CONFLICT-FREE ROUTE PLANNING
FINAL REPORT

151220134002 Mert Batuhan AKGÖK
151220142007 Hasan ALTINTAŞ
151220142048 Cihan KARAÇAY
308520189001 Abdullah YALÇIN

Problem:

Conflicts in route for AGVs which are used in industrial zones.

Project Goal:

- Defining a route for each AGV.
- AGVs stay in pre-specified route.
- Conflicts are avoided.
- Defined routes will be efficient in case of performance measure

Constraints:

- Static Environment
- Every AGV will have the same properties
- Layout of the routes will be a grid (10x10)
- 3 AGVs will be present
- Every unit path will be the same length
- Some AGVs have a priority

Algorithms:

- A*
- Dijkstra

Tools:

- Language: Python
- Simulation: Pygame Library

The situations that they are not regarded:

Speed of all AGVs are static or equal.

Roads work for one type of car, we have selected AGV.

On routes, there are no traffic rules/elements like traffic lambs or crosswalk which could change the arrival time.

If there are any traffic rules, they don’t work dynamically.

Performance measure can be selected only for one goal not multi.

A* Search Algorithm:

A* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/ location having the lowest ‘f’, and process that node/location. ‘f’, ‘g’, ‘h’ represents result function, real cost, heuristic cost respectively.

Euclidean distance:

In mathematics, the Euclidean distance is an ordinary straight-line distance between two points in space.

In this algorithm, the distance is between the current location and the goal location is found with the distance formula.

Comparing the algorithms:

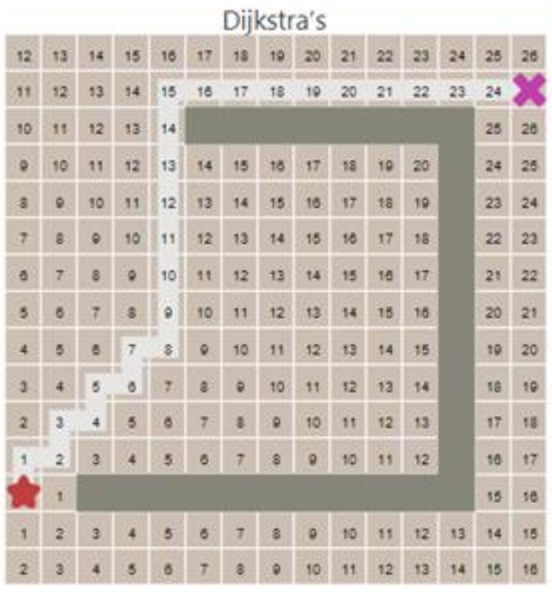


Figure 1-Dijkstra’ Algorithm

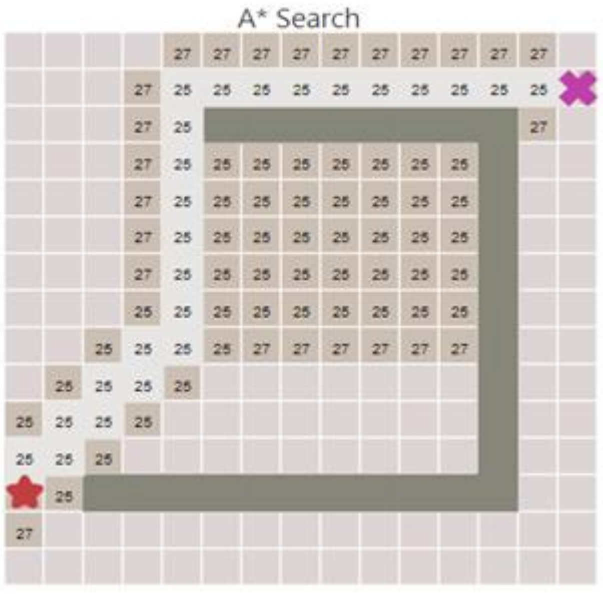


Figure 2-A* Algorithm

Dijkstra’s Algorithm calculates the distance from the start point. A* is using the sum of those two distances.

Project works:

Firstly, the algorithms which are to be used in the project were determined. The algorithms which are more efficient was decided. Two algorithms that are efficient from these algorithms were determined. These algorithms are A* and dijkstra algorithms. These are the most commonly used algorithms for this field. As we have seen in our studies, these algorithms proved to be really applicable to conflict route-planning.

In the beginning of the project, the path of an AGV is determined by using A* algorithm on 3*3 grid area. After that, the size of grid area is increased to 10*10. Then, second AGV is added to the environment and the path of each AGVs are determined . At that point, there were some conflicts occurred. To solve this problem, priority is assigned to each AGV.

First solution was if there is a conflict at a node, the AGV which has less priority is waited in the node before the conflict node on its path. There would be no change for the path of AGV which has a high priority. After that the third AGV is added on 10*10 grid area and different experiments are tested, any conflict is prohibited by using that solution.

