JunctionName,coordinate(A, B),coordinate(C, D),coordinate(E, F),coordinate(G, H)). where (A, B), (C, D), (E, F), (G, H) are 4 different corners. Ie junction(tJuntion1,coordinate(450, 450),coordinate(500, 450),coordinate(450, 500),coordinate(500, 500)). There is a junction called tJuntion1 the four corner are (450, 450), (500, 450), (450, 500), (500, 500).

Car moving straight



the lps program example is

% we assume the destination is reachable

maxTime(100).

cycleInterval(50).

loadModule('../scripts/module.js').

fluents([

stopped(VehicleName),

moving(VehicleName),

coordinate(X, Y),

location(VehicleName, coordinate(X, Y), Direction),

trafficLight(coordinate(X, Y), Working\_status, Color, FacingDirection),

street(StreetName, coordinate(X, Y), Width, Height, Number\_of\_lane),

goal(VehicleName,coordinate(X, Y))

]).

actions ([

step(Vehicle, NextPlace),

turn(Vehicle, NewHeading),

arrive(Vehicle)

]).

initially([

moving(car0),

location(car0, coordinate(150, 225), eastward),

goal(car0,coordinate(900, 225))

]).

street(mainStreet , coordinate(100, 200), 900, 50, 1).

cloud(coordinate(140,100)).

cloud(coordinate(250,90)).

cloud(coordinate(300,120)).

cloud(coordinate(640,100)).

cloud(coordinate(750,90)).

cloud(coordinate(800,120)).

goal(Vehicle,coordinate(A, B)) from \_ to T,

location(Vehicle, coordinate(X, Y), Direction), A==X, B==Y,moving(Vehicle) at T ->

% testPrint(Vehicle+ ' we have arrived'),

arrive(Vehicle) at T.

goal(Vehicle,coordinate(A, B)) from \_ to T,

location(Vehicle, coordinate(X, Y), Direction), A!=X at T ->

% need to find the right direction here

% driving forward

drive(Vehicle) from T to \_.

goal(Vehicle,coordinate(A, B)) from \_ to T,

location(Vehicle, coordinate(X, Y), Direction), B!=Y at T ->

% need to find the right direction here

% driving forward

drive(Vehicle) from T to \_.

drive(Vehicle) from T to T1 <-

location(Vehicle, coordinate(X, Y), Direction),

Direction == northward,

NewY = Y - 10,

NextPlace = coordinate(X, NewY),

step(Vehicle, NextPlace) from T1 to T2.

drive(Vehicle) from T to T1 <-

location(Vehicle, coordinate(X, Y), Direction),

Direction == southward,

NewY = Y + 10,

NextPlace = coordinate(X, NewY),

step(Vehicle, NextPlace) from T1 to T2.

drive(Vehicle) from T to T1 <-

location(Vehicle, coordinate(X, Y), Direction),

Direction == westward,

NewX = X-10,

NextPlace = coordinate(NewX, Y),

step(Vehicle, NextPlace) from T1 to T2.

drive(Vehicle) from T to T1 <-

location(Vehicle, coordinate(X, Y), Direction),

Direction == eastward,

NewX = X + 10,

NextPlace = coordinate(NewX, Y),

step(Vehicle, NextPlace) from T1 to T2.

% on(coordinate(X,Y),Street) <-

updates(step(Vehicle, NextPlace), location(Vehicle, OldPlace, Direction), location(Vehicle, NextPlace, Direction)).

updates(turn(Vehicle, NewHeading) , location(Vehicle, Place, OldHeading), location(Vehicle, Place, NewHeading)).

terminates(arrive(Vehicle), moving(Vehicle)).

initiates(arrive(Vehicle), stopped(Vehicle)).

