Final Project Report

William Mann & Ayman Sandouk

Problem Statement:

Create a program that would allow the random placement of a predefined number "robots", "boxes", and "doors" in a set grid of nxn tiles in which the robots and boxes could move. After initializing these aspects, determine a methodology to move the robots to "push" the boxes to a predetermined door set to each robot randomly during the initialization phase. Upon the box reaching the door, remove the box and robot from the grid as their job is complete.

Version 1 (v1): The above, with no checking of placement of other robots and boxes.

Version 2 (v2): The above, now with move checking.

Design & Implementation:

The general premise of the initialization phase is to use uniform distributions and a random generator to provide coordinates of a possible placement of each object type. In order to avoid placement of objects on top of other objects, instead of using a for-loop and iterating through the required number of objects, we use a while-loop checking that the number of each object is less than that required. Each iteration a new coordinate pair is generated and checked if the location is taken. If it is not, then we place the object and "fill" the given coordinates. This occurs for doors, then boxes, and finally robots. During the initialization phase of the robots, an extra random number is found for the door it and its box should route to.

After initialization, the threads for the robots are initialized and they are free to roam around the map. Each robot first plans a path to the point next to its box that will allow for it to push the box in the direction of the door. In v1, the robot doesn't check for overlapping robots or boxes aside from its own box. If the robot runs into its own box while attempting to get to the initial push spot, it gets stuck as there is no repathing. In v2, before each movement, the robot locks the grid and checks if the spot is taken. If it isn't, then it moves and releases the grid. If it is, then it releases the grid and waits for it to become free. For both versions, the method of movement is predefined in terms of 8 possible operations: move left, right, up, down; push left, right, up, down. In Extra Credit 1 (ec1), an additional operation is added, repath. During the pathing process, if the robot runs into an object, it will wait for a given amount of time dependent on its own number and check again. If it cannot move for 5 wait periods (attempts), it attempts to repath by moving one block up or down and one block left or right and recalling the pathing function.

Limitations:

In v1, currently, robots often find themselves unable to push their box to their door as they would have to move through their box in order to get to the position to push the box. This could be resolved by adding a couple of operations to move it around the box but as repathing is disabled in this version, them being unable to move was allowed.

In v2, deadlock happens quite often. If a robot bumps into another robot's box, it doesn't attempt to move around it and instead waits for the box to be moved. This of course leads to issues if the point that the robot is stopped on is in fact the pushing position the box's owner is trying to get to. Along with this, if two robots collide going in opposite directions, both will be unable to move as they wait for the position to be freed, which it never will be.

In ec1, deadlock happens less often, but still happens. This usually occurs during the pushing operations, as the robots are unable to repath the pushing operations. In this version, the robots are only able to repath during their traversal to their pushing position. Once there, they simply push the box in the direction of the door. This could be avoided to a lesser extent if a methodology to reroute the box pushing into different directions in the case of hitting other objects.

Difficulties:

- Figuring out the path planning was the first difficulty we faced, we had to draw a few grids on paper and place objects on them to create a good plan for movement.

- To actually implement those movements we created two algorithms:

  - The first would find "pushing spots" for the robots to go to from which the robot is in contact with the box and ready to push it, along with calculating the distances from the robot to its pushing spots and from the box to its door. This algorithm found these spots and the distances to them and moved the robots immediately, which posed some difficulties in debugging and wasn't very efficient for writing the output to the outputs file.

- The second algorithm depended heavily on the first one, but instead of moving immediately it would create a list of commands for the robot to do in order to complete the job, after creating the list the robot would attempt to follow these commands to complete the job.

Deadlocks:

A possible method for detecting deadlocks is to have another thread going that solely checks the state of the grid lock. If it finds that the grid lock isn't changing at all over some period of time, it is likely due to a deadlock. Due to this being a separate thread, this detection does not require that the robot itself check that it can or cannot move. In order to implement this, we could simply create a thread to do the detection. If a deadlock is detected, a rollback could be initiated with the addition of a change of pathing. This would allow for the robots to take different times to get to their locations, likely allowing for different paths to open up, reducing the chance of encountering a deadlock. If a deadlock is encountered again, a more involved method of pathing will likely have to be involved. This could be that during the pushing phase, if a deadlock is detected, the robots are allowed to find a new pushing path, likely in a direction that goes against the optimal path to the door. This change of pathing would allow for blocks being pushed against each other to be reduced. A deadlock would likely still occur in the case of having multiple boxes (>2 or so) collide.

Common deadlocks:

- Two boxes being pushed in the opposite direction from each other.
- A box being pushed hits a still box with the robot for said still box is unable to reach it because that moving box or its robot are in the way.