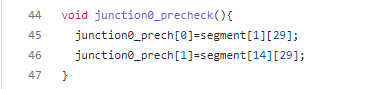
The functions in file **crash.h** are used to calculate different crashes and rule violations in the designed system.

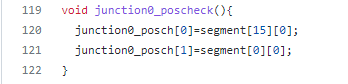
void junction0\_precheck():



The function junction0\_precheck() initializes the array of slots to be checked at the junction 0. These are the slots at the junctions before car moves to the next segment. For instance, junction0\_prech[0] indicates the last slot in segment 1 i.e. segment[1][29]. Similarly, junction0\_prech[1] indicates the last slot in segment 14 i.e. segment[14][29]. If car is in these slots at current time, for the next time step the cars will move to another segment if signal is green.

On the similar lines, arrays are initialised for each junction. Since, there are 9 junctions, 9 arrays junction1\_precheck(), junction2\_precheck(), junction3\_precheck() upto junction8\_precheck() are initialised.

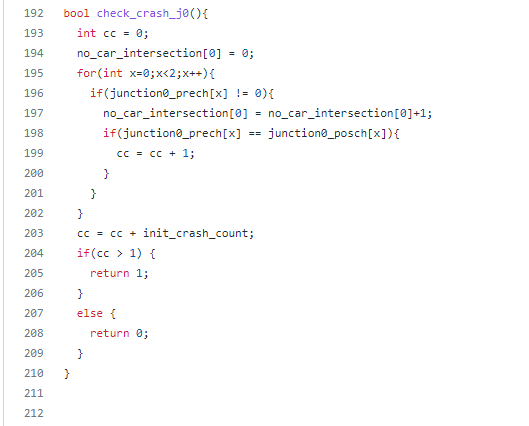
void junction0\_poscheck():



The function junction0\_ poscheck () initializes the array of slots to be checked at the junction 0 after next time step. These are the slots at the junctions where car will move to the next segment in next time step. For instance, junction0\_ poscheck[0] indicates that the car in slot segment[1][29] will move to segment[15][0] in next time step if signal is green. Similarly, junction0\_ poscheck[0] indicates that the car in slot segment[41][29] will move to segment[0][0] in next time step if signal is green. The junction\*\_posch array has list of all the possible slots where can move in next time step at a junction if signal is green and next slot is empty.

On the similar lines, arrays are initialised for each junction. Since, there are 9 junctions, 9 arrays junction1\_ poscheck (), junction2\_ poscheck (), junction3\_ poscheck () upto junction8\_ poscheck () are initialised.

bool check\_crash\_j0():



This function returns true if there is crash at junction 0. A crash will occur if two or more cars cross the intersection at the same time. To identify a crash junction0\_ precheck and junction0\_ poscheck arrays are used. If a car has moved to next segment by crossing intersection/junction then the value of junction0\_ precheck will be equal to junction0\_ poscheck. Here, a non-zero check was added as 0 indicates no car. If more than two values in arrays match then it can be interpreted as a crash. Only for junction 0, the initial crashes are also counted when two cars try to return at A at the same time.

Similarly, a crash check is implemented from all the 9 junctions.

void junction\_pre\_check\_for\_crash():

Calling all the junction\_pre\_check functions for 9 junctions and initializing junction\*\_prech[] arrays.

void junction\_pos\_check\_for\_crash()

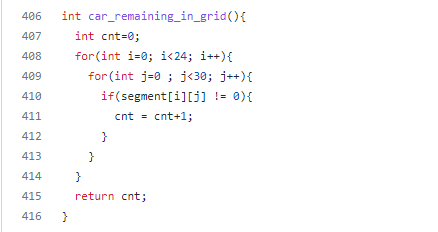
Calling all the junction\_pos\_check functions for 9 junctions and initializing junction\*\_posch[] arrays.

int check\_crash():

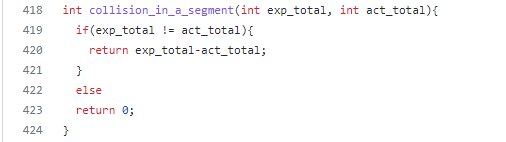
The function returns the total number of junctions suffered by crashes.

car\_remaining\_in\_grid():

This function returns total number of cars present in the grid at a time. The segment array is traversed and non-zero values are counted as the non-zero values indicate the car IDs.