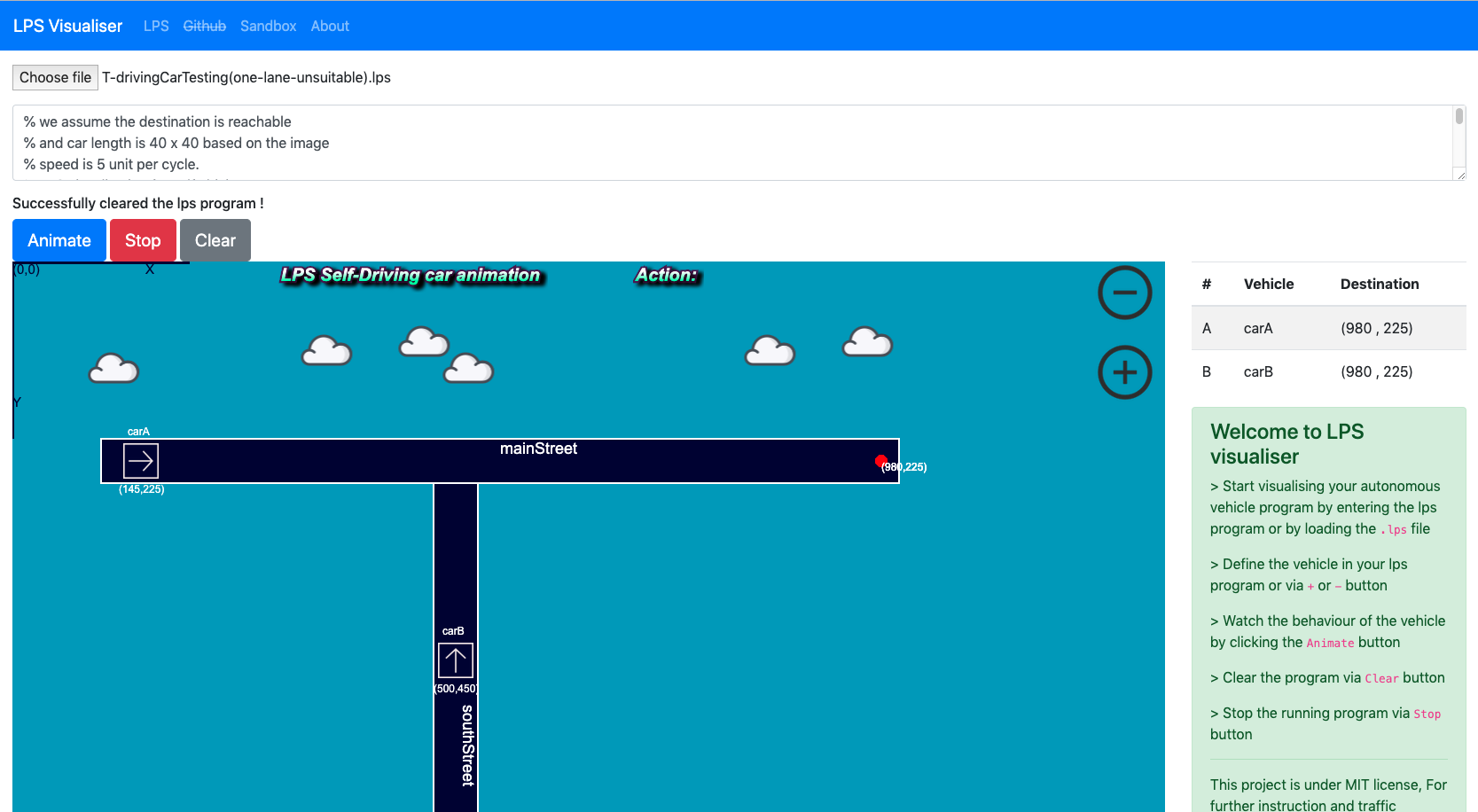
% you can not move (x++) if you are not facing eastward

The reactive rule checks if the goal has been reached if not then keep driving towards the current direction.

When step forward step action is performed which will terminate the old location and initialize a new location.

Once the goal was reached, the moving state will be terminated.

T junction with both narrow lane.



In the graph the red dot represent the goal location of the car. In this particular example both carA and carB has the same goal of (980 225). However when both car reached the T junction carB should give way to the carA because the car on the main road has higher priority in this case.

goal(Vehicle,coordinate(A, B)) from \_ to T,

location(Vehicle, coordinate(X, Y), Direction),

moving(Vehicle) at T ->

% need to find the right direction here

direction(Vehicle) at T.

This reactive rule was added in this scenario because the vehicle need to check the direction

T junction with one narrow lane and a double lane

Two car:



This is similar to the previous T junction with both narrow lane however a new road and junction need to be defined in this case.

Three cars

The three car in a T junction scenario is more completed. When cars meet at the T junction it need to make decision depends on the intension of the other car. However some rules are applied here:  Assumption: carA intend to turn right or go straight. carB intend to turn left or right, carC intend to turn left or go straight

  Based on the assumption, few rules are made:

 ->A car driving straight has a higher priority than a car turning right or left at an un-controlled T-junction.

->If a car is driving into a narrow road it should wait until other car drive out of the narrow.

 ->One car is turning left and another car is turnning right to the same road at the same time, left turn has higher priority

% % There are 8 cases in total that your route will be blocked

% if Vehicle is at same horizontal level as its goal and there is another car in between