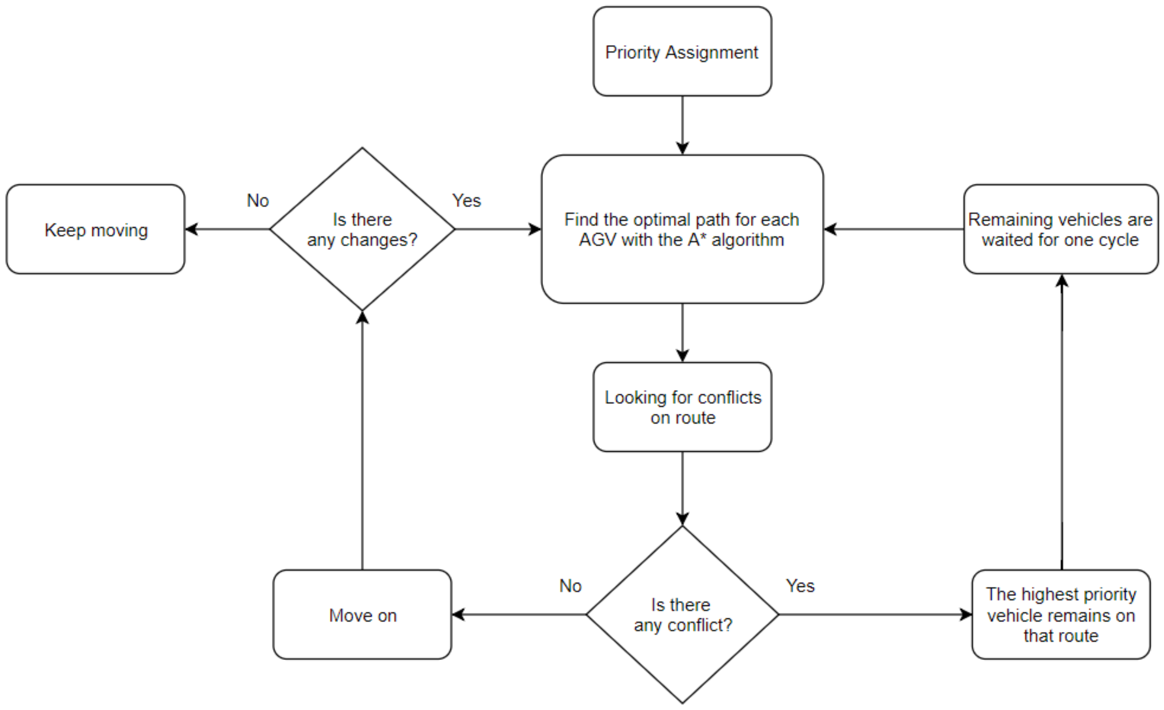

Priority Assignment
Keep moving
No
Is there any changes?
Yes
Find the optimal path for each AGV with the A* algorithm
Remaining vehicles are waited for one cycle
Looking for conflicts on route
Move on
No
Is there any conflict?
Yes
The highest priority vehicle remains on that route

Figure 3- Block Diagram of the Conflict-Free Route Planning for adding a delay case.

The second solution is basically works like, if there is a conflict at a node, the path of AGV which has a less priority is again determined without using that node (like if there is an obstacle in that node) by using A* algorithm. Hereby, two AGVs can move on from desired beginning node to any desired end node without conflict. Moreover, the third AGV is added on 10*10 grid area and different experiments are tested, any conflict is prohibited by using that solution. In addition, AGVs cannot have a same beginning point because any node can be used one time at the same moment by an AGV.

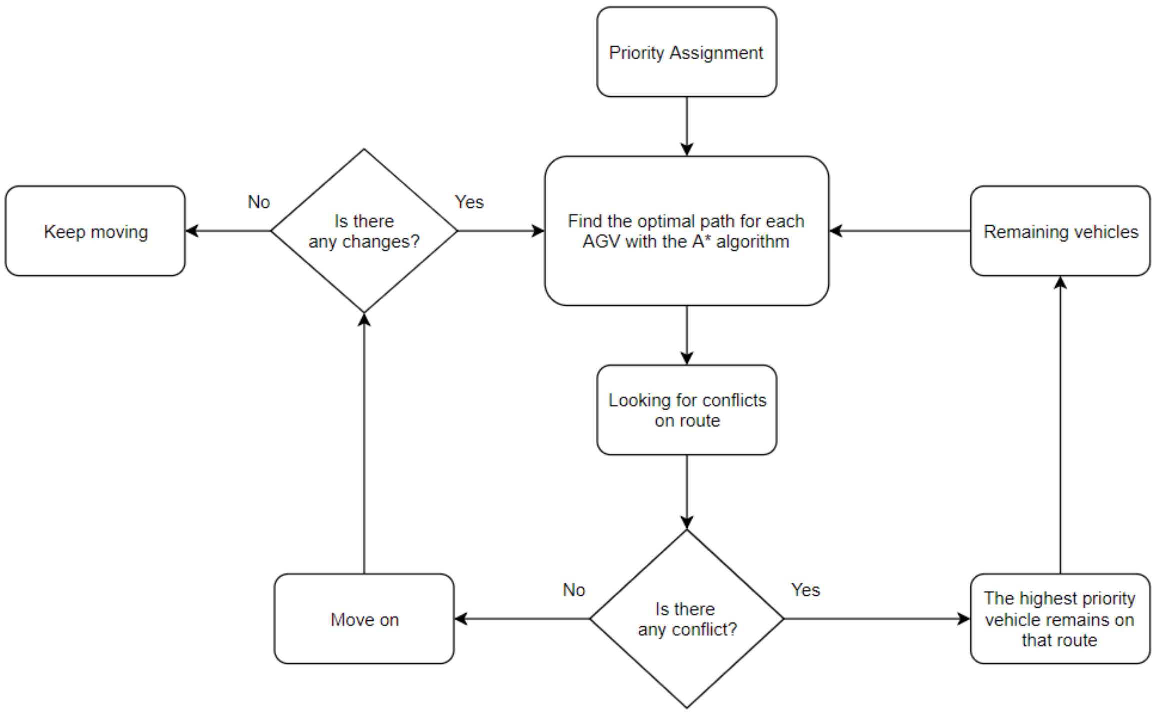
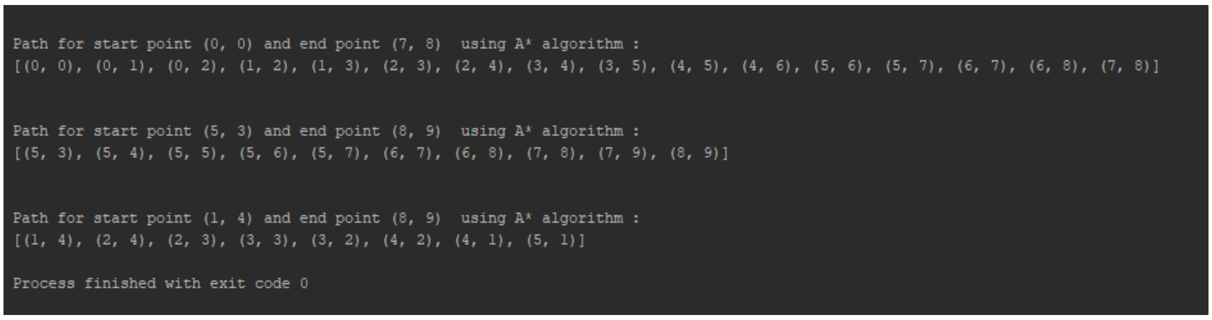

Priority Assignment
Keep moving
No
Is there any changes?
Yes
Find the optimal path for each AGV with the A* algorithm
Remaining vehicles
Looking for conflicts on route
Move on
No
Is there any conflict?
Yes
The highest priority vehicle remains on that route

Figure 4- Block Diagram of the Conflict-Free Route Planning for replanning case.

