# ECEN689-602/CSCE689-603 Introduction to Formal Verification Fall 2021

Project Report Phase B

Submitted to the Faculty

Of

Texas A&M University

By

**Team 6 I-Group Integration**

Brent Arnold Basiano UIN: 130004869

Sri Hari Pada Kodi UIN: 932003040

Under the Guidance of

**Professor Jiang Hu**

**Department of Electrical and Computer Engineering**


November 2021

**TABLE OF CONTENTS**



**LIST OF FIGURES**

# ECEN 689 - Introduction to Formal Verification

## Phase B

## Team 6 - I-Group

## 1. Abstract

Phase B integrated software to create the full simulation software of the road system. The traffic signal lights were converted to a one-dimensional array to allow the array to be allocated to shared memory for the vehicles. The traffic signal sequence algorithm was modified to allow subarrays to rotate with respect to their intersections. The road was converted to a Numpy array which contained the current car slot and the associated the car id. Vehicle behavior stayed the same. After integration, the software was successfully simulated with 30 cars with no traffic violation, collisions, and U-turns.

## 2. Background

The project is to design a road system and verify its properties. All cars start in a common point in the road called point A which moves to other points labelled B, C, and D. The car must go to all the points in any order but must return to point A. For this project, two groups in a team deal with the road (I-Group) and vehicles (V-Group) separately. The project is also broken down to three phases. This report focuses on Phases B.

For phase B, the I-Group focused on integration of both programs. Vehicle behavior provided by V-Group would know traffic light signals made by I-Group. The road system created by I-Group should know where vehicles are located. The V-Group focused on identifying a model checking software that will be used in this project. A demonstration of the model checker will also be given by V-Group

## 3. Procedures

To allow integration, certain aspects of the program were changed. For the road system, a directed graph was being used to access the traffic signal keys while the road segments are converted into a Numpy two-dimensional array which would contain the 30 slots and the vehicle IDs. For the traffic signals, the data structure was converted from a two-dimensional array to a one-dimensional array. A subarray within the array specifies a certain intersection. Traffic signal sequence follows the same algorithm but done for each subarray.

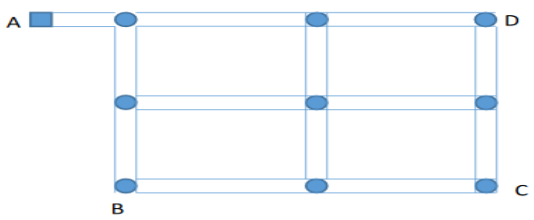


*Figure 1. Road System*

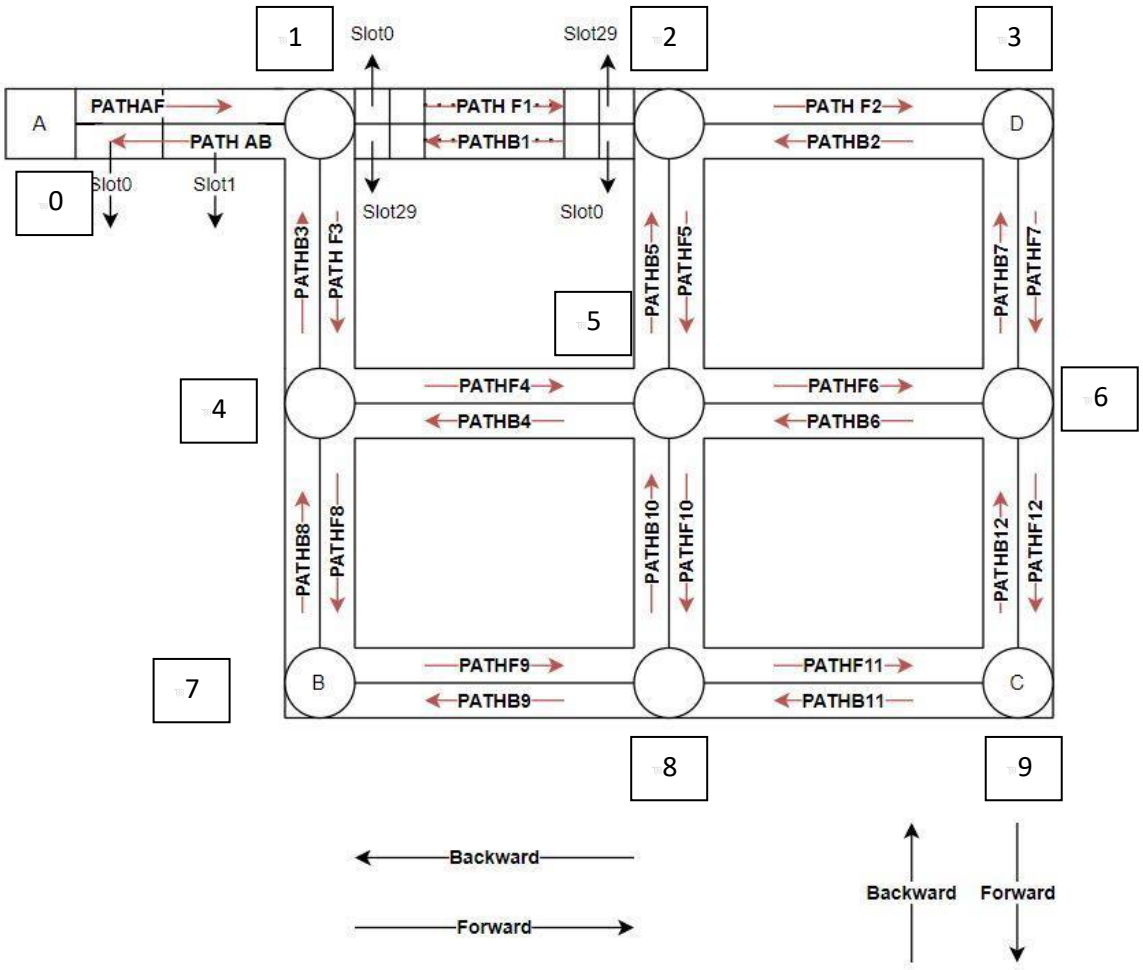


*Figure 2: Directed Graph of the Road System*

```
graph = {
    0: { 1: [road_seg_A, 0]},
    1: { 0: [road_seg[0], 25], 2: [road_seg[2], 3], 4: [road_seg[1], 10]},
    2: { 1: [road_seg[3], 2], 5: [road_seg[4], 14], 3: [road_seg[5], 6]},
    3: { 2: [road_seg[6], 5], 6: [road_seg[7], 17]},
    4: { 1: [road_seg[8], 1], 5: [road_seg[9], 11], 7: [road_seg[10], 19]},
    5: { 2: [road_seg[11], 4], 4: [road_seg[14], 9], 6: [road_seg[12], 15], 8: [road_seg[13], 22]},
    6: { 3: [road_seg[16], 7], 5: [road_seg[15], 13], 9: [road_seg[17], 24]},
    7: { 4: [road_seg[18], 8], 8: [road_seg[19], 20]},
    8: { 5: [road_seg[21], 12], 7: [road_seg[20], 18], 9: [road_seg[23], 23]},
    9: { 6: [road_seg[23], 16], 8: [road_seg[24], 21]}
}
```

