**REPORT**

Artificial Intelligence Assignment 1.

Members:

44679129 Guan, Qin

Yifei Wang

# Please define the agent design problem for this mover’s prototype.

**Action space (A)**

Shift the robotic arm up, down, left, right, and rotate the arm clockwise and counterclockwise.

**Percept space (O)**

The robotic arm can push one moving box or movable and it cannot collide with static obstacles.

**State space (S)**

The position and angle for the robotic arm, and all other positions for moving box, movable obstacles and static obstacles.

**World dynamics (T: SXA → S)**

The position of moving boxes and movable obstacles will change to next state if the robotic arm coincide with them.

**Percept function (Z: S → O)**

The robotic arm can only push one moving or movable obstacles, and it cannot collide with static obstacles. In addition, any objects in the environment cannot collide with each other.

**Utility function (U: S → real number)**

1. To push moving boxes or movable obstacles, the arms must coincide three-quarter of the length to side of box/obstacles.
2. Each unit of moving distance is 0.001
3. The boundary of the environment is from [0,0] to [1,1]
4. The length of robotic arm is 0.1

# Please describe your search method at the conceptual level (i.e., pseudo code and what abstract data structure is used for the container). If you use sampling-based method, please describe the strategy you apply or develop for each of its four components. Otherwise, please describe the details of your discretization method.

//For broad-first search (BFS):

//two kinds of BFS search (Arm and Box) are basically same.

**While** (the queue is not empty)

Poll the first position from the queue

**if** the position is the target:

**end**while and backtrack the path

**else do**

**For** (visit next position [upper, lower, left, right]){

**If** (the position is empty and is never visited){

Add the position to queue

Add the position to visited list

}

**//Main**

**Load** Input file

**While** (movingBox List is not empty)

**Do**

Calculate the Manhattan distance from each moving boxes to arm

Get the closest box based on the Manhattan distance.

Putall Boxes into ‘unmoved\_box’ except for the closest one.

**//Moving Box to the goal position**

Put the closest box’s position to the queue

**Do** BFS to find the box path

Use the first step’s direction to define the arm target orientation and position

**Do** BFS to find the arm path

**Merge** the two paths and add rotate paths if the box direction changed each time.

**Update** the arm position. Do it again until all the Boxes are pushed to their destination.

**Output** the Path.

**Run** visualization.

# Which class of scenarios do you think your program will be able to solve well? Please explain your answer.

* 1. The robotic arm cannot repeat same path for more than once. So the program can still be efficient even if there are many “dead end” in that scenarios.
  2. Our program can make the robotic arm always find the right side of box to coincide, so many collisions caused by rotation can