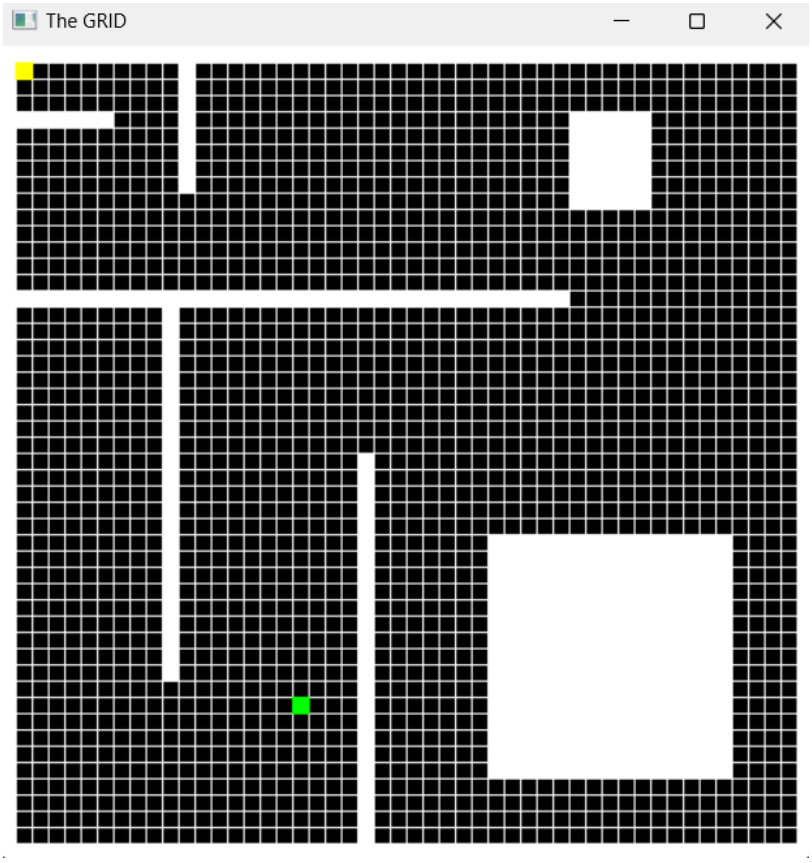
AGV Task 1 Documentation

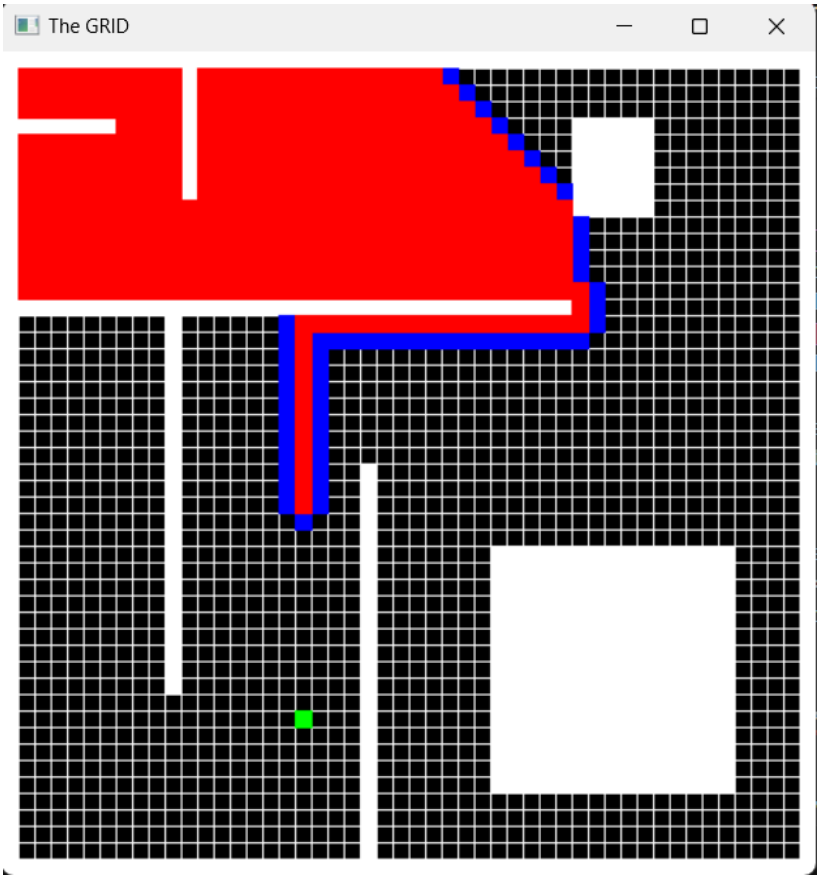
Step 1: Apply A* to a single agent

First thing to do was to make a grid environment. The function named GridMaker() gets it done. It draws white lines on a black numpy array image. Then the obstacles are defined as white boxes. Then start and end points were randomly chosen. The heuristic employed is the Manhattan distance. The green node represents the goal and the yellow node represents the start node.



The algorithm moves forward by calculating the f score by taking the sum of the g score and the h score. The g score is defined as the cost incurred to reach that node from the start node. The h score is the estimated cost to reach the goal node from the current node. The g score is calculated as the sum of the g score of its parent node plus the cost to travel from the parent node to the current node.

The neighboring node of the current node with the least f score is taken as the next node. The previous node is said to be explored and the neighbors are said to be visited. The explored nodes are represented in red while the visited nodes are represented in blue.



I used a dictionary to keep track of the parent child relationships in the grid. The key of any dictionary element is the parent of the node present in the value position of that same element. When the algorithm reaches the goal, it uses this dictionary to trace the path back to the start.

Step 2: Apply A* to n agents

To apply the algorithm mentioned in step 1 to n number of agents, just put all the steps into a loop that executes for as many times as there are agents. In my program there are 3 agents. The algorithm finds paths for these three agents individually. Then on the higher level, it searches for conflicts between the agents in their paths. If there are no conflicts then the program ends, otherwise the

algorithm makes one of the agents involved in the conflict to take a back step when it reaches the point of conflict. The list named CT stores all the conflicts. The conflicts are found out by keeping track of another parameter that is similar to time. So at time t there must be one or less than one agent in any given node. If not, there exists a conflict. The final paths are shown as below.