*Figure 3: Graph Implementation*

In Figure 3, the graph [0][1] means access the road segment and traffic signal between 0 and 1. Each road segment between the intersections is 0.5 miles while the road between the starting point(A) and the first intersection is 1/30 of a mile. One road segment has 30 uniformed slots. To

help keep track of the cars in the road segment, a Numpy two-dimensional array is used to keep track of the slots and vehicle IDs.

For the traffic signal, the original implementation was to utilize a two-dimensional array where each ith index is the intersection and jth index is the current light signal of the road segment. Instead, the array has been converted to a one-dimensional array where a specified subarray is an intersection. This conversion is to allow the multiprocessing module in Python to put the array into a shared memory for the vehicles. To create a sequence, a subarray rotation algorithm was used to rotate the traffic signals. In Figure 4, 5, and 6, old the traffic signal code, the new traffic signal code, and the new rotation algorithm.

```
traff_sig = [
    [1,0,0],
    [0,1,0],
    [0,1],
    [1,0,0],
    [0,0,1,0],
    [0,1,0],
    [0,1],
    [0,0,1],
    [0,1]
]
```

*Figure 4: Old Traffic Code*

```
traff_sig_1d = [1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1]

traff_sig = mp.Array('i', traff_sig_1d)
```

*Figure 5: New Traffic Code*

```
def TrafficUpdate():
  while car_count.value:
    time.sleep(2)
    traff_sig[0:3] = traff_sig[1:3] + traff_sig[0:1]
    traff_sig[3:6] = traff_sig[4:6] + traff_sig[3:4]
    traff_sig[6:8] = traff_sig[7:8] + traff_sig[6:7]
    traff_sig[8:11] = traff_sig[9:11] + traff_sig[8:9]
    traff_sig[11:15] = traff_sig[12:15] + traff_sig[11:12]
    traff_sig[15:18] = traff_sig[16:18] + traff_sig[15:16]
    traff_sig[18:20] = traff_sig[19:20] + traff_sig[18:19]
    traff_sig[20:23] = traff_sig[21:23] + traff_sig[20:21]
    traff_sig[23:25] = traff_sig[24:25] + traff_sig[23:24]
    traff_sig[25] ^= 1
```

*Figure 6: New Rotation Algorithm*

Looking at Figure 4 and 5, the traffic signal is converted to a one-dimensional array to allow the traffic signal to be allocated in shared memory for the road system. For the traffic signal rotation in Figure 6, the rotation is based on the subarray which represents the intersection. Figure 7 shows the visual representation of the Figure 5 traffic signals.