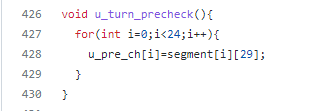


int collision\_in\_a\_segment(int exp\_total, int act\_total):



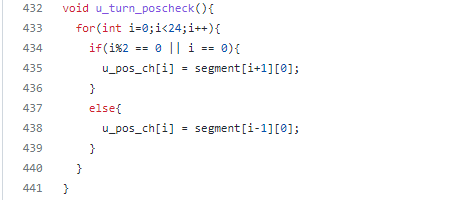
The function returns the collision in a segment. If Car 1 moves to next slot in the same segment even though there is Car 2 in next slot, the collision in segment will occur. If such collision happens, the ID of Car 2 in the next segment will be overwritten by Car 1’s ID. And the value of slot where Car 1 was present will be assigned the value zero. Hence, Car 2 will disappear from the grid. The difference between sum of cars present on a grid & cars returned and total generated cars is used to identify the collisions in a segment.

u\_turn\_precheck():



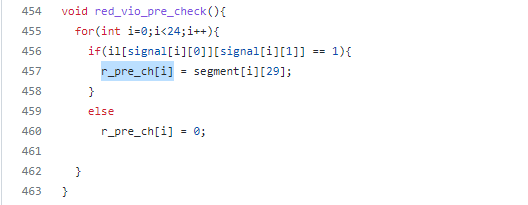
The function u\_turn\_precheck() initializes an array which stores the car ID present in end of each segment. For example, the first element of array u\_pos\_ch will store the car ID present in segment[0][29]. The segment[0][29] is the last slot of segment 0.

void u\_turn\_poscheck()



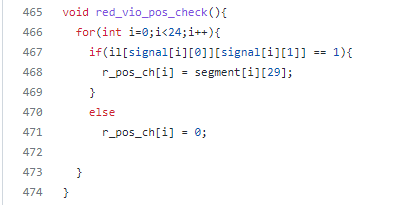
This function also in initializes an array which stores the car ID present in start of next segment if a car takes U-turn in next time step. The first element of array u\_pos\_ch[] will store the car ID present in segment[0][29]. The segment[0][29] is the last slot of segment 0. Hence, if the car is present at segment[0][29] at current time stamp and it takes a U-turn in next time step, it will go to segment[1][0]. If a car present at segment[1][29] at current time stamp and it takes a U-turn in next time step, it will go to segment[0][0]. Hence, the values of segment[0][0] and segment[1][0] are stored in the array u\_pos\_ch[]. Similarly values at stored for all 24 segments.

void red\_vio\_pre\_check():



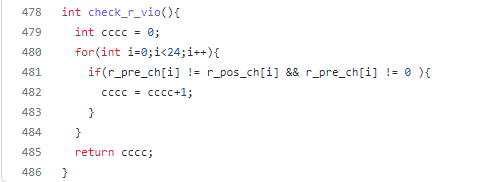
In this function, an array r\_pre\_ch[] stores the values of car IDs present in the last slot of each segment i.e. segment[i][29] where i = 0, 1, …23 if the signal for that car is RED at current time. For other signals GREEN or NO Signal, a zero is stored in the array. This is used to calculate red signal violations.

void red\_vio\_pos\_check():



In this function, an array r\_pos\_ch[] stores the values of car IDs present in the last slot of each segment i.e. segment[i][29] where i = 0, 1, ..., 23 for next time step. For other signals GREEN or NO Signal, a zero is stored in the array. This is used to calculate red signal violations.

int check\_r\_vio():