blockedRoute(Vehicle,Vehicle2) at T<-

location(Vehicle, coordinate(X1, Y1), Direction),

Direction == eastward,

goal(Vehicle,coordinate(A1, B1)),

location(Vehicle2, coordinate(X2, Y2), \_),

Vehicle != Vehicle2,

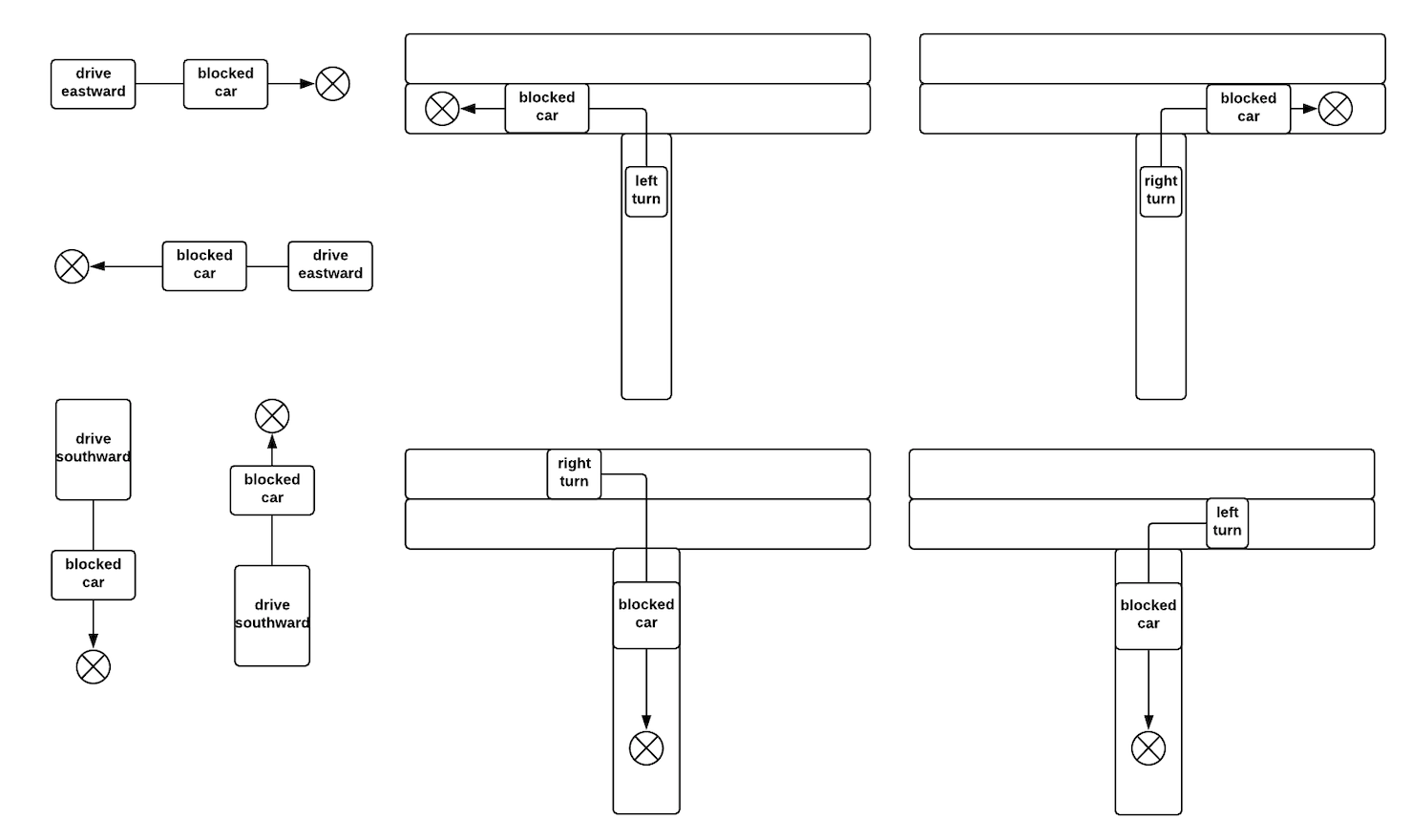
Y1 == B1,

Y2 == B1,

X2 >= X1,

X2 <= A1.

as described above, one example ofvehicle2 blocked the route of vehicle code of is written as above This example describe about Vehicle is at same horizontal level as its goal and there is another car in between. However, there are 8 cases in the blocking road. There are 4 cases in car driving straight toward (toward north south east west) to the destination and there is a blocking car. there are 4 more cases about car in a L shape road these cases are car on the main road and turning left car on the main road is turning right. Car on the narrow road and turning left, car on the narrow road and turning right. As I lustrated in the graph:



intention(VehicleName, Plan) <-

location(VehicleName, coordinate(X, Y), Direction),

goal(VehicleName,coordinate(A, B)),

Direction == northward,

Y > B,

X > A,

Plan = turnLeft.

Intension of the vehicle will also be added since vehicle meet at this particular junction need to make decision depends on the intension of other vehicles. The predicate of intension is also defined as follow:

% both car have clear routes but car not on main street can not step

<- step(Vehicle, NextPlace),

not onMainRoad(Vehicle) at T,

collisionPossible(Vehicle, Vehicle2) at T,

Vehicle != Vehicle2,

clearRoute(Vehicle) at T,

clearRoute(Vehicle2) at T.

Also the restriction is added in this case because the vehicle need to check wether the route is clear is not is cannot step ahead

% both car have clear routes but car on the narrow road can not step

% (car on main street has higher priority)

<- step(Vehicle, NextPlace),

not onMainRoad(Vehicle) at T,

collisionPossible(Vehicle, Vehicle2) at T,

Vehicle != Vehicle2,

onMainRoad(Vehicle2) at T,

not blockedRoute(Vehicle,Vehicle2) at T,

not blockedRoute(Vehicle2,Vehicle) at T.

The restriction of step action is will also be applied one of the example of both car have clear routes but car on the narrow road can not step is shown as above.

T junction with two double lanes

In this case the traffic light is introduced here which means car is not going making decision when there is more than one car at the junction such that the decision making code as we described in section above will be removed. The reactive rule for traffic light will be added.

trafficLight(coordinate(X, Y), Color, FacingDirection,Street) at T->

opposite(Color,Color2),

Reminder = mod(T,40),

Reminder == 0,

changeTrafficLight(coordinate(X, Y),Color2) at T.

the above code means every 40 time cycle the color of the traffic light will be opposite (from red to green or from green to red).

% updating the traffic light

updates(changeTrafficLight(Place,Color) , trafficLight(PlaceIns, OldColor, \_, \_), trafficLight(PlaceIns, Color, \_, \_))<-

% testPrint('traffic light as been updates'),

Place == PlaceIns.