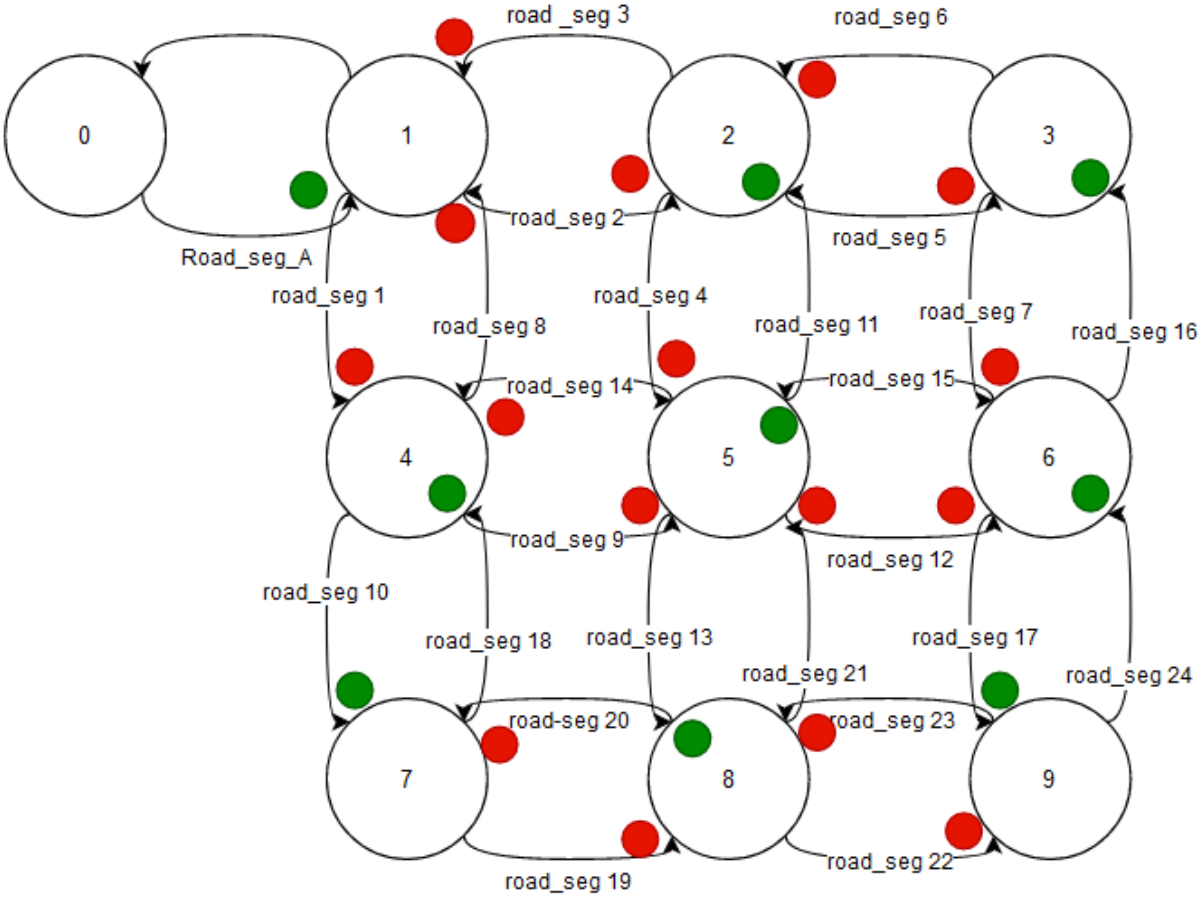


*Figure 7: Representation of Traffic Signals based on Figure 5*

For the simulation of the software, the original 15 cars have been increased to 30 cars. The simulation successfully finished as shown in Figure 8 with a throughput of 115 cars/hr, no red-light violation and collisions.

```
vehicle 12 is in pathB 1 and slot 11
vehicle 6 returned to A covering B C and D
vehicle 13 is in pathB 3 and slot 25
vehicle 10 is in pathAB and slot 0
vehicle 15 is in pathB 3 and slot 19
vehicle 11 is in pathB 1 and slot 29
vehicle 14 is in pathB 3 and slot 22
0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0
6
```

Car ID 6 Finished the course.

Next Traffic Signal Sequence

6 Cars Remaining

```
1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1
30
vehicle 9 is in pathB 11 and slot 28
vehicle 12 is in pathB 11 and slot 22
vehicle 16 is in pathB 12 and slot 21
vehicle 1 is in pathB 1 and slot 4
vehicle 4 is in pathB 2 and slot 26
vehicle 18 is in pathB 12 and slot 19
vehicle 23 is in pathB 11 and slot 16
vehicle 2 is in pathB 1 and slot 1
for vehicle 3 to move to next pathB 1 signal is red
```

For vehicle 3 to move to path B1, light must be green (1). Therefore, red light violation is avoided.

```
for vehicle 2 Slot 0 of pathAF is full
vehicle 1 is in pathAF and slot 0
for vehicle 3 Slot 0 of pathAF is full
for vehicle 4 Slot 0 of pathAF is full
for vehicle 5 Slot 0 of pathAF is full
for vehicle 6 Slot 0 of pathAF is full
for vehicle 7 Slot 0 of pathAF is full
for vehicle 8 Slot 0 of pathAF is full
for vehicle 9 Slot 0 of pathAF is full
for vehicle 10 Slot 0 of pathAF is full
for vehicle 11 Slot 0 of pathAF is full
for vehicle 12 Slot 0 of pathAF is full
for vehicle 13 Slot 0 of pathAF is full
for vehicle 14 Slot 0 of pathAF is full
for vehicle 15 Slot 0 of pathAF is full
for vehicle 16 Slot 0 of pathAF is full
for vehicle 17 Slot 0 of pathAF is full
for vehicle 19 Slot 0 of pathAF is full
for vehicle 18 Slot 0 of pathAF is full
for vehicle 20 Slot 0 of pathAF is full
for vehicle 21 Slot 0 of pathAF is full
for vehicle 23 Slot 0 of pathAF is full
for vehicle 22 Slot 0 of pathAF is full
for vehicle 24 Slot 0 of pathAF is full
for vehicle 25 Slot 0 of pathAF is full
for vehicle 27 Slot 0 of pathAF is full
for vehicle 26 Slot 0 of pathAF is full
for vehicle 28 Slot 0 of pathAF is full
for vehicle 29 Slot 0 of pathAF is full
for vehicle 30 Slot 0 of pathAF is full
```

While car 1 is in path AF slot 0, no cars cannot go to the $0^{th}$ slot. Therefore, collision is avoided.

```
vehicle 25 is in pathAB  and slot 1
1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0
1
vehicle 25 returned to A covering B C and D
0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1
0
Finished in 936.2 second(s)
```

30 cars/936s * 3600s/hr = 115 cars/hr

*Figure 8: Simulation Snippets*

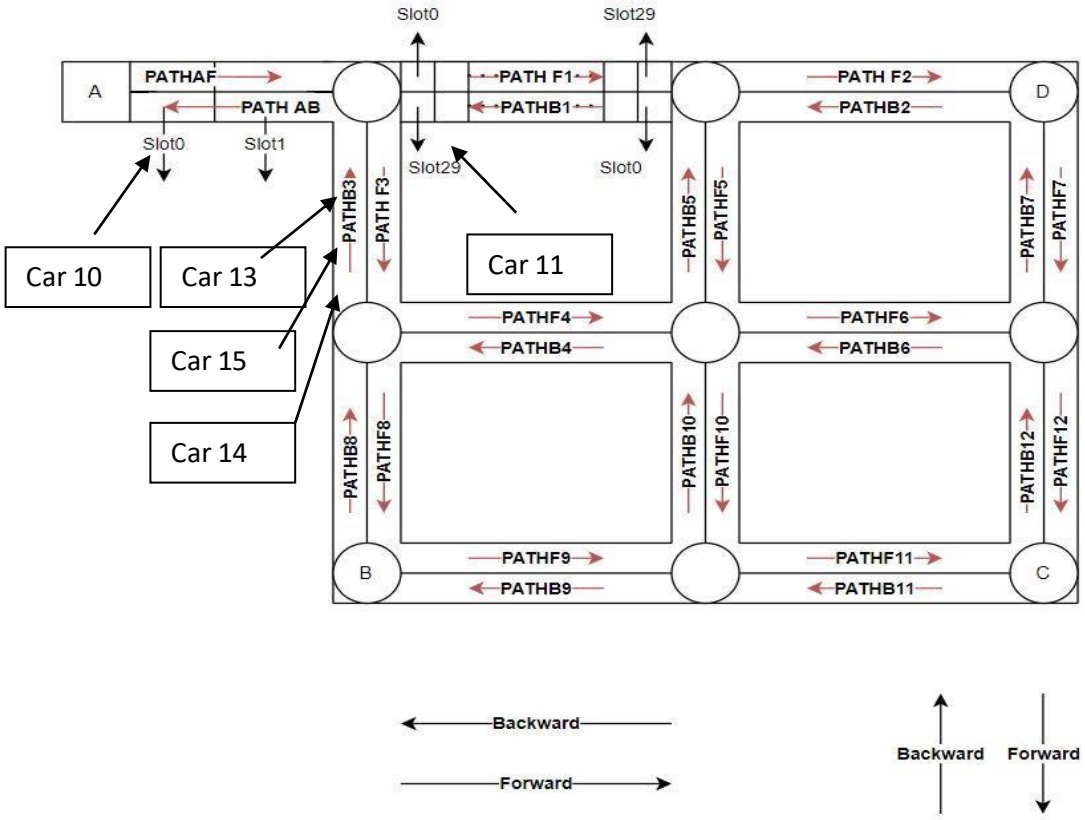


*Figure 9: Visual Representation of Cars from Figure 8*

# 4. Conclusion

For Phase B, I-Group was to integrate both programs. The traffic signal lights were converted to a one-dimensional array to allow the multiprocessor module to allocate it in shared memory for the vehicles. The road was converted to a Numpy array which makes the road compatible to vehicle behaviors. The vehicle behaviors provided by V-Group was not modified, but the number of vehicles was increased to 30 vehicles. The software successfully simulated with no violations meaning the integration was successful.