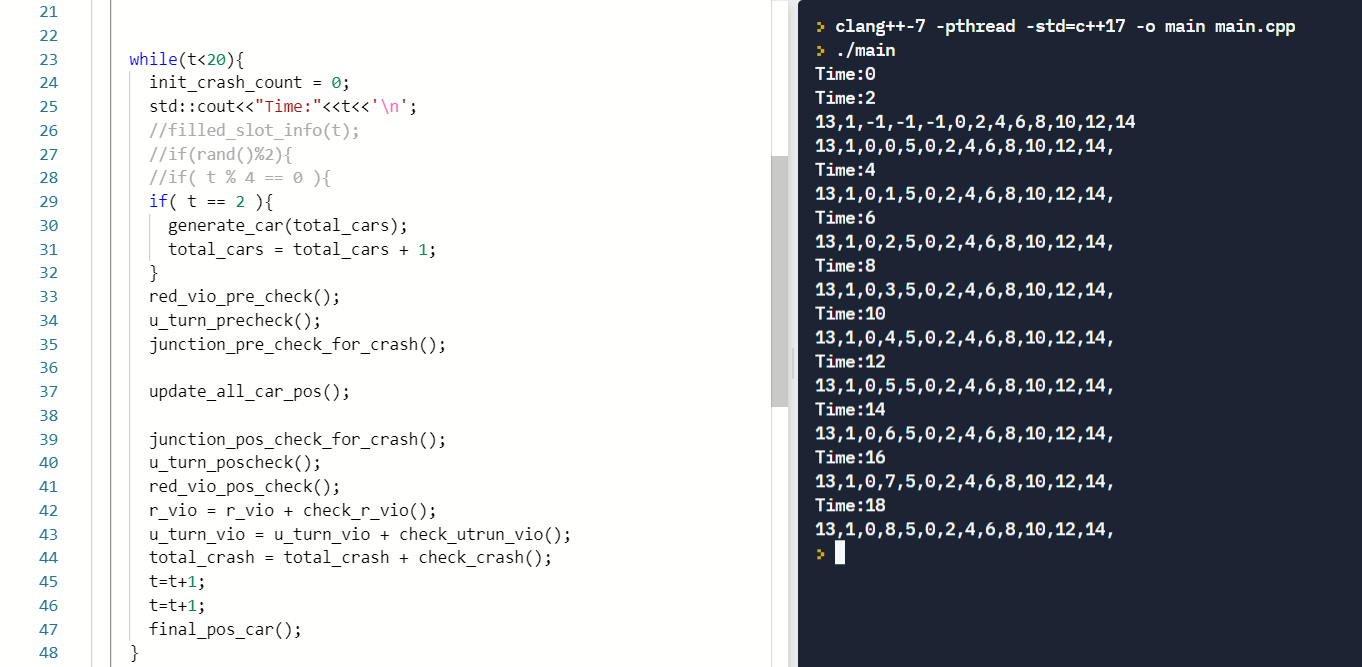


The function check\_r\_vio() returns the number of red signal violations occurred in one time step at the junction. The arrays r\_pre\_ch[] and r\_pos\_ch[] are compared. If the car has moved although the signal is RED at time t = t + 2, then the for that car index the value in r\_pos\_ch[] will become zero. Hence, if r\_pre\_ch[i] is not equal to r\_pos\_ch[i], then it is interpreted as red signal violation by a car. The zeros are excluded from the check as the zero in r\_pre\_ch[] indicates that there is no car at the junction at time = t.