The action of traffic light need to be added to change the state of the traffic light

<- step(Vehicle, NextPlace),

location(Vehicle, coordinate(X, Y), Direction1),

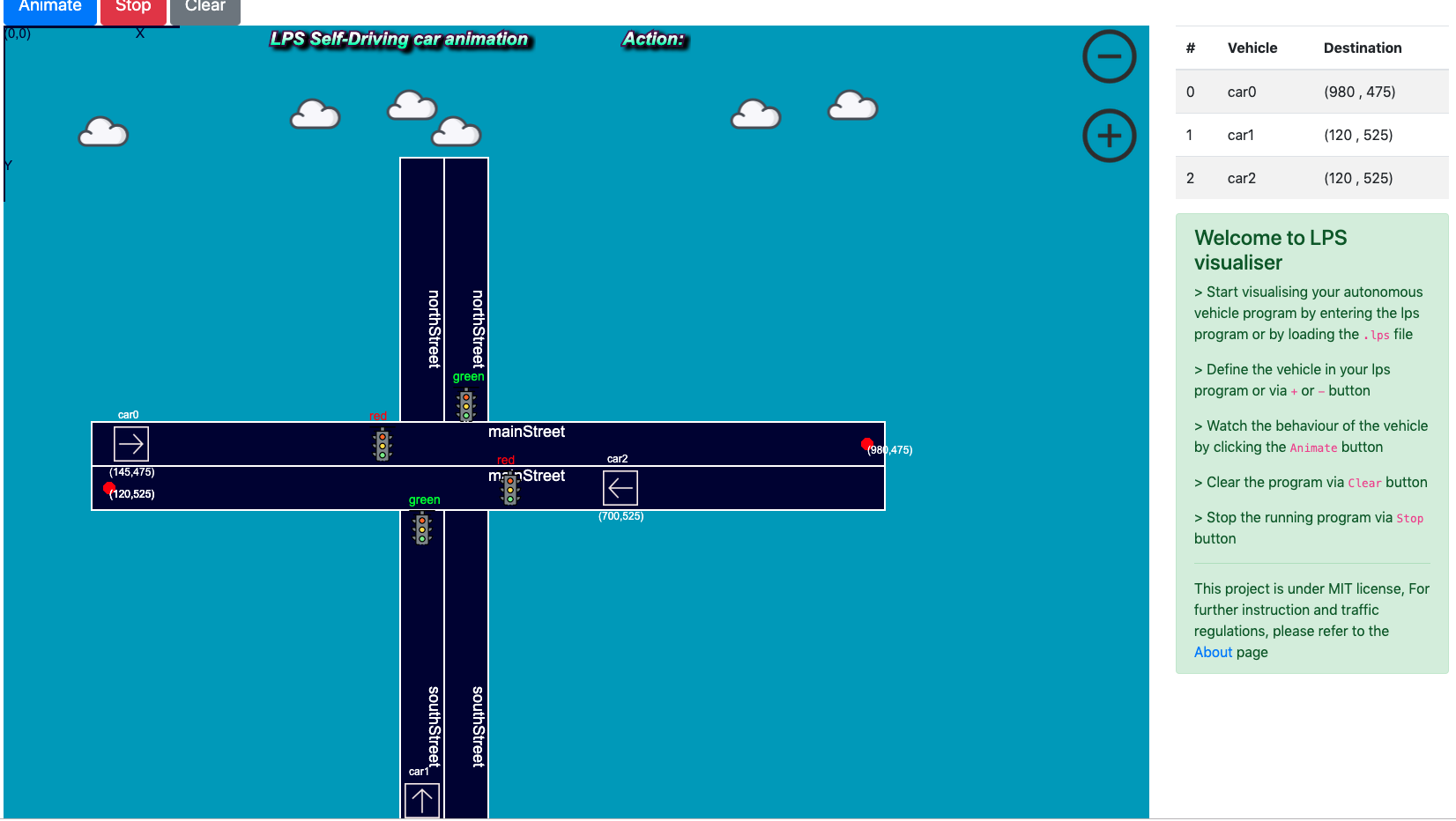
trafficLight(coordinate(A, B), red, Direction2, Street),

X==A,

Y==B.

Restriction of step action need to be added since vehicle can not go forward when the traffic light is red

Cross junction with traffic light



In the case of the cross junction there are still 4 junction block in total so the reactive rule and the restriction of the step action is the same as the previous two double lane T junction road. No more reactive rule is introduced here.

Traffic with restart feature

To visualise traffic flow better, a restart feature is design to putting the vehicle back to orginal porint instead of stop at the arriving location. As we do not have information of the intial fluent so the start location need to be stored as a facts.

startLocation(car0, coordinate(145, 475), eastward),

startLocation(car1, coordinate(475, 880), northward),

startLocation(car2, coordinate(700, 525), westward),

For the traffic light with restart feature

% if the car is stopped and it has a start location.

stopped(Vehicle) at T ,startLocation(Vehicle, \_, \_)->

restart(Vehicle) at T.

terminates(restart(Vehicle), stopped(Vehicle)).

initiates(restart(Vehicle), moving(Vehicle)).