# 5. Appendix

```
import numpy as np
import ctypes as c
import multiprocessing as mp
from multiprocessing import Process,Array,Value
import time
from collections import deque
start=time.perf_counter()
#randon signals used for testing code
# signal=[0,1,0,1,0,1,1,1,1,1,1,1]

# traff_sig = [
#     [1,0,0], 0:3
#     [0,1,0], 3:6
#     [0,1],   6:8
#     [1,0,0], 8:11
#     [0,0,1,0], 11:15
#     [0,1,0], 15:18
#     [0,1],   18:20
#     [0,0,1], 20:23
#     [0,1],   23:25
#     [1]      25
# ]

traff_sig_1d = [1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1]

traff_sig = mp.Array('i', traff_sig_1d)

road_seg_A = deque([0 for _ in range(2)])
road_seg = [deque([0 for _ in range(30)])] * 25
car_count = mp.Value('i', 30)

graph = {
    0: { 1: [road_seg_A, 0]},
    1: { 0: [road_seg[0], 25], 2: [road_seg[2], 3], 4: [road_seg[1], 10]},
    2: { 1: [road_seg[3], 2], 5: [road_seg[4], 14], 3: [road_seg[5], 6]},
    3: { 2: [road_seg[6], 5], 6: [road_seg[7], 17]},
    4: { 1: [road_seg[8], 1], 5: [road_seg[9], 11], 7: [road_seg[10], 19]},
    5: { 2: [road_seg[11], 4], 4: [road_seg[14], 9], 6: [road_seg[12], 15]
, 8: [road_seg[13], 22]},
    6: { 3: [road_seg[16], 7], 5: [road_seg[15], 13], 9: [road_seg[17], 24
]},
    7: { 4: [road_seg[18], 8], 8: [road_seg[19], 20]},
```

```
    8: { 5: [road_seg[21], 12], 7: [road_seg[20], 18], 9: [road_seg[23], 2
3]},
    9: { 6: [road_seg[23], 16], 8: [road_seg[24], 21]}
}


#pathAF[slot]
m =  3
mp_pathAF = mp.Array(c.c_double, m) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
b = np.frombuffer(mp_pathAF.get_obj())
pathAF = b.reshape(m)

#pathAB[slot]
q = 3
mp_pathAB = mp.Array(c.c_double, q) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
a = np.frombuffer(mp_pathAB.get_obj())
pathAB = a.reshape(q)

#pathF[path_segment][slot]
r, s = 13, 30
mp_pathF = mp.Array(c.c_double, r*s) # shared, can be used from multiple p
rocesses
# then in each new process create a new numpy array using:
e = np.frombuffer(mp_pathF.get_obj())
pathF = e.reshape((r,s))

#pathb[path_segment][slot]
t,u  = 13, 30
mp_pathB = mp.Array(c.c_double,t*u) # shared, can be used from multiple pr
ocesses
# then in each new process create a new numpy array using:
d = np.frombuffer(mp_pathB.get_obj())
pathB = d.reshape((t,u))

def TrafficUpdate():
  while car_count.value:
    time.sleep(2)
    traff_sig[0:3] = traff_sig[1:3] + traff_sig[0:1]
    traff_sig[3:6] = traff_sig[4:6] + traff_sig[3:4]
    traff_sig[6:8] = traff_sig[7:8] + traff_sig[6:7]
    traff_sig[8:11] = traff_sig[9:11] + traff_sig[8:9]
```

```
    traff_sig[11:15] = traff_sig[12:15] + traff_sig[11:12]
    traff_sig[15:18] = traff_sig[16:18] + traff_sig[15:16]
    traff_sig[18:20] = traff_sig[19:20] + traff_sig[18:19]
    traff_sig[20:23] = traff_sig[21:23] + traff_sig[20:21]
    traff_sig[23:25] = traff_sig[24:25] + traff_sig[23:24]
    traff_sig[25] ^= 1
    print(*traff_sig)
    print(car_count.value)

#vehicle generation and insertion
def GenerateVehicle(id,pnext_num,tl):
  count=0
  count_a=0
  count_b=0
  intersect = tl
  for i in range(0,2):
    while(pathAF[i]!=id):
      if (pathAF[i]==0):
        pathAF[i] = id
        pathAF[i-1]=0
        print("vehicle %d is in pathAF and slot %d" %(id,i))
      else:
        count=count+1
        if(count==1):
         print("for vehicle %d Slot %d of pathAF is full"%(id,i))
      time.sleep(2)
    if(i == 1):
      while(pathF[pnext_num][0]!=id):
        if (traff_sig[intersect]==1):
          if(pathF[pnext_num][0]==0):
            pathF[pnext_num][0]=id
            pathAF[1]=0
            print("vehicle %d is in pathF %d and slot 0" %(id,pnext_num))
          else:
            count_a=count_a+1
            if(count_a==1):
             print("for vehicle %d signal is green but Slot %d of pathF %d
 is full"%(id,0,pnext_num))
        else:
          count_b=count_b+1
          if(count_b==1):
           print("for vehicle %d to move to next pathF %d signal is red"%(
id,pnext_num))
        time.sleep(2)
```

```
#vehicle returning to A '''
def ReturnVehicle(id,p_num,tl):
   global car_count
   intersect = tl
   count=0
   for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathAB[0]!=id):
            if(traff_sig[intersect]==1):
                if(pathAB[0]==0):
                     pathAB[0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathAB and slot 0" %(id))
                else:
                  count=count+1
                  if(count==1):
                     print("for vehicle %d signal is green but next pathAB
  slot %d is not empty"%(id,0))
            else:
              count=count+1
