The Problem:

We chose problem xyz. The aim of this was to get a robot from point A to a target on a grid. The grid was made of 1 by 1 squares and was meant to replicate a department store. The Robot is to hace a diameter of 1.6 squares and there will be obstacles. The robot can move between 1 and 3 meters in a single direction or turn. The robot can only turn to the cardinal directions and can only move straight forward.

Our solutions:

We planned to do 3 solutions and have attempted all solutions. We had issues with finishing some due to issues for all members as one had to self-isolate one had to visit the hospital for another issue and then time management was poor as a group. Despite this we have got progress towards all solutions.

Grid Transformation:

WE figured we would transform the grid from the one provided in a method. This would allow us to treat 4 blocks as a singular block which would allow us to simplify the problem before tackling it. We did this by using a 2 by 2 window sliding algorithm. This meant that if a block of 4 had a 1 in it then the robot could not occupy that area meaning on the new grid the combination of those 4 squares would be a 1 to prevent the path using that area. When the grid has been transformed, we are able to treat the robot as a 1 by 1 square instead making the problem simpler to solve. This is used for all our solutions.

Solution 1:

Our first solution was to attempt a brute force method that would traverse the grid using the breath first search method. Prior to this we transform the grid that is passed. There is no checks to make this efficient it just find the first possible path, we would improve this to find all paths then return the shortest path out of possible paths however then this would be more than a brute force solution. We do have other solutions and ideas for how to improve off this base.

Solution 2:

Our first improved method was to use Dijkstra's Algorithm. We chose Dijkstra's as it is one of the best path finding algorithms, the idea behind it is that you add up the length of the path as you go and chose the next shortest path each time while adding up each possible path storing them as you go. At the end of the search you then backwards traverse the tree and find the smallest value end path and find the path from there.

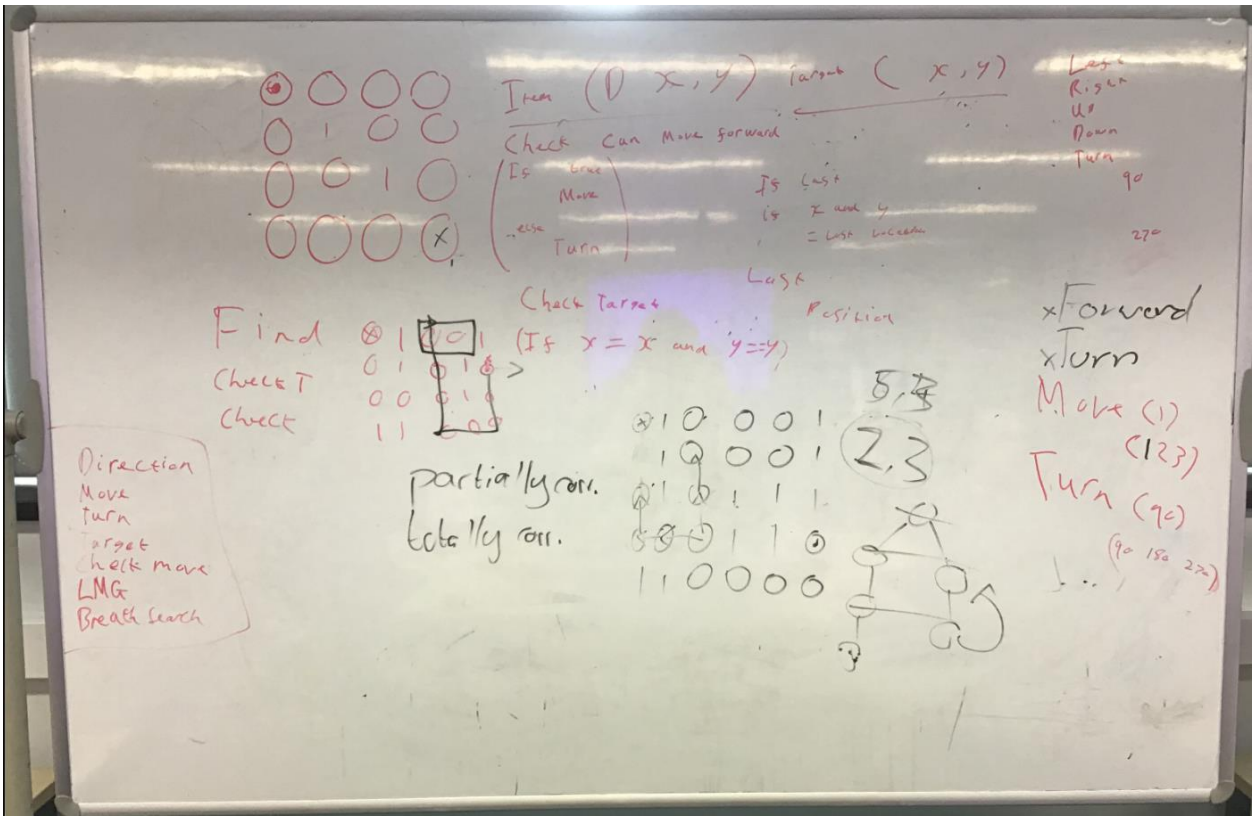
We started this solution when we thought we had a working brute force solution. This path finding is a far better method than using brute force as it will find the optimal path and give the locations to move to. Despite this not being a fully working solution we had some base for it built however we had to go back to fixing an issue we found in the brute force solution while doing more testing on it.

Solution 3:

Our second improved method was to use Dijkstra's Algorithm. This is like the Dijkstra's algorithm with one key difference, that direction is taken into account, meaning A* ignores paths that move you further away from the target destination. When it comes to the actual code this means adding in something to check the x and y values are always getting closer to the target location despite it being a longer route. This could cause issues if this leads the robot into a dead end as it would then go against the optimal path. We did look into this and have some design written up however we did not make much progress on this. We started some stuff when we started Dijkstra's as they could both draw from the same methods however as soon as we found issues in the brut force method we had to divert attention back to that solution again resulting in this being bottom of our priorities.

Algorithms and Notes:

Brute force:



Return a map of nodes from point starting point to target point with length required to traverse.

Path

Loop

For (locations (location))

+ Path-length 1

Dijkstra:

For each path,

Compare path-length

If path-length < shortest

Shortest = path length

Return shortest

Locations(Direction, x, y)

Return list

+ North (0,1)

+ South (0,-1)

+ East (1,0)

+ West (-1,0)

A*:

For A* method some of these functions can be used again due to the similarities of the two algorithms,

Direction

If target x + current x) > (target x + new x)

Return false

Else

Return true

A*-path

Loop

For (locations (location))

If (((location x + target x) < (current x + target x) || (location y + target y) < (current y + target y))

+ Path-length 1

If turned

+ path-length 1

For each A*-path,

Compare path-length

If path-length < shortest

Shortest = path length

Return shortest

Peer points:

Patryk 3.5

Dom C 3.5

Dom K 3.5

Ben 2.5