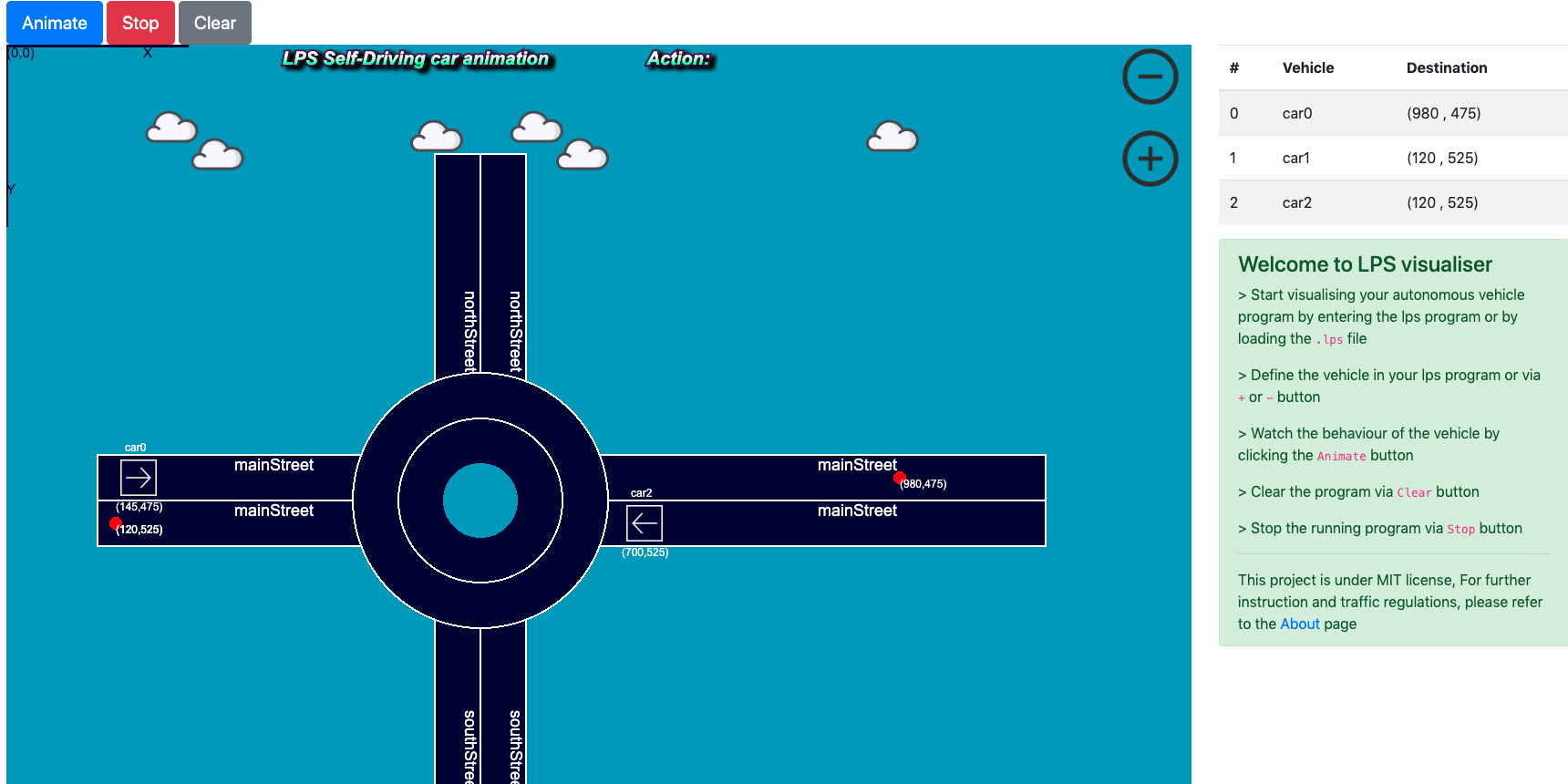
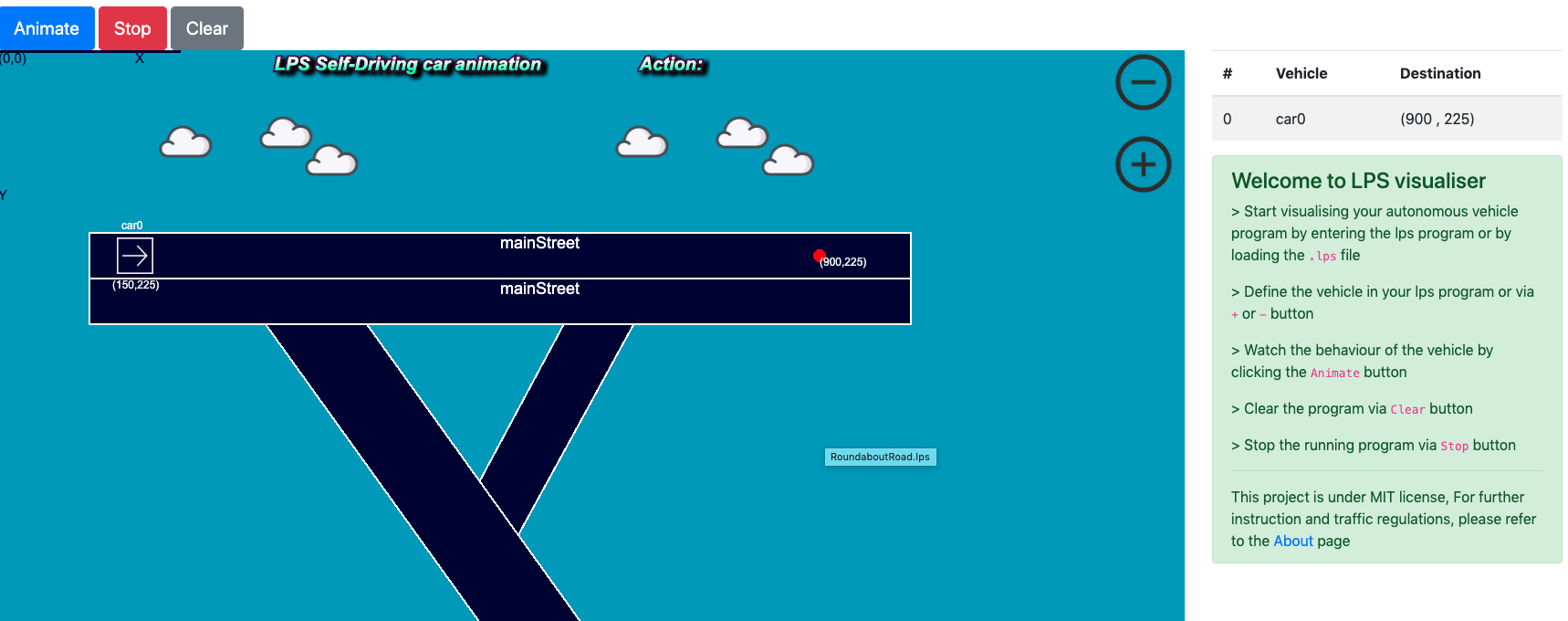
updates(restart(Vehicle) , location(VehicleIns, OldPlace, OldDirection), location(VehicleIns, NextPlace, NewDirection))<-

