Korakot Santiudommongkol

CSC 412 Final Project Report

Data Structure used to hold the data

I created two different structs one to hold the information on the robot+box pairing, and the other to hold information on the door. The door struct only consists of its coordinate: int row and int col, and the index of the door or door number. As for the robot+box pair struct I called it roboxInfo, it contain information on coordinates of the robot and box, the assigned door that the box must reached, the number of row and col the box must be pushed to reach its assigned door, the Direction the robot is moving or pushing, and the status of the robot whether it is moving, pushing, or end.

RobotsV1

I stored the robot+box struct as a global vector and the door information as a global vector. They need to be global because that's how the robot,box, and door is going to be displayed on the GUI. The other global variable I had implemented was the file so that the functions that needed to write into that particular file does not need to take any argument since they are writing into the same file. I also had a char array to represent the Direction as N,W,S,E. I have two functions that writes into the file, the first one writes all the initial information on robot,box, door. While the function writeRobotexe writes all the commands the robot does, the function only checks for the current status of the robot if its moving,pushing, or end. This is where the char array comes in usually I would have to create nested if or switch statements to check the direction, since the directions are being represented as an enum I can use chararray[direction] to print out the direction. My general algorithm I used to move and push the box is that the way the box needs to be push to the door is just pushing one direction and another direction to reach the door, or if you are lucky you just need to push one direction. So with that in mind I could preplan the path of where the box is pushing beforehand be subtracting box coordinate with door coordinate. So the pushcommand I have is straight forward just checks for the direction it needs to push and push the number amount it needs to push that was calculated in the robot+box pair struct. Pushing the box is the easy part of this algorithm, the moving part took a bit of time to come up with a working solution. For the move command, it cannot be preplan incase the robot has to move through its paired box to get to the correct position to push the box. The way my move algorithm works is that it will move the general direction where it needs to be to push the box by checking the row and col of the robot ,and comparing it with the row and col of its destination. The move function will check to make sure that the box is not in front of the place it needs to move and then used the turn function that will make the bot turn to perpendicular direction, and then move the original direction it needed to. The way the robot moves the box to the door is by creating a

vector move list to store the direction the box needs to be pushed, then loop through the vector performing one move function and one push function each time.

RobotsV2

Same as Version 1 excepted I use a mutex lock to lock access to the file that needs to write in. In both of the write function I used the mutex lock to lock access to file then unlock at the end.

RobotsV3

It is still incomplete, there is a deadlock problem that has to do with my move algorithm, since in the previous version I did not account for the other robots and boxes besides the robot+box pairing, there will be deadlock that occurs. For instance two robot could be in the same row one could moving west while the other moving east, they would collide with one another unable to move elsewhere. For this version I created the lock for each space on the grid, before a robot move I will lock the grid space that is moving to first move, then unlock its previous grid space. If its a robot pushing a box, I would lock the grid space the box is being pushed into first, then unlock the box previous gridspace, then the robot will lock the gridspace its moving, and then unlock its previous one.

Limitation

The limitation is with the algorithm I used to move and push the box, I never accounted for it the robot or push will move outside of the grid because for the fact that the box cannot be placed anywhere on the edge of the grid. The other obvious limitation is the grid size, the row has to be more than or equal to 3 and the col has to be more than or equal to 3 at all times. The last limitation is that there is a deadlock problem in version 3. If there is only 1 robot+box pair in the grid then no deadlock would occur. The more robot+box pair there are the more likely the deadlock problem would occur.

Difficulties

The only difficulties I found in this project is coming up with a solution to move the robot to the right position while taking account of the box that it needs to push is blocking the robot. My initial solution to the problem with the box being in the way is checking if the box is in the way, if it is switch the direction to one that is perpendicular to the current direction. The problem with this solution that the robot might go back to the previous space where the box is blocking its way. My solution to the problem is that if the box is blocking the way move once in the perpendicular direction and then move to the current working direction.