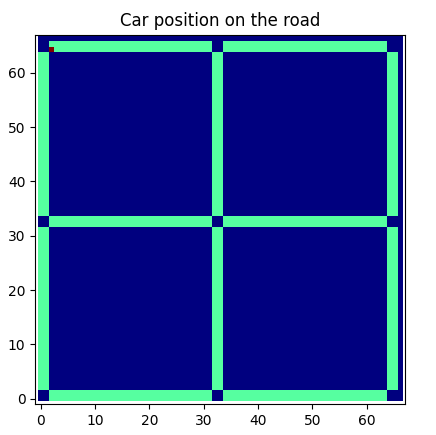
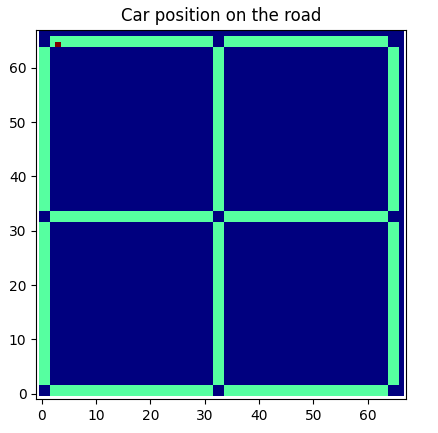
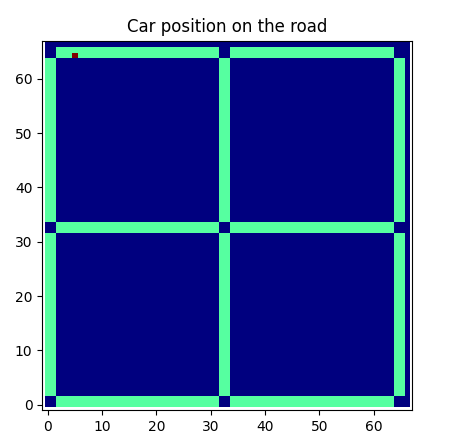
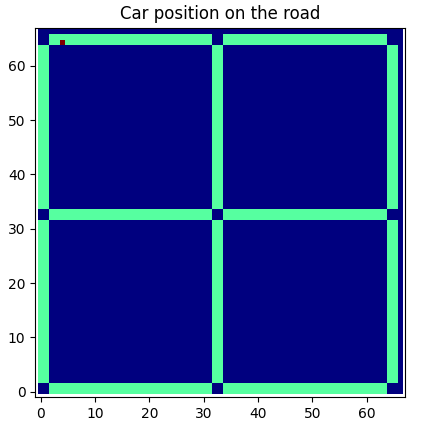
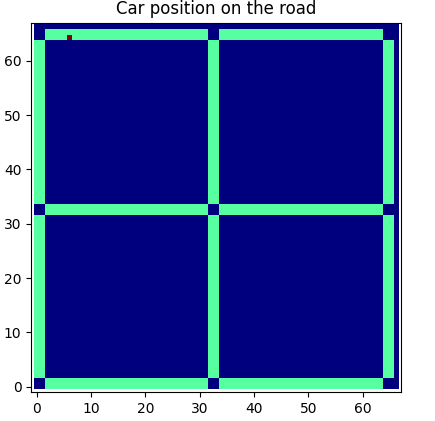
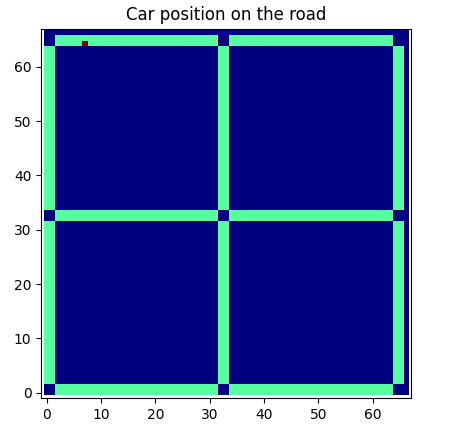
**TEST CASES:**

1. A car can run
2. A car is present in a slot and its position is updated after time increment.

Below is the output for above test case 1 and test case 2:



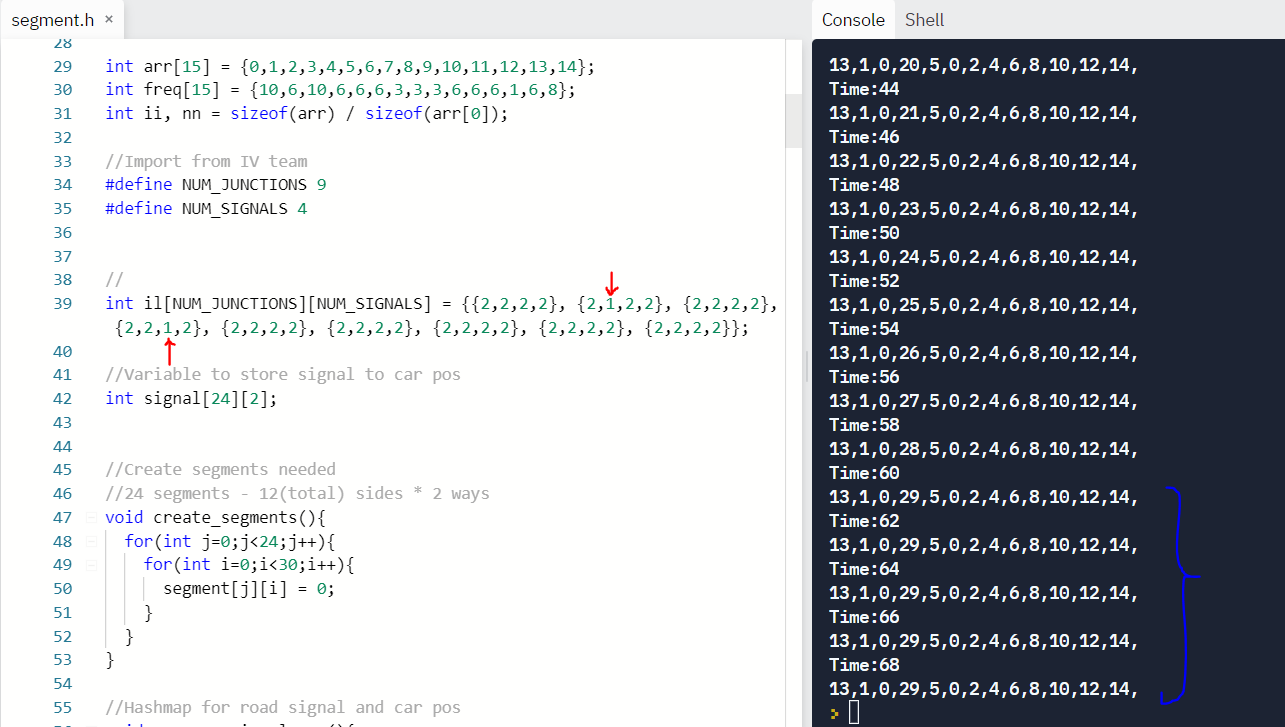
At time t = 2, car with ID 1 is generated. For each time step increment of 2 seconds, the car I smovinf forward in segment 0. This is indicated in the console by index 2 and 3 in printed lines. At time t = 2 , the car is generated and its initial position is -1, -1. At T= 2, the car has moved to first slot in segment 0. This is indicated by position 0,0. At time t = 4, car’s position is 0, 1 i.e. segment zero slot one.