VehicleIns == Vehicle,

startLocation(Vehicle,NextPlace,NewDirection).

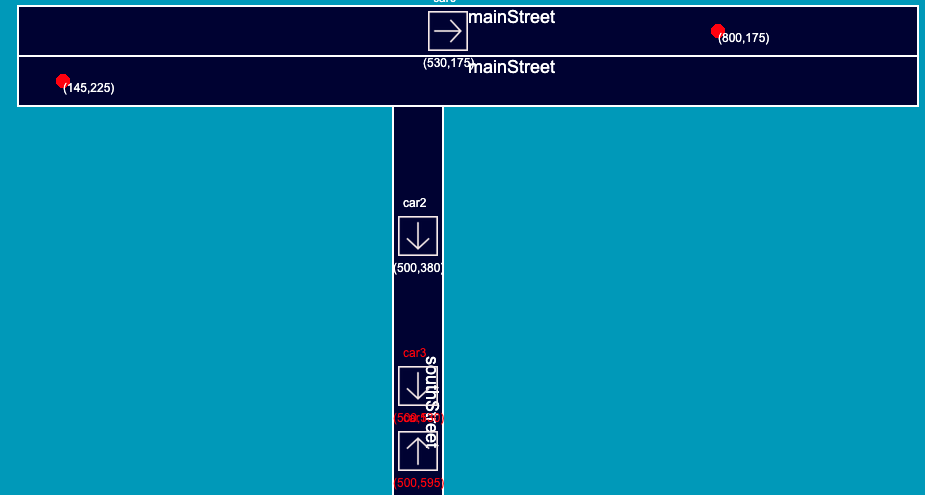
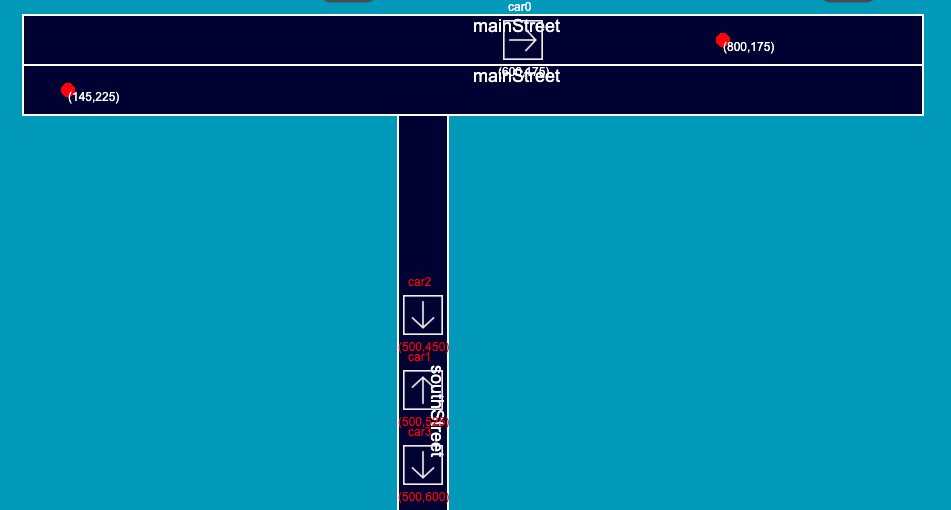
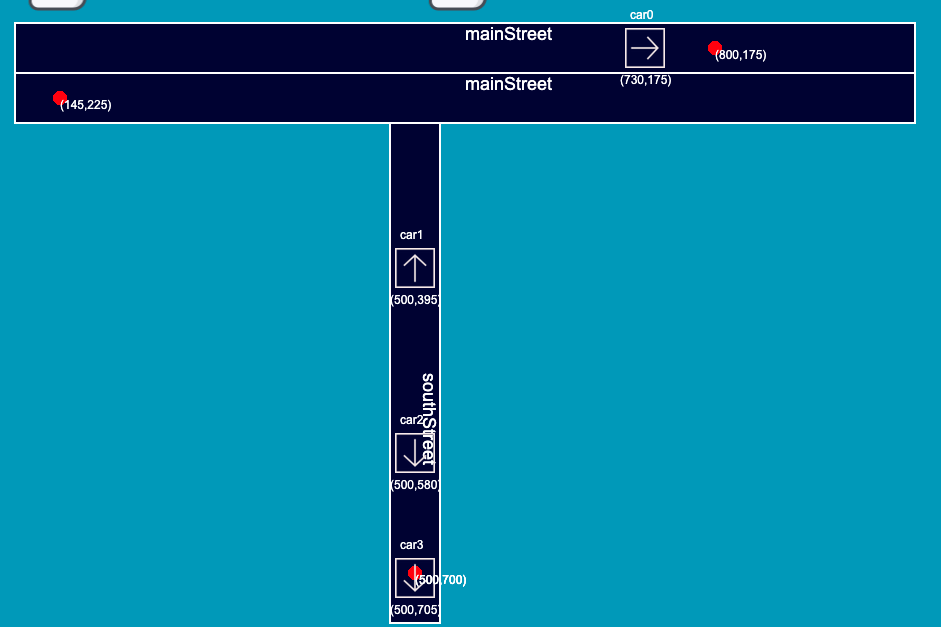
The new reactive rule is define as above saying whenever the car in stopped fluent it will restart fron the starting point. Hence a new action is also need to be written as follow:

Other Traffic road interface:



The interface of define the roundabout and the Y shape junction is also defined in the project. However due to the limitation of the direction it only restricts to north south west and east. Vehicle is not able to be similutate in Y shape road and roundabout.

Cars passing through each other on a narrow road

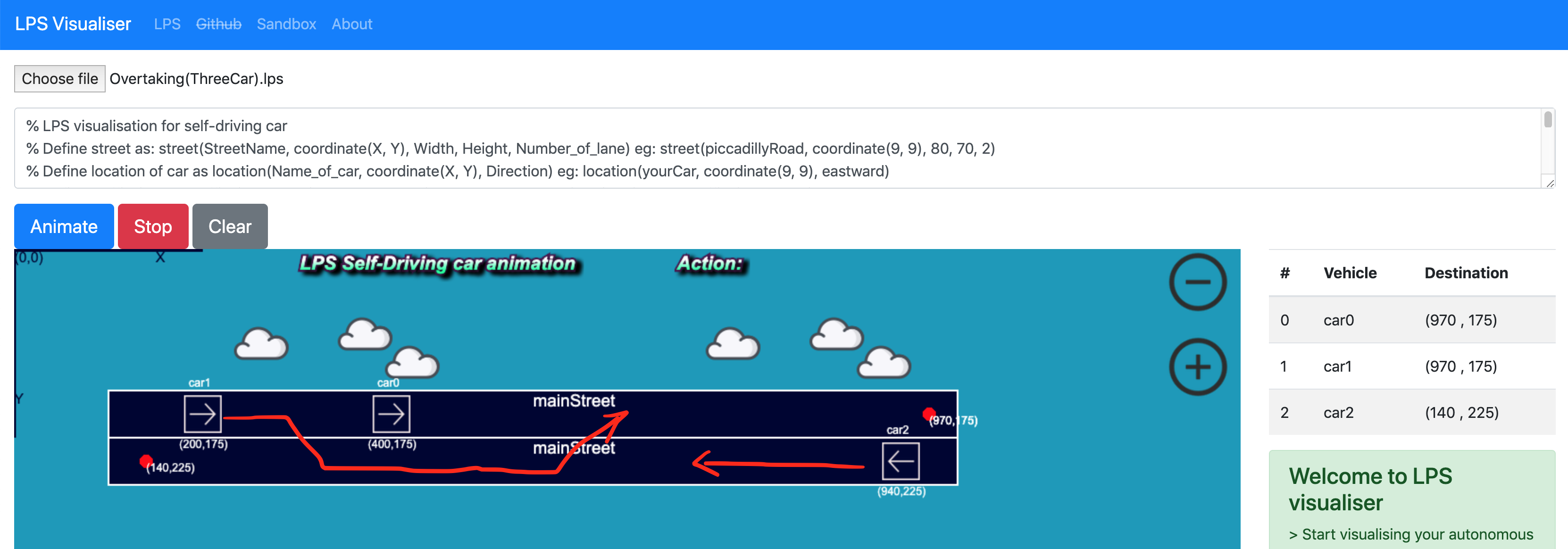
  

while two cars are driving towards each other (within a safety distance) we can assume two car can squeeze by each other, indicate by turning the text and location into red. After this stage the colour of the vehicle will turn back to it origin colour.

