AGV SOFTWARE TASK

Task-1:

"*The research paper laid some foundation to me about path finding algorithms*"

In the paper it is explained that the problem of Multi Agent Path Finding is solved using the Conflict Based Search Algorithm. The CBS Algorithm works on two levels, The high level search that solves the conflicts by a Conflict Tree and a low level search that finds the paths to agents by a search algorithm such as A*.

Part-1: Single Agent Low Level Implementation

In my implementation of A*, I have used the python libraries pyamaze and priorityQueue. The algorithm creates maze with many possible paths between starting and ending point. The start and end points can be chosen and the size of maze can also be chosen. I've set the start and end points to random in a maze of size 30x30. The goal position will be shown in green color and the agent's path can be seen as it follows the path it was given after solving the A* algorithm between the two points. The cost function is created as f(x) = g(x)+h(x), where g(x) is node to node travel cost and h(x) is the heuristic cost. The priority queue sorts out the node costs by the priority of total cost; in case total cost is same then the heuristic cost is taken. I have used the Manhattan Distance as the heuristic function as it was a maze (rectangular in shape).

Part-2: Multi Agent CBS Implementation

The Conflict Based Search Algorithm works for mutli agent path finding by using two levels of implementation. In the low level implementation it finds the paths of individual agents by a search algorithm such as A* and in the high level implementation it uses a constraint tree to find the conflicts between the paths of agents (conflicts are nothing but collisions that occur because of two or more agents being at the same location at the same time). The constraint tree check the conflicts between two paths and puts constraints on those agents by creating child nodes and it continues this process until it finds a node where there are two non conflicted paths are found with the least cost.

For this implementation I have used the libraries pygame for grid implementation and agent visualisation in the grid environment. heapq library for priority queue.

Although there is quite a lot of visual details left to work on, due to time constraints I would continue it again later.

THANK YOU FOR THE TASK