              if(count==1):
               print("for vehicle %d to move to next pathAB %d signal is r
ed"%(id, 25) )
            time.sleep(2)
   if(pathAB[0]==id):
     while(pathAB[1]!=id):
      if(pathAB[1]==0):
        pathAB[1]=id;
        pathAB[0]=0;
        print("vehicle %d is in pathAB  and slot 1" %(id))
      else:
        count=count+1
        if(count==1):
         print("for vehicle %d next pathAB slot %d is not empty"%(id,1))
```

```
      time.sleep(2)
   if(pathAB[1]==id):
        pathAB[1]=0
        print("vehicle %d returned to A covering B C and D" %(id))
        with car_count.get_lock():
          car_count.value -= 1



def TravF(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathF[p_num][i]!=id):
        if (pathF[p_num][i])==0:
            pathF[p_num][i]=id
            pathF[p_num][i-1]=0
            print("vehicle %d is in pathF %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathF %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathF[pnext_num][0]!=id):
            if(traff_sig[intersect]==1):
                if(pathF[pnext_num][0]==0):
                     pathF[pnext_num][0]=id
                     pathF[p_num][29]=0
                     print("vehicle %d is in pathF %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathF
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
               print("for vehicle %d to move to next pathF%d signal is red
"%(id,pnext_num) )
            time.sleep(2)
```

```
def TravB(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathB[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathB[pnext_num][0]==0):
                     pathB[pnext_num][0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathB %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathB
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
               print("for vehicle %d to move to next pathB %d signal is re
d"%(id,pnext_num))
            time.sleep(2)


def TravFtoB(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
```

```
      while(pathF[p_num][i]!=id):
        if (pathF[p_num][i])==0:
            pathF[p_num][i]=id
            pathF[p_num][i-1]=0
            print("vehicle %d is in pathF %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
            print("for vehicle %d next pathF %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathB[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathB[pnext_num][0]==0):
                     pathB[pnext_num][0]=id
                     pathF[p_num][29]=0
                     print("vehicle %d is in pathB %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathB
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
                print("for vehicle %d to move to next pathB %d signal is r
ed"%(id,pnext_num) )
            time.sleep(2)

def TravBtoF(id,p_num,tl,pnext_num):
    count=0
    count_a=0
    count_b=0
    intersect = tl
    for i in range (1,30):
      while(pathB[p_num][i]!=id):
        if (pathB[p_num][i])==0:
            pathB[p_num][i]=id
            pathB[p_num][i-1]=0
            print("vehicle %d is in pathB %d and slot %d" %(id,p_num,i))
        else:
          count=count+1
          if(count==1):
```

```
            print("for vehicle %d next pathB %d slot %d is not empty" %(id
,p_num,i))
        time.sleep(2)
      if (i==29):
           while(pathF[pnext_num][0]!=id):
            if(traff_sig[intersect] ==1):
                if(pathF[pnext_num][0]==0):
                     pathF[pnext_num][0]=id
                     pathB[p_num][29]=0
                     print("vehicle %d is in pathF %d and slot 0" %(id,pne
xt_num))
                else:
                  count_a=count_a+1
                  if(count_a==1):
                     print("for vehicle %d signal is green but next pathF
%d slot %d is not empty"%(id,pnext_num,0))
            else:
              count_b=count_b+1
              if(count_b==1):
                print("for vehicle %d to move to next pathF %d signal is r
ed"%(id,pnext_num) )
            time.sleep(2)


def path1_vehicle1():
   GenerateVehicle(1,3,graph[0][1][1])
   TravF(1,3,graph[1][4][1],8)
   TravF(1,8,graph[4][7][1],9)
   TravF(1,9,graph[7][8][1],11)
   TravFtoB(1,11,graph[8][9][1],12)
   TravB(1,12,graph[9][6][1],7)
   TravB(1,7,graph[6][3][1],2)
   TravB(1,2,graph[3][2][1],1)
   ReturnVehicle(1,1,graph[2][1][1])
def path1_vehicle2():
   GenerateVehicle(2,3,graph[0][1][1])
   TravF(2,3,graph[1][4][1],8)
   TravF(2,8,graph[4][7][1],9)
   TravF(2,9,graph[7][8][1],11)
   TravFtoB(2,11,graph[8][9][1],12)
   TravB(2,12,graph[9][6][1],7)
   TravB(2,7,graph[6][3][1],2)