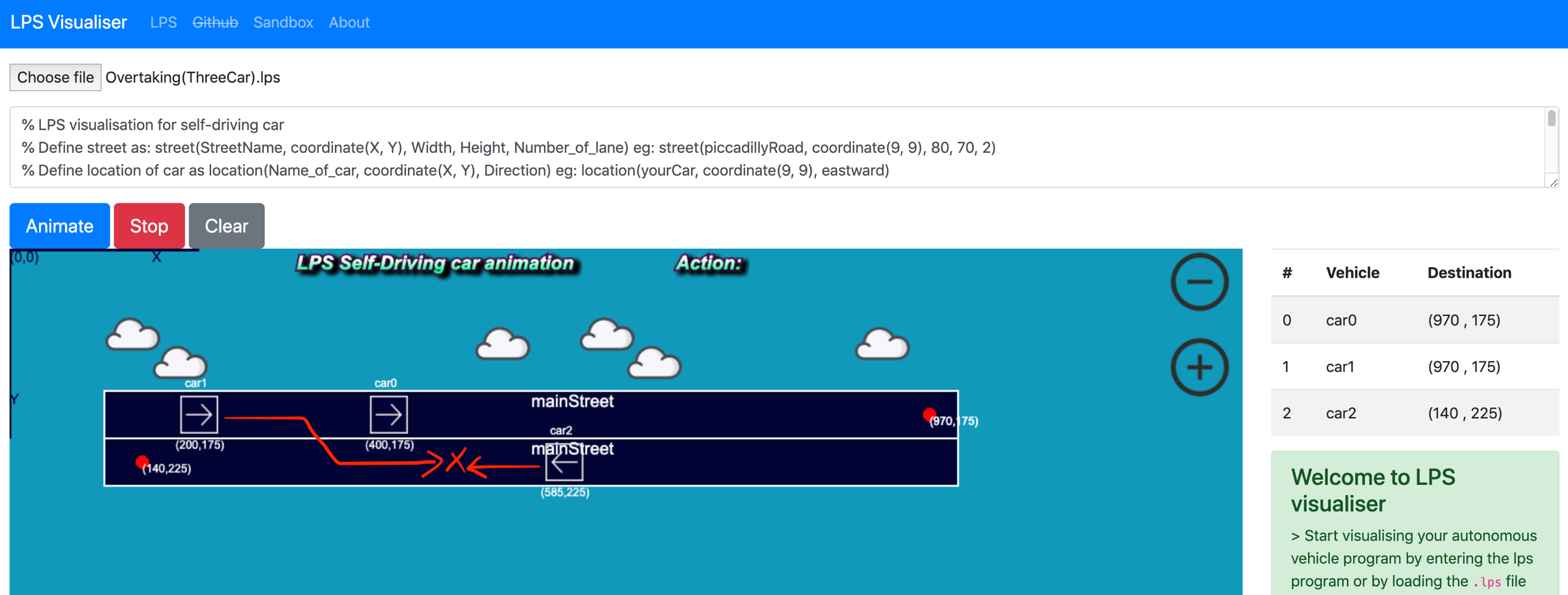
Overtaking

The car need to perform overtaking so different speed need to be assign to each car as a fluent.

velocity(car1,5). meaning car1 has speed of 5 pixcel per running cycle.



as describe in the background (section?) the car need to check if there another within its overtaking route if the overtaking route is clear the overtaking can perfume overtaking. If not car will follow in the behind of the car. As the example above, car1 will perform the overtaking because car2 will not collide with car1 during the overtaking action.



In the example above car1 will not perform the overtaking

collisionPossible(Vehicle, Vehicle2) from T1 to T3,

Vehicle != Vehicle2,

velocity(Vehicle, S1),

velocity(Vehicle2, S2),

S1 > S2,

blockedRoute(Vehicle,Vehicle2) at T1 ->

overtaking(Vehicle,Vehicle2,T1,T3).

overtaking(Vehicle,Vehicle2,T1,T3) <-

location(Vehicle, coordinate(X2, Y2), \_) at T1,

NewY = Y2+50,

NewLocation = coordinate(X2, NewY),

aheadClear(Vehicle,Vehicle2,Vehicle3),

not goalTooNear(Vehicle),

step(Vehicle, NewLocation,T2, T3).

navigateBack(Vehicle,T1,T3)<-

location(Vehicle, coordinate(X2, Y2), \_) at T1,

NewY = Y2-50,

NewLocation = coordinate(X2, NewY),

step(Vehicle, NewLocation,T3, \_).

goalTooNear(Vehicle)<-

goal(Vehicle,coordinate(A1, B1)),

location(Vehicle, coordinate(X1, Y1), \_),

Diff = abs(A1-X1),

Diff < 300.

not collisionPossible(Vehicle, Vehicle2) from T1 to T3,

velocity(Vehicle, S1),

velocity(Vehicle2, S2),

location(Vehicle, coordinate(X1, Y1), Dir1),

location(Vehicle2, coordinate(X2, Y2), Dir2),

Dir1 ==Dir2,

X1 > X2,

Y1 > Y2,

S1 > S2 ->

navigateBack(Vehicle,T1,T3).

LPS code of overtaking from upper lane to the lower lane is shown as above. If the front car has a low speed and is blocking the way them perform overtaking event. If the ahead route is also clear and the goal is far away then overtaking will be successful performed otherwise fail. 50 is the street width and 300 is the clearance distance setting as for convience. However in reaility the clearance should decide depends on the speed of the car.