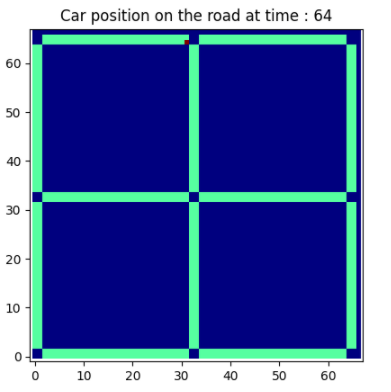
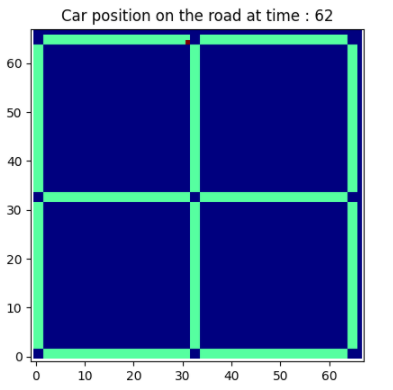
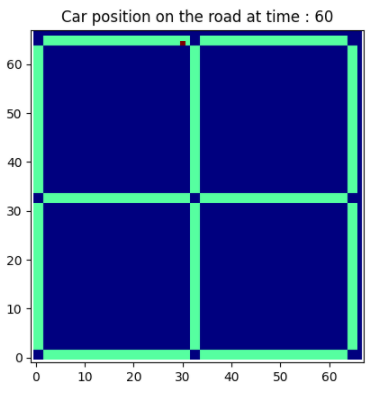
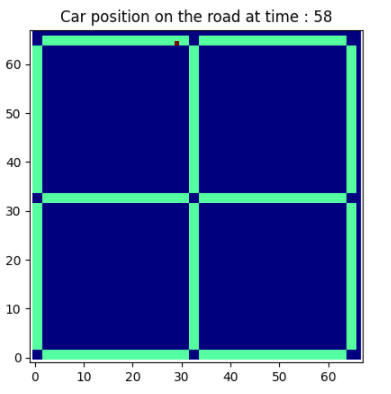
Above is the visual representation of the the console output. In the above images the red dot indicates the car’s position. For the first row, time increments from left to right. The first plot indicate the car’s position at time t = 0. The first picture in second row indicates car’s position at time t = 8 and time increments from left to right.