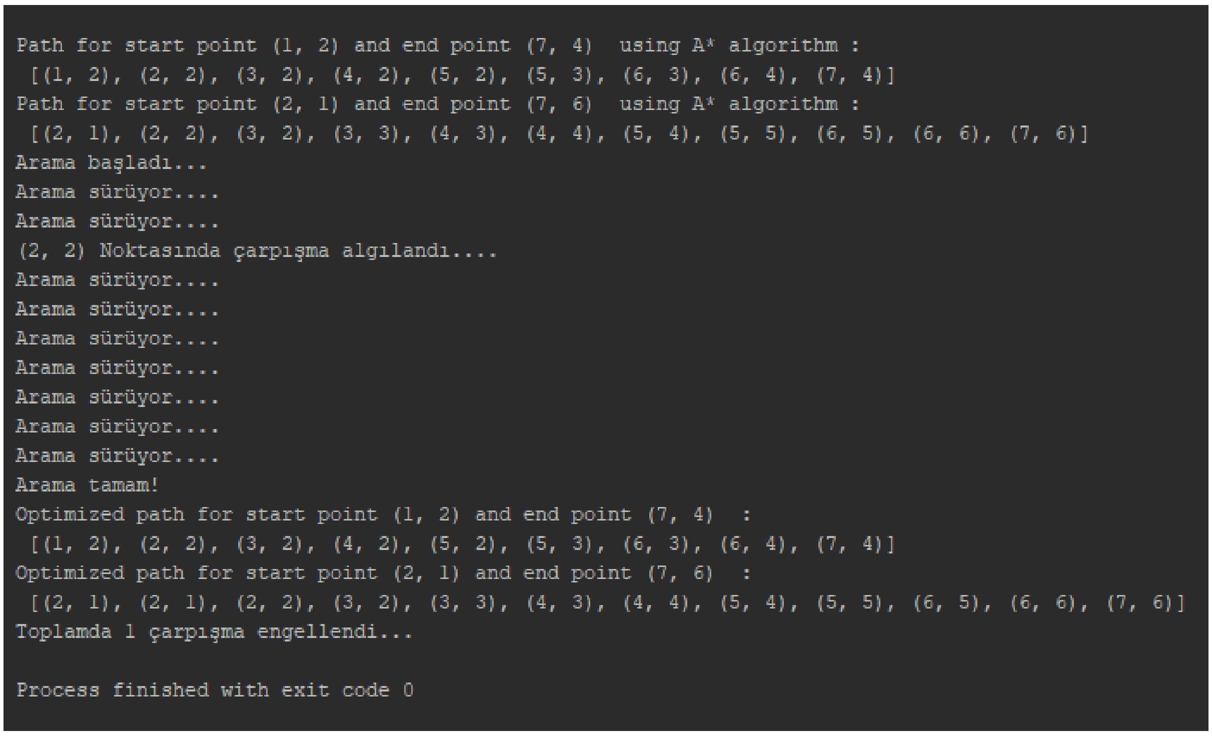
To conclude, there are two solution ways that are tested to solve this problem. Both solutions are successfully solved the problem. The first solution increased the time with adding a delay node on the less priority AGV’s route. Nevertheless, the second solution is finding a new path without losing time. At the beginning, second solution could be seen that is more optimized than first one. However, According to the results, it can be clearly said that first solution is solved more optimized.

Results:



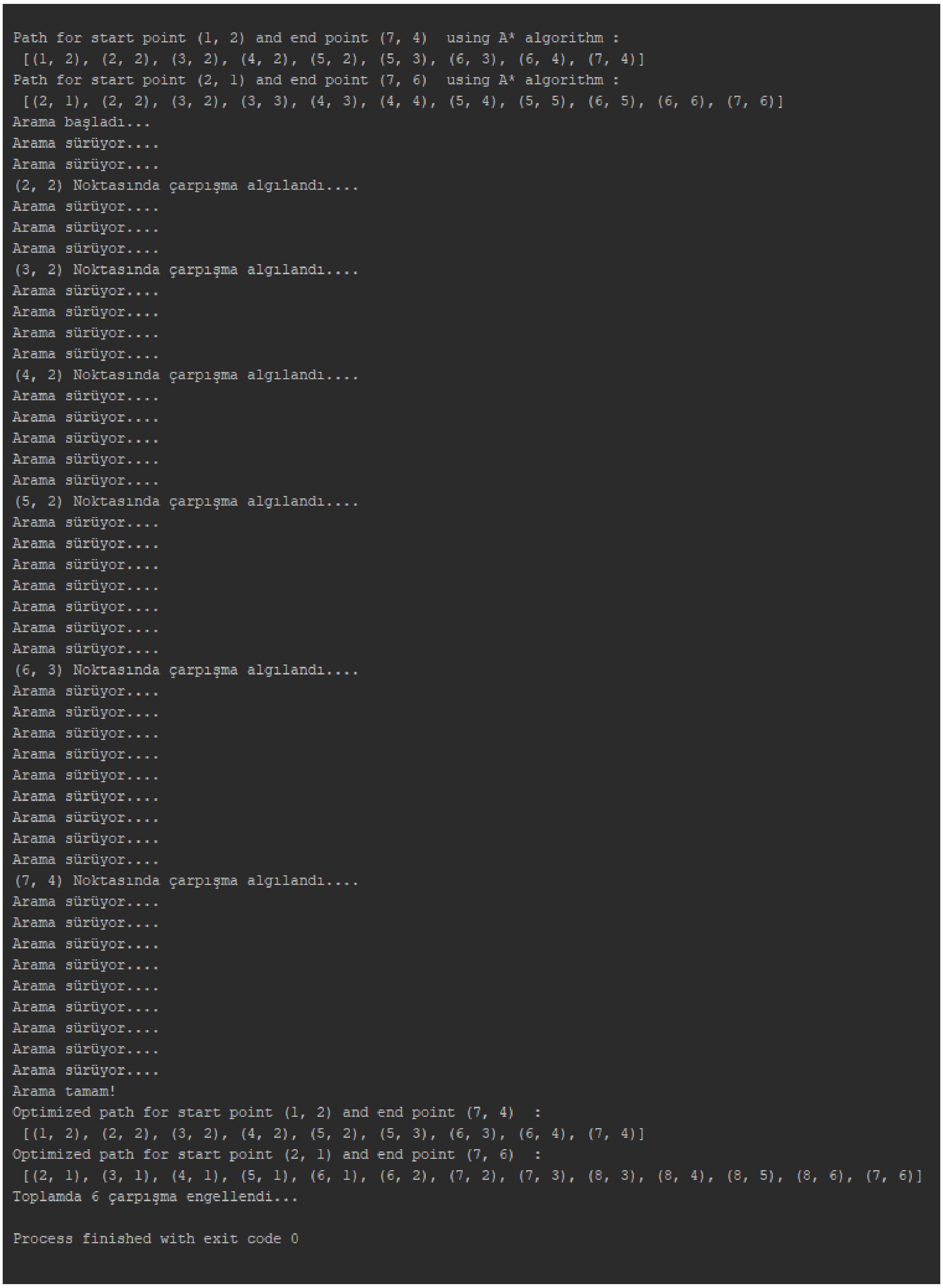
```
Path for start point (0, 0) and end point (7, 8)  using A* algorithm :
[(0, 0), (0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8)]


Path for start point (5, 3) and end point (8, 9)  using A* algorithm :
[(5, 3), (5, 4), (5, 5), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (7, 9), (8, 9)]


Path for start point (1, 4) and end point (8, 9)  using A* algorithm :
[(1, 4), (2, 4), (2, 3), (3, 3), (3, 2), (4, 2), (4, 1), (5, 1)]

Process finished with exit code 0
```

Figure 5.1- A* algorithm with three different start and ending points.



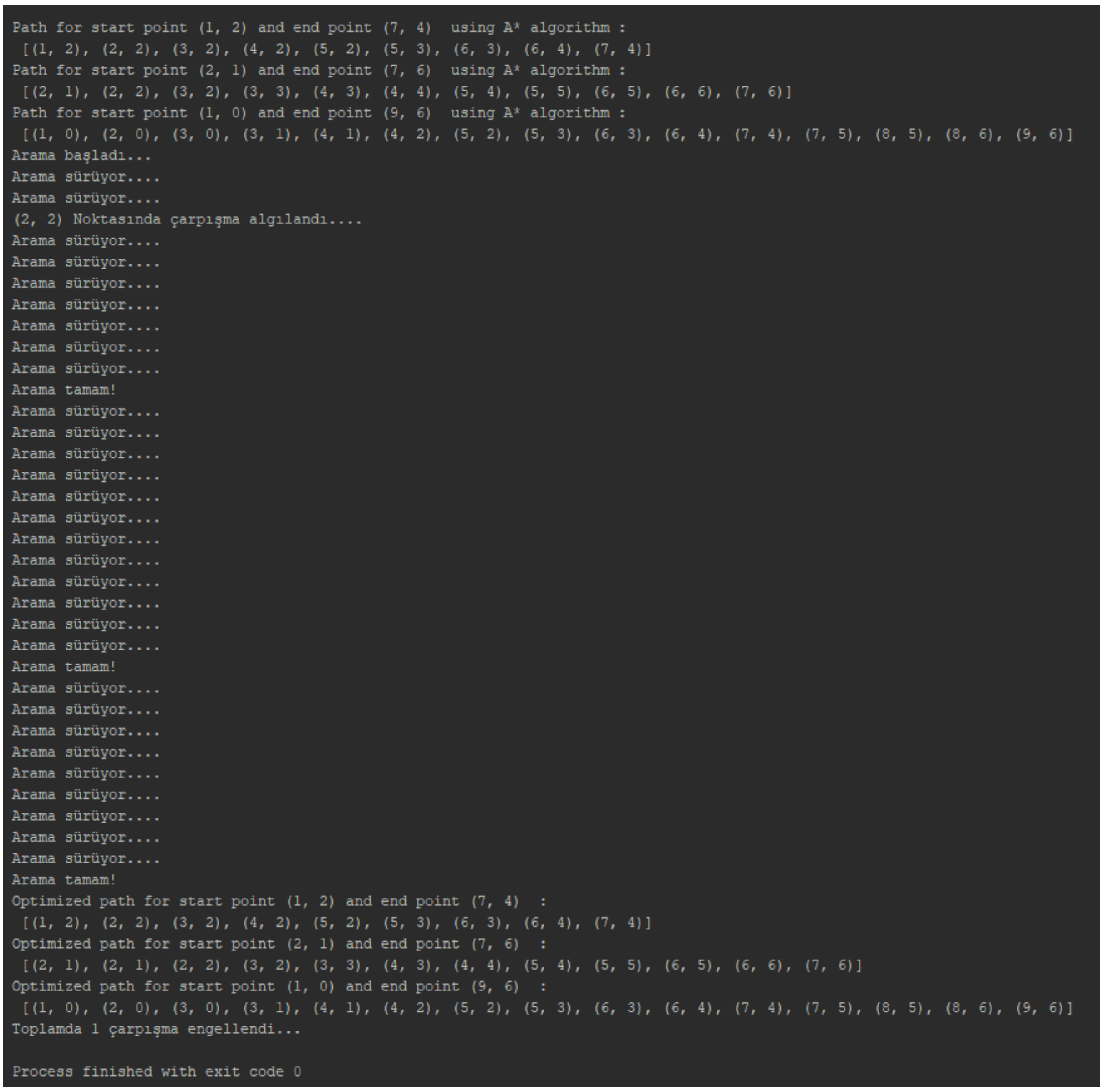
```
Path for start point (1, 2) and end point (7, 4)  using A* algorithm :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Path for start point (2, 1) and end point (7, 6)  using A* algorithm :
 [(2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Arama başladı...
Arama sürüyor....
Arama sürüyor....
(2, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Optimized path for start point (1, 2) and end point (7, 4)   :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Optimized path for start point (2, 1) and end point (7, 6)   :
 [(2, 1), (2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Toplamda 1 çarpışma engellendi...

Process finished with exit code 0
```

Figure 5.2a - 2 AGV route planning results with delay on collision method.