   TravB(2,2,graph[3][2][1],1)
   ReturnVehicle(2,1,graph[2][1][1])
def path1_vehicle3():
```

```
    GenerateVehicle(3,3,graph[0][1][1])
    TravF(3,3,graph[1][4][1],8)
    TravF(3,8,graph[4][7][1],9)
    TravF(3,9,graph[7][8][1],11)
    TravFtoB(3,11,graph[8][9][1],12)
    TravB(3,12,graph[9][6][1],7)
    TravB(3,7,graph[6][3][1],2)
    TravB(3,2,graph[3][2][1],1)
    ReturnVehicle(3,1,graph[2][1][1])
def path2_vehicle4():
    GenerateVehicle(4,1,graph[0][1][1])
    TravF(4,1,graph[1][2][1],5)
    TravF(4,5,graph[2][5][1],10)
    TravF(4,10,graph[5][8][1],11)
    TravFtoB(4,11,graph[8][9][1],12)
    TravB(4,12,graph[9][6][1],7)
    TravB(4,7,graph[6][3][1],2)
    TravBtoF(4,2,graph[3][2][1],5)
    TravF(4,5,graph[2][5][1],10)
    TravFtoB(4,10,graph[5][8][1],9)
    TravB(4,9,graph[8][7][1],8)
    TravB(4,8,graph[7][4][1],3)
    ReturnVehicle(4,3,graph[4][1][1])
def path2_vehicle5():
    GenerateVehicle(5,1,graph[0][1][1])
    TravF(5,1,graph[1][2][1],5)
    TravF(5,5,graph[2][5][1],10)
    TravF(5,10,graph[5][8][1],11)
    TravFtoB(5,11,graph[8][9][1],12)
    TravB(5,12,graph[9][6][1],7)
    TravB(5,7,graph[6][3][1],2)
    TravBtoF(5,2,graph[3][2][1],5)
    TravF(5,5,graph[2][5][1],10)
    TravFtoB(4,10,graph[5][8][1],9)
    TravB(5,9,graph[8][7][1],8)
    TravB(5,8,graph[7][4][1],3)
    ReturnVehicle(5,3,graph[4][1][1])
def path2_vehicle6():
    GenerateVehicle(6,1,graph[0][1][1])
    TravF(6,1,graph[1][2][1],5)
    TravF(6,5,graph[2][5][1],10)
    TravF(6,10,graph[5][8][1],11)
    TravFtoB(6,11,graph[8][9][1],12)
    TravB(6,12,graph[9][6][1],7)
    TravB(6,7,graph[6][3][1],2)
```

```
    TravBtoF(6,2,graph[3][2][1],5)
    TravF(6,5,graph[2][5][1],10)
    TravFtoB(6,10,graph[5][8][1],9)
    TravB(6,9,graph[8][7][1],8)
    TravB(6,8,graph[7][4][1],3)
    ReturnVehicle(6,3,graph[4][1][1])
def path3_vehicle7():
  GenerateVehicle(7,1,graph[0][1][1])
  TravF(7,1,graph[1][2][1],2)
  TravF(7,2,graph[2][3][1],7)
  TravF(7,7,graph[3][6][1],12)
  TravFtoB(7,12,graph[6][9][1],11)
  TravB(7,11,graph[9][8][1],9)
  TravB(7,9,graph[8][7][1],8)
  TravB(7,8,graph[7][4][1],3)
  ReturnVehicle(7,3,graph[4][1][1])
def path3_vehicle8():
  GenerateVehicle(8,1,graph[0][1][1])
  TravF(8,1,graph[1][2][1],2)
  TravF(8,2,graph[2][3][1],7)
  TravF(8,7,graph[3][6][1],12)
  TravFtoB(8,12,graph[6][9][1],11)
  TravB(8,11,graph[9][8][1],9)
  TravB(8,9,graph[8][7][1],8)
  TravB(8,8,graph[7][4][1],3)
  ReturnVehicle(8,3,graph[4][1][1])
def path3_vehicle9():
  GenerateVehicle(9,1,graph[0][1][1])
  TravF(9,1,graph[1][2][1],2)
  TravF(9,2,graph[2][3][1],7)
  TravF(9,7,graph[3][6][1],12)
  TravFtoB(9,12,graph[6][9][1],11)
  TravB(9,11,graph[9][8][1],9)
  TravB(9,9,graph[8][7][1],8)
  TravB(9,8,graph[7][4][1],3)
  ReturnVehicle(9,3,graph[4][1][1])
def path4_vehicle10():
  GenerateVehicle(10,3,graph[0][1][1])
  TravF(10,3,graph[1][4][1],4)
  TravF(10,4,graph[4][5][1],6)
  TravFtoB(10,6,graph[5][6][1],12)
  TravB(10,12,graph[6][9][1],11)
  TravB(10,11,graph[9][8][1],9)
  TravB(10,9,graph[8][7][1],8)
  TravBtoF(10,8,graph[7][4][1],4)
```

```
  TravF(10,4,graph[4][5][1],6)
  TravFtoB(10,6,graph[5][6][1],7)
  TravB(10,7,graph[6][3][1],2)
  TravB(10,2,graph[3][2][1],1)
  ReturnVehicle(10,1,graph[2][1][1])
def path4_vehicle11():
  GenerateVehicle(11,3,graph[0][1][1])
  TravF(11,3,graph[1][4][1],4)
  TravF(11,4,graph[4][5][1],6)
  TravFtoB(11,6,graph[5][6][1],12)
  TravB(11,12,graph[6][9][1],11)
  TravB(11,11,graph[9][8][1],9)
  TravB(11,9,graph[8][7][1],8)
  TravBtoF(11,8,graph[7][4][1],4)
  TravF(11,4,graph[4][5][1],6)
  TravFtoB(11,6,graph[5][6][1],7)
  TravB(11,7,graph[6][3][1],2)
  TravB(11,2,graph[3][2][1],1)
  ReturnVehicle(11,1,graph[2][1][1])
def path4_vehicle12():
  GenerateVehicle(12,3,graph[0][1][1])
  TravF(12,3,graph[1][4][1],4)
  TravF(12,4,graph[4][5][1],6)
  TravFtoB(12,6,graph[5][6][1],12)
  TravB(12,12,graph[6][9][1],11)
  TravB(12,11,graph[9][8][1],9)
  TravB(12,9,graph[8][7][1],8)
  TravBtoF(12,8,graph[7][4][1],4)
  TravF(12,4,graph[4][5][1],6)
  TravFtoB(12,6,graph[5][6][1],7)
  TravB(12,7,graph[6][3][1],2)
  TravB(12,2,graph[3][2][1],1)
  ReturnVehicle(12,1,graph[2][1][1])
def path5_vehicle13():
  GenerateVehicle(13,3,graph[0][1][1])
  TravF(13,3,graph[1][4][1],8)
  TravF(13,8,graph[4][7][1],9)
  TravFtoB(13,9,graph[7][8][1],10)
  TravB(13,10,graph[8][5][1],5)
  TravBtoF(13,5,graph[5][2][1],2)
  TravF(13,2,graph[2][3][1],7)
  TravF(13,7,graph[3][6][1],12)
  TravFtoB(13,12,graph[6][9][1],11)
  TravB(13,11,graph[9][8][1],9)
  TravB(13,9,graph[8][5][1],8)
```

```
  TravB(13,8,graph[5][2][1],3)
  ReturnVehicle(13,3,graph[2][1][1])
def path5_vehicle14():