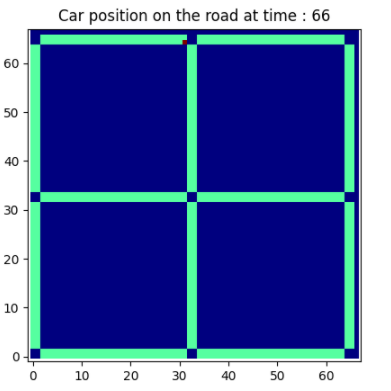
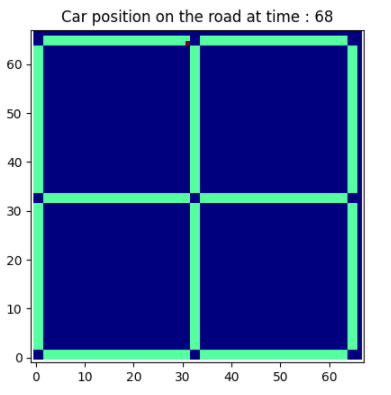
1. Car can stop
2. Car stop at incremental time when signal is RED

Below is the output for above test case 1 and test case 2:



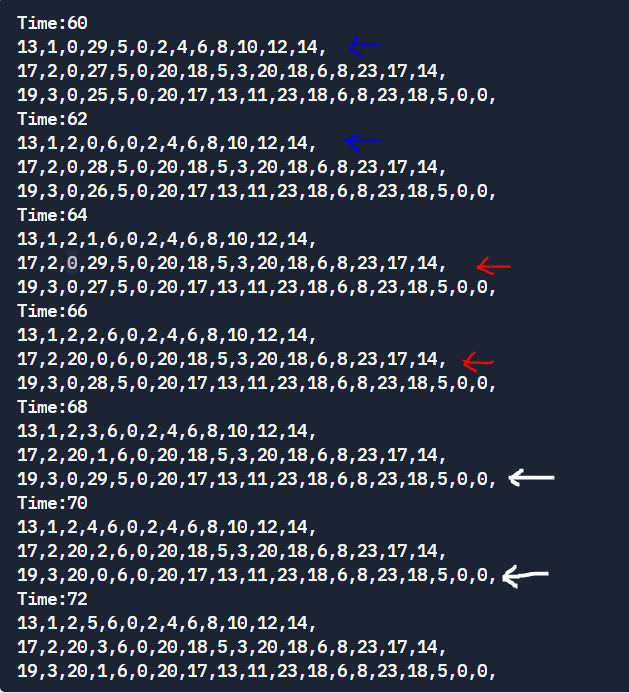
The signal 1 at juction 1 and signal 2 at juction 3 are set as RED. Refer architecture diagram for naming conventions. At time t = 44, car starts moving in next slots. At time t = 60, it reaches at the juction i.e. last slot of the segment 0 but since signal is RED, it won’t be able move to next segment. As indicated in the console output the car position is not updated at time t = 62 and onwards.

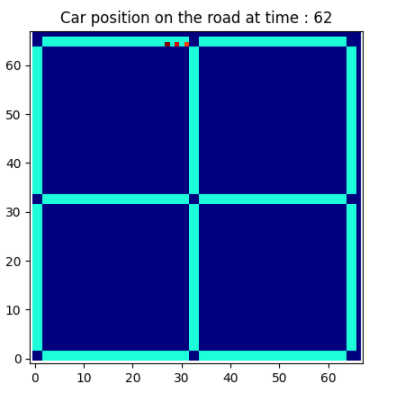
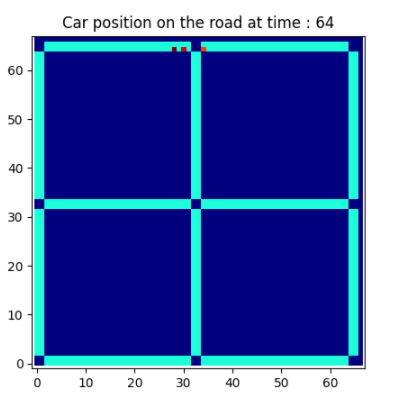