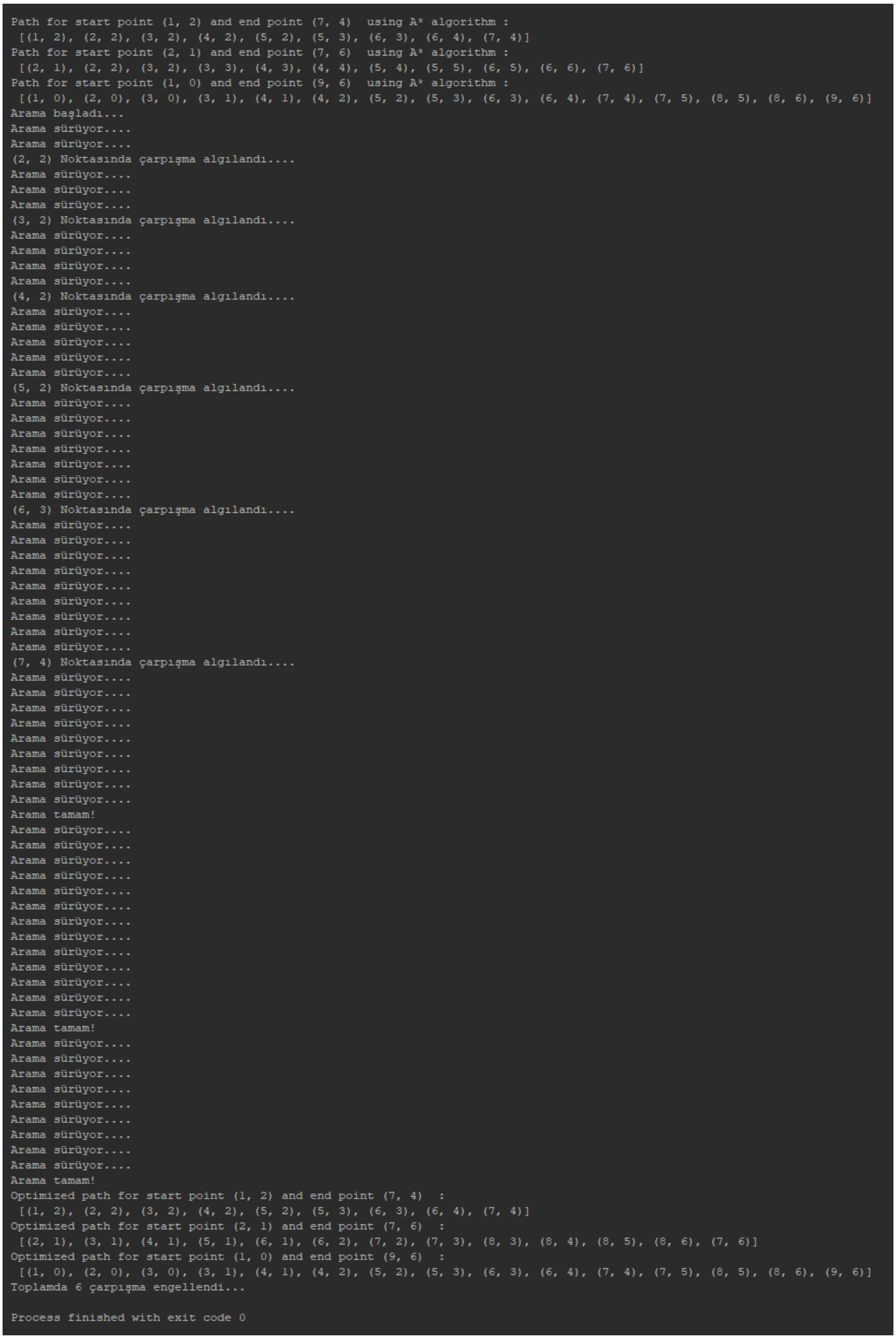
```
Path for start point (1, 2) and end point (7, 4)  using A* algorithm :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Path for start point (2, 1) and end point (7, 6)  using A* algorithm :
 [(2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Arama başladı...
Arama sürüyor....
Arama sürüyor....
(2, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(3, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(4, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(5, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(6, 3) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(7, 4) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Optimized path for start point (1, 2) and end point (7, 4)  :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Optimized path for start point (2, 1) and end point (7, 6)  :
 [(2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (7, 2), (7, 3), (8, 3), (8, 4), (8, 5), (8, 6), (7, 6)]
Toplamda 6 çarpışma engellendi...

Process finished with exit code 0
```

Figure 5.2b - 2 AGV route planning result with re-planning the route after collision node.



```
Path for start point (1, 2) and end point (7, 4)  using A* algorithm :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Path for start point (2, 1) and end point (7, 6)  using A* algorithm :
 [(2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Path for start point (1, 0) and end point (9, 6)  using A* algorithm :
 [(1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9, 6)]
Arama başladı...
Arama sürüyor....
Arama sürüyor....
(2, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Optimized path for start point (1, 2) and end point (7, 4)  :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Optimized path for start point (2, 1) and end point (7, 6)  :
 [(2, 1), (2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Optimized path for start point (1, 0) and end point (9, 6)  :
 [(1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9, 6)]
Toplamda 1 çarpışma engellendi...

Process finished with exit code 0
```

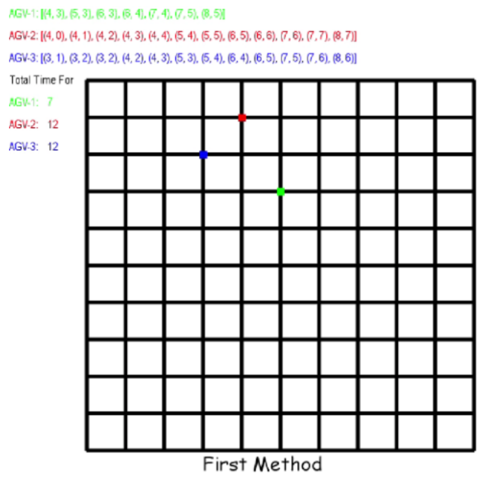
Figure 5.3a - 2 AGV route planning results with delay on collision method.