  GenerateVehicle(14,3,graph[0][1][1])
  TravF(14,3,graph[1][4][1],8)
  TravF(14,8,graph[4][7][1],9)
  TravFtoB(14,9,graph[7][8][1],10)
  TravB(14,10,graph[8][5][1],5)
  TravBtoF(14,5,graph[5][2][1],2)
  TravF(14,2,graph[2][3][1],7)
  TravF(14,7,graph[3][6][1],12)
  TravFtoB(14,12,graph[6][9][1],11)
  TravB(14,11,graph[9][8][1],9)
  TravB(14,9,graph[8][5][1],8)
  TravB(14,8,graph[5][2][1],3)
  ReturnVehicle(14,3,graph[2][1][1])
def path5_vehicle15():
  GenerateVehicle(15,3,graph[0][1][1])
  TravF(15,3,graph[1][4][1],8)
  TravF(15,8,graph[4][7][1],9)
  TravFtoB(15,9,graph[7][8][1],10)
  TravB(15,10,graph[8][5][1],5)
  TravBtoF(15,5,graph[5][2][1],2)
  TravF(15,2,graph[2][3][1],7)
  TravF(15,7,graph[3][6][1],12)
  TravFtoB(15,12,graph[6][9][1],11)
  TravB(15,11,graph[9][8][1],9)
  TravB(15,9,graph[8][5][1],8)
  TravB(15,8,graph[5][2][1],3)
  ReturnVehicle(15,3,graph[2][1][1])

def path1_vehicle16():
   GenerateVehicle(16,3,graph[0][1][1])
   TravF(16,3,graph[1][4][1],8)
   TravF(16,8,graph[4][7][1],9)
   TravF(16,9,graph[7][8][1],11)
   TravFtoB(16,11,graph[8][9][1],12)
   TravB(16,12,graph[9][6][1],7)
   TravB(16,7,graph[6][3][1],2)
   TravB(16,2,graph[3][2][1],1)
   ReturnVehicle(16,1,graph[2][1][1])
def path1_vehicle17():
   GenerateVehicle(17,3,graph[0][1][1])
   TravF(17,3,graph[1][4][1],8)
   TravF(17,8,graph[4][7][1],9)
```

```
    TravF(17,9,graph[7][8][1],11)
    TravFtoB(17,11,graph[8][9][1],12)
    TravB(17,12,graph[9][6][1],7)
    TravB(17,7,graph[6][3][1],2)
    TravB(17,2,graph[3][2][1],1)
    ReturnVehicle(17,1,graph[2][1][1])
def path1_vehicle18():
    GenerateVehicle(18,3,graph[0][1][1])
    TravF(18,3,graph[1][4][1],8)
    TravF(18,8,graph[4][7][1],9)
    TravF(18,9,graph[7][8][1],11)
    TravFtoB(18,11,graph[8][9][1],12)
    TravB(18,12,graph[9][6][1],7)
    TravB(18,7,graph[6][3][1],2)
    TravB(18,2,graph[3][2][1],1)
    ReturnVehicle(18,1,graph[2][1][1])
def path2_vehicle19():
    GenerateVehicle(19,1,graph[0][1][1])
    TravF(19,1,graph[1][2][1],5)
    TravF(19,5,graph[2][5][1],10)
    TravF(19,10,graph[5][8][1],11)
    TravFtoB(19,11,graph[8][9][1],12)
    TravB(19,12,graph[9][6][1],7)
    TravB(19,7,graph[6][3][1],2)
    TravBtoF(19,2,graph[3][2][1],5)
    TravF(19,5,graph[2][5][1],10)
    TravFtoB(19,10,graph[5][8][1],9)
    TravB(19,9,graph[8][7][1],8)
    TravB(19,8,graph[7][4][1],3)
    ReturnVehicle(19,3,graph[4][1][1])
def path2_vehicle20():
    GenerateVehicle(20,1,graph[0][1][1])
    TravF(20,1,graph[1][2][1],5)
    TravF(20,5,graph[2][5][1],10)
    TravF(20,10,graph[5][8][1],11)
    TravFtoB(20,11,graph[8][9][1],12)
    TravB(20,12,graph[9][6][1],7)
    TravB(20,7,graph[6][3][1],2)
    TravBtoF(20,2,graph[3][2][1],5)
    TravF(20,5,graph[2][5][1],10)
    TravFtoB(20,10,graph[5][8][1],9)
    TravB(20,9,graph[8][7][1],8)
    TravB(20,8,graph[7][4][1],3)
    ReturnVehicle(20,3,graph[4][1][1])
def path2_vehicle21():
```

```
    GenerateVehicle(21,1,graph[0][1][1])
    TravF(21,1,graph[1][2][1],5)
    TravF(21,5,graph[2][5][1],10)
    TravF(21,10,graph[5][8][1],11)
    TravFtoB(21,11,graph[8][9][1],12)
    TravB(21,12,graph[9][6][1],7)
    TravB(21,7,graph[6][3][1],2)
    TravBtoF(21,2,graph[3][2][1],5)
    TravF(21,5,graph[2][5][1],10)
    TravFtoB(21,10,graph[5][8][1],9)
    TravB(21,9,graph[8][7][1],8)
    TravB(21,8,graph[7][4][1],3)
    ReturnVehicle(21,3,graph[4][1][1])
def path3_vehicle22():
  GenerateVehicle(22,1,graph[0][1][1])
  TravF(22,1,graph[1][2][1],2)
  TravF(22,2,graph[2][3][1],7)
  TravF(22,7,graph[3][6][1],12)
  TravFtoB(22,12,graph[6][9][1],11)
  TravB(22,11,graph[9][8][1],9)
  TravB(22,9,graph[8][7][1],8)
  TravB(22,8,graph[7][4][1],3)
  ReturnVehicle(22,3,graph[4][1][1])
def path3_vehicle23():
  GenerateVehicle(23,1,graph[0][1][1])
  TravF(23,1,graph[1][2][1],2)
  TravF(23,2,graph[2][3][1],7)
  TravF(23,7,graph[3][6][1],12)
  TravFtoB(23,12,graph[6][9][1],11)
  TravB(23,11,graph[9][8][1],9)
  TravB(23,9,graph[8][7][1],8)
  TravB(23,8,graph[7][4][1],3)
  ReturnVehicle(23,3,graph[4][1][1])
def path3_vehicle24():
  GenerateVehicle(24,1,graph[0][1][1])
  TravF(24,1,graph[1][2][1],2)
  TravF(24,2,graph[2][3][1],7)
  TravF(24,7,graph[3][6][1],12)
  TravFtoB(24,12,graph[6][9][1],11)
  TravB(24,11,graph[9][8][1],9)
  TravB(24,9,graph[8][7][1],8)
  TravB(24,8,graph[7][4][1],3)
  ReturnVehicle(24,3,graph[4][1][1])
def path4_vehicle25():
  GenerateVehicle(25,3,graph[0][1][1])
```

```
    TravF(25,3,graph[1][4][1],4)
    TravF(25,4,graph[4][5][1],6)
    TravFtoB(25,6,graph[5][6][1],12)
    TravB(25,12,graph[6][9][1],11)
    TravB(25,11,graph[9][8][1],9)
    TravB(25,9,graph[8][7][1],8)