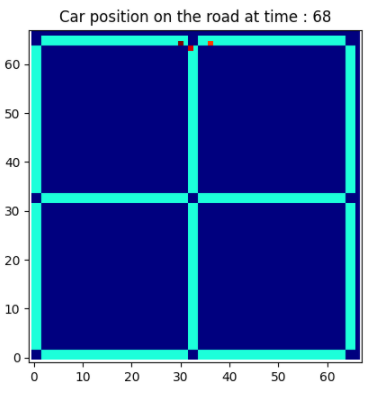
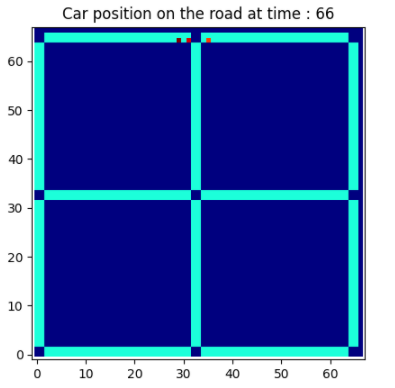
 

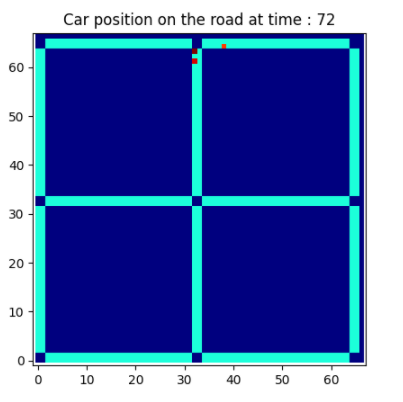
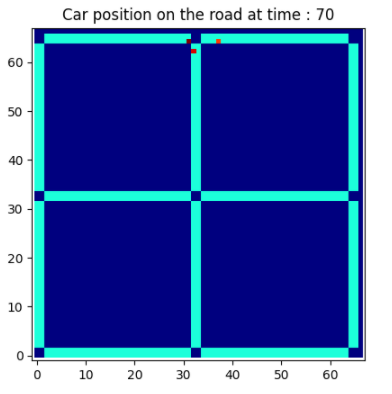
1. Car move straight at juction
2. Car can turn left at juction
3. Car can turn right at juction

Below is the output for above test case 5, test case 6, and test case 7:

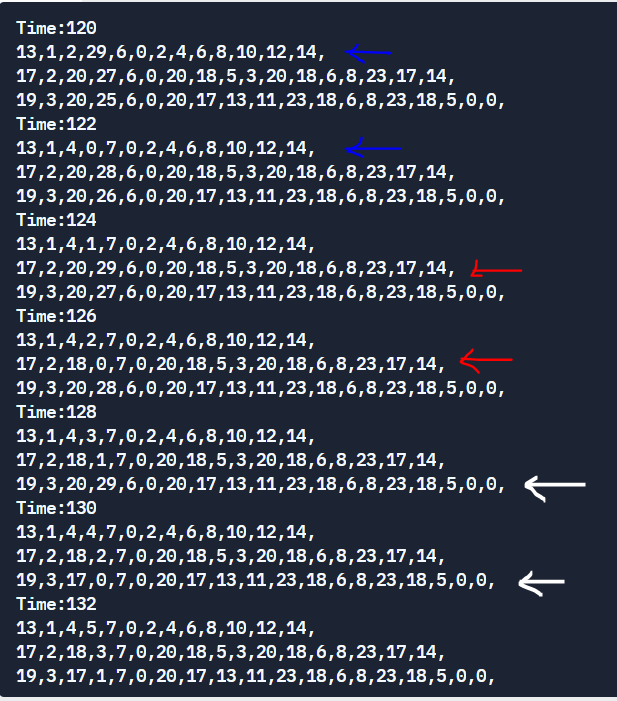


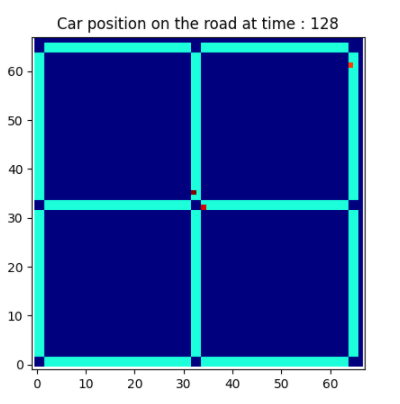
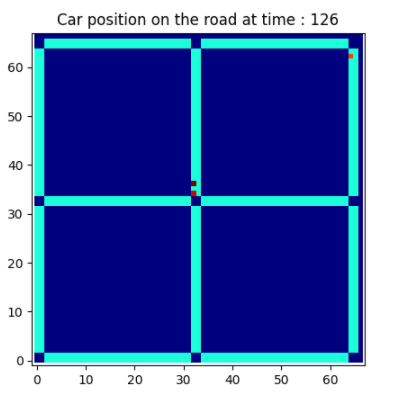
 