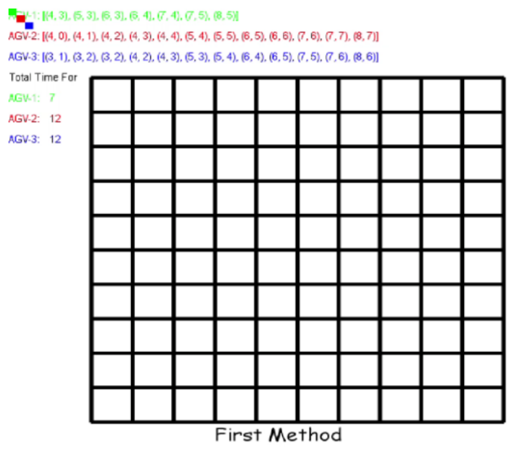


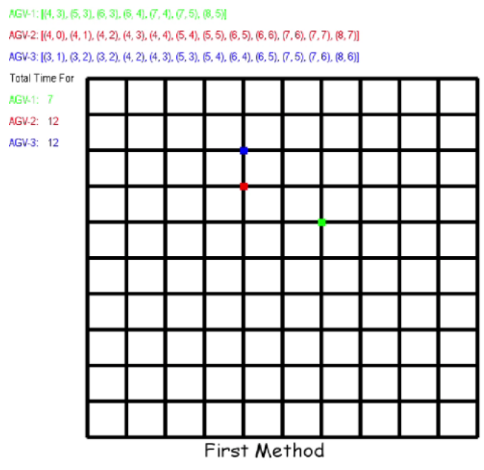
```
Path for start point (1, 2) and end point (7, 4)  using A* algorithm :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Path for start point (2, 1) and end point (7, 6)  using A* algorithm :
 [(2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6)]
Path for start point (1, 0) and end point (9, 6)  using A* algorithm :
 [(1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9, 6)]
Arama başladı...
Arama sürüyor....
Arama sürüyor....
(2, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(3, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(4, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(5, 2) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(6, 3) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
(7, 4) Noktasında çarpışma algılandı....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama sürüyor....
Arama tamam!
Optimized path for start point (1, 2) and end point (7, 4)  :
 [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4)]
Optimized path for start point (2, 1) and end point (7, 6)  :
 [(2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (7, 2), (7, 3), (8, 3), (8, 4), (8, 5), (8, 6), (7, 6)]
Optimized path for start point (1, 0) and end point (9, 6)  :
 [(1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (4, 2), (5, 2), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5), (8, 6), (9, 6)]
Toplamda 6 çarpışma engellendi...

Process finished with exit code 0
```

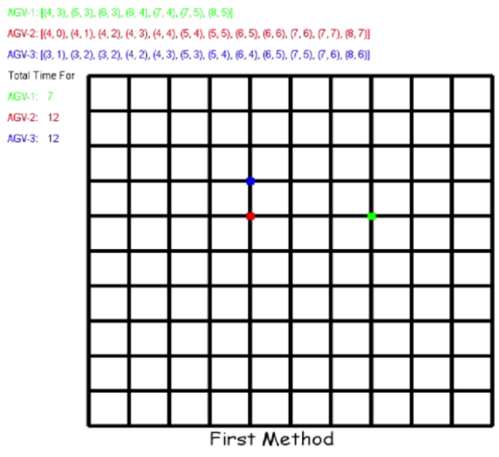
Figure 5.3b - 3 AGV route planning results with re-planning the route after collision node.

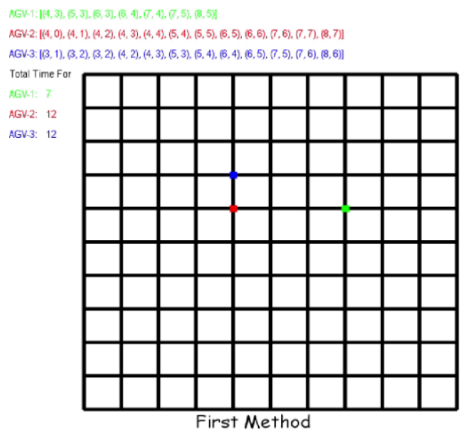
First Method:

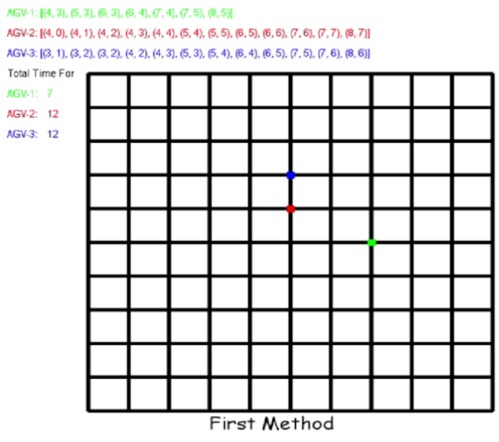

AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12

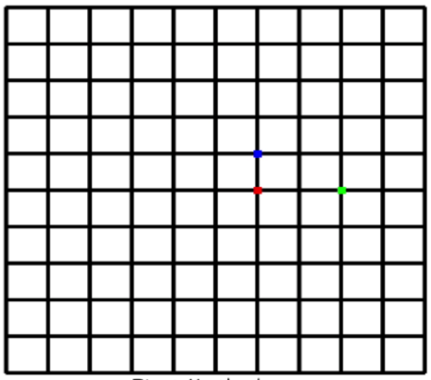

AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12

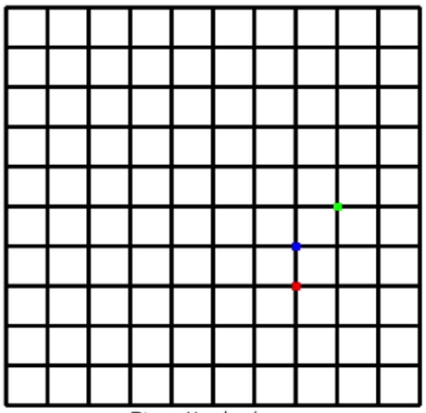
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