    TravBtoF(25,8,graph[7][4][1],4)
    TravF(25,4,graph[4][5][1],6)
    TravFtoB(25,6,graph[5][6][1],7)
    TravB(25,7,graph[6][3][1],2)
    TravB(25,2,graph[3][2][1],1)
    ReturnVehicle(25,1,graph[2][1][1])
  def path4_vehicle26():
    GenerateVehicle(26,3,graph[0][1][1])
    TravF(26,3,graph[1][4][1],4)
    TravF(26,4,graph[4][5][1],6)
    TravFtoB(26,6,graph[5][6][1],12)
    TravB(26,12,graph[6][9][1],11)
    TravB(26,11,graph[9][8][1],9)
    TravB(26,9,graph[8][7][1],8)
    TravBtoF(26,8,graph[7][4][1],4)
    TravF(26,4,graph[4][5][1],6)
    TravFtoB(26,6,graph[5][6][1],7)
    TravB(26,7,graph[6][3][1],2)
    TravB(26,2,graph[3][2][1],1)
    ReturnVehicle(26,1,graph[2][1][1])
  def path4_vehicle27():
    GenerateVehicle(27,3,graph[0][1][1])
    TravF(27,3,graph[1][4][1],4)
    TravF(27,4,graph[4][5][1],6)
    TravFtoB(27,6,graph[5][6][1],12)
    TravB(27,12,graph[6][9][1],11)
    TravB(27,11,graph[9][8][1],9)
    TravB(27,9,graph[8][7][1],8)
    TravBtoF(27,8,graph[7][4][1],4)
    TravF(27,4,graph[4][5][1],6)
    TravFtoB(27,6,graph[5][6][1],7)
    TravB(27,7,graph[6][3][1],2)
    TravB(27,2,graph[3][2][1],1)
    ReturnVehicle(27,1,graph[2][1][1])
  def path5_vehicle28():
    GenerateVehicle(28,3,graph[0][1][1])
    TravF(28,3,graph[1][4][1],8)
    TravF(28,8,graph[4][7][1],9)
    TravFtoB(28,9,graph[7][8][1],10)
```

```
    TravB(28,10,graph[8][5][1],5)
    TravBtoF(28,5,graph[5][2][1],2)
    TravF(28,2,graph[2][3][1],7)
    TravF(28,7,graph[3][6][1],12)
    TravFtoB(28,12,graph[6][9][1],11)
    TravB(28,11,graph[9][8][1],9)
    TravB(28,9,graph[8][5][1],8)
    TravB(28,8,graph[5][2][1],3)
    ReturnVehicle(28,3,graph[2][1][1])
  def path5_vehicle29():
    GenerateVehicle(29,3,graph[0][1][1])
    TravF(29,3,graph[1][4][1],8)
    TravF(29,8,graph[4][7][1],9)
    TravFtoB(29,9,graph[7][8][1],10)
    TravB(29,10,graph[8][5][1],5)
    TravBtoF(29,5,graph[5][2][1],2)
    TravF(29,2,graph[2][3][1],7)
    TravF(29,7,graph[3][6][1],12)
    TravFtoB(29,12,graph[6][9][1],11)
    TravB(29,11,graph[9][8][1],9)
    TravB(29,9,graph[8][5][1],8)
    TravB(29,8,graph[5][2][1],3)
    ReturnVehicle(29,3,graph[2][1][1])
  def path5_vehicle30():
    GenerateVehicle(30,3,graph[0][1][1])
    TravF(30,3,graph[1][4][1],8)
    TravF(30,8,graph[4][7][1],9)
    TravFtoB(30,9,graph[7][8][1],10)
    TravB(30,10,graph[8][5][1],5)
    TravBtoF(30,5,graph[5][2][1],2)
    TravF(30,2,graph[2][3][1],7)
    TravF(30,7,graph[3][6][1],12)
    TravFtoB(30,12,graph[6][9][1],11)
    TravB(30,11,graph[9][8][1],9)
    TravB(30,9,graph[8][5][1],8)
    TravB(30,8,graph[5][2][1],3)
    ReturnVehicle(30,3,graph[2][1][1])



traff = mp.Process(target=TrafficUpdate)
p1=mp.Process(target=path1_vehicle1)
p2=mp.Process(target=path1_vehicle2)
```

```
p3=mp.Process(target=path1_vehicle3)
p4=mp.Process(target=path2_vehicle4)
p5=mp.Process(target=path2_vehicle5)
p6=mp.Process(target=path2_vehicle6)
p7=mp.Process(target=path3_vehicle7)
p8=mp.Process(target=path3_vehicle8)
p9=mp.Process(target=path3_vehicle9)
p10=mp.Process(target=path4_vehicle10)
p11=mp.Process(target=path4_vehicle11)
p12=mp.Process(target=path4_vehicle12)
p13=mp.Process(target=path5_vehicle13)
p14=mp.Process(target=path5_vehicle14)
p15=mp.Process(target=path5_vehicle15)
p16=mp.Process(target=path1_vehicle16)
p17=mp.Process(target=path1_vehicle17)
p18=mp.Process(target=path1_vehicle18)
p19=mp.Process(target=path2_vehicle19)
p20=mp.Process(target=path2_vehicle20)
p21=mp.Process(target=path2_vehicle21)
p22=mp.Process(target=path3_vehicle22)
p23=mp.Process(target=path3_vehicle23)
p24=mp.Process(target=path3_vehicle24)
p25=mp.Process(target=path4_vehicle25)
p26=mp.Process(target=path4_vehicle26)
p27=mp.Process(target=path4_vehicle27)
p28=mp.Process(target=path5_vehicle28)
p29=mp.Process(target=path5_vehicle29)
p30=mp.Process(target=path5_vehicle30)

traff.start()
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()
p7.start()
p8.start()
p9.start()
p10.start()
p11.start()
p12.start()
p13.start()
p14.start()
p15.start()
```

```
p16.start()
p17.start()
p18.start()
p19.start()
p20.start()
p21.start()
p22.start()
p23.start()
p24.start()
p25.start()
p26.start()
p27.start()
p28.start()
p29.start()
p30.start()

traff.join()
p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()
p7.join()
p8.join()
p9.join()
p10.join()
p11.join()
p12.join()
p13.join()
p14.join()
p15.join()
p16.join()
p17.join()
p18.join()
p19.join()
p20.join()
p21.join()
p22.join()
p23.join()
p24.join()
p25.join()
p26.join()
p27.join()
p28.join()
```

```
p29.join()
p30.join()


finish=time.perf_counter()

print(f'Finished in {round(finish-start,2)} second(s)')
```