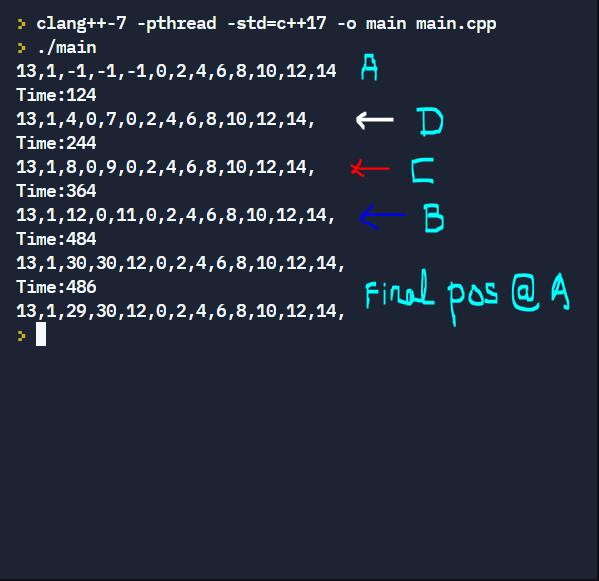
At time t = 62,the car 1 has reached at the junction. At t = 4, car 1 has moved straight to next segment. This indicated by blue arrows in console output. Similarly, at t = 68, car 2 has turned right and this is indicated by red arrows. At t = 72, car 3 has turned right, indicated by white arrows in console.

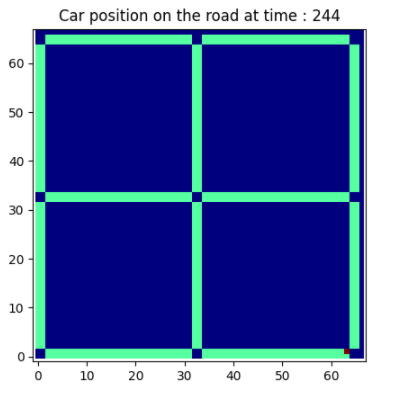
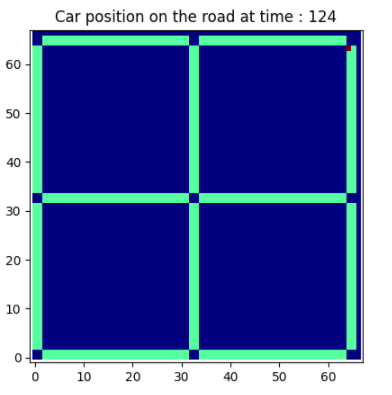


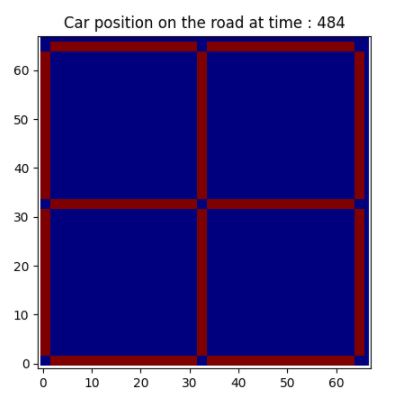
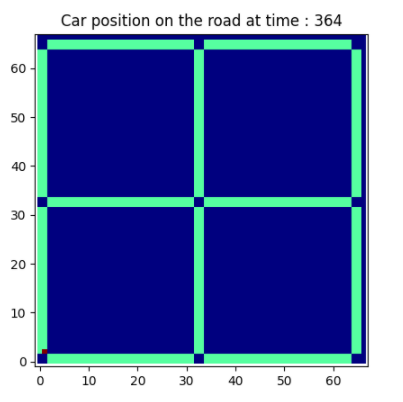


At time t = 126, the car 2 has reached at the junction. At t = 128, car 1 has turned left. This indicated by white arrows in console output.

1. Verify the car is goes to each node A, B, C and D and returns







The car 1 has started from node A, then visits node D at t = 124, then visits node C at t = 244 and node B at t = 364. After visiting all the nodes, the car returns again to A at t = 484. In the last plot, there is no car as car has returned.