First Method

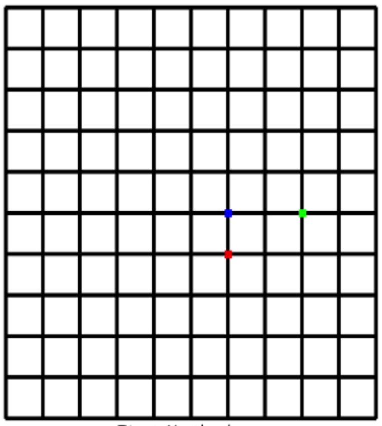
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


First Method

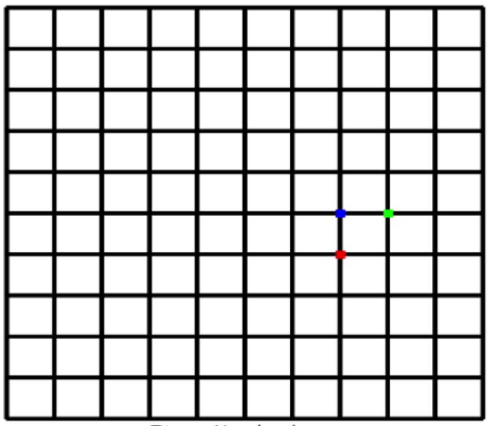
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


First Method

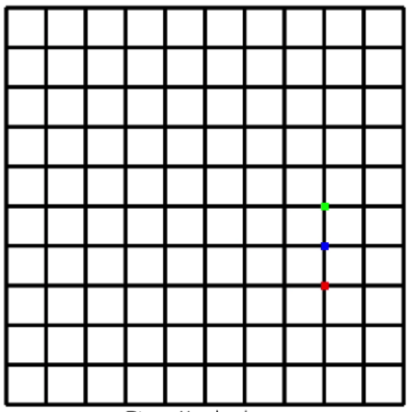
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


First Method

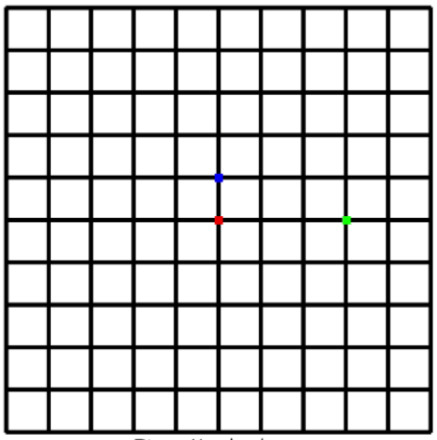
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12

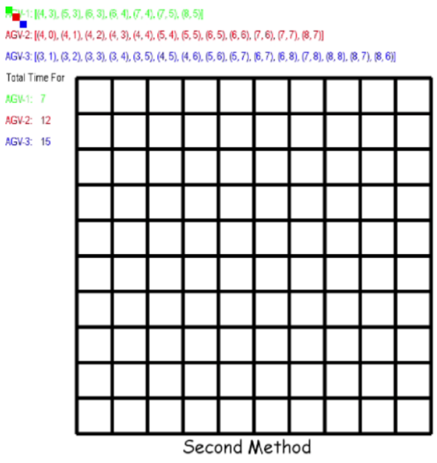

First Method

AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6)]

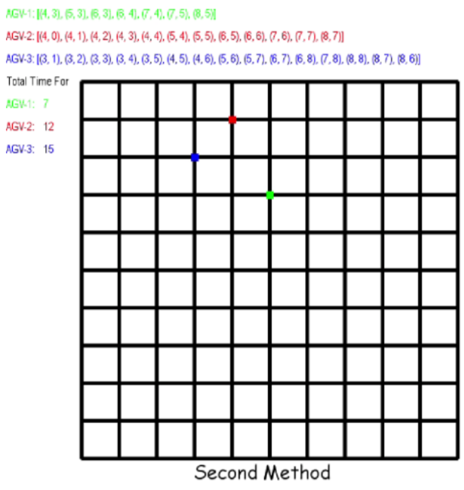
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 12


First Method

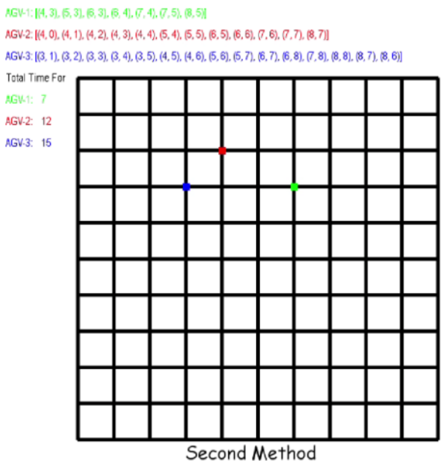
Second Method:



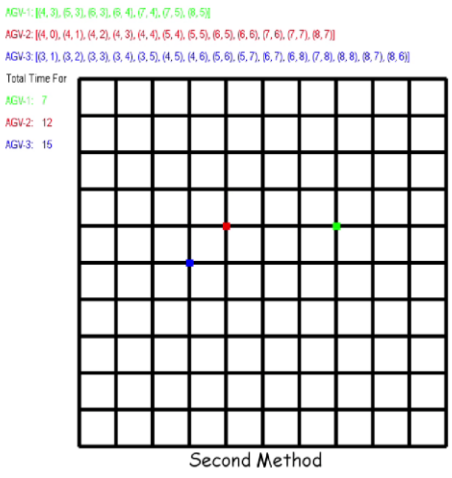
V-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



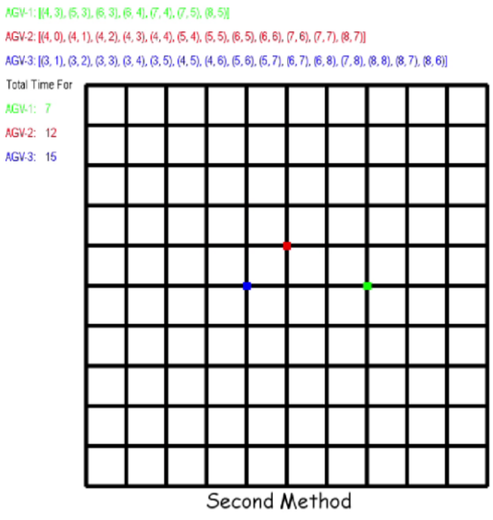
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



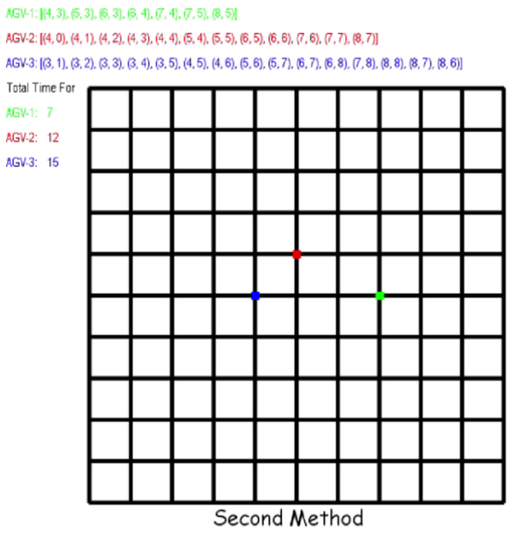
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



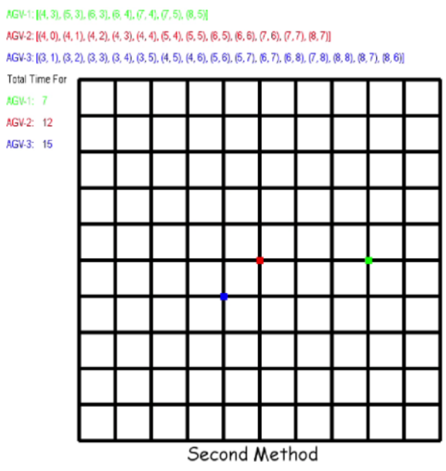
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



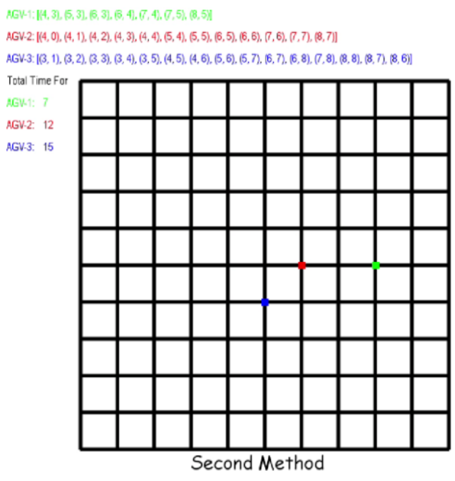
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



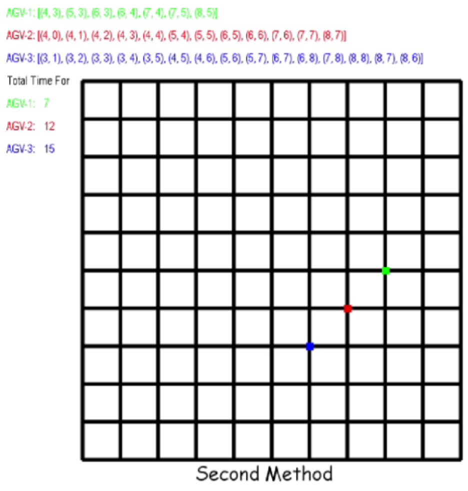
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1: 7
AGV-2: 12
AGV-3: 15
Second Method



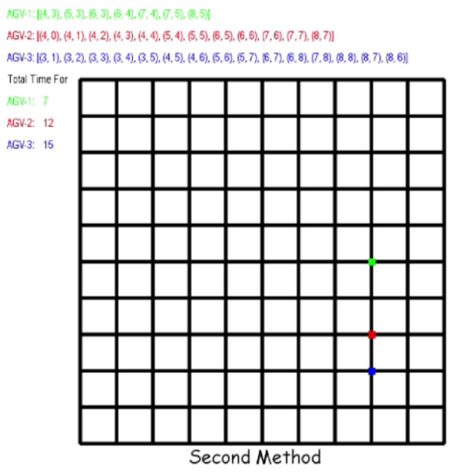
```
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1:  7
AGV-2:  12
AGV-3:  15
```



```
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1:  7
AGV-2:  12
AGV-3:  15
```



```
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1:  7
AGV-2:  12
AGV-3:  15
```



```
AGV-1: [(4, 3), (5, 3), (6, 3), (6, 4), (7, 4), (7, 5), (8, 5)]
AGV-2: [(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7)]
AGV-3: [(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (8, 8), (8, 7), (8, 6)]
Total Time For
AGV-1:  7
AGV-2:  12
AGV-3:  15
```

References:

https://www.pygame.org/docs/

https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

http://mnemstudio.org/path-finding-a-star.htm

https://en.wikipedia.org/wiki/A*_search_algorithm

https://www.geeksforgeeks.org/a-search-algorithm/

https://www.redblobgames.com/pathfinding/a-star/introduction.html

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://brilliant.org/wiki/dijkstras-short-path-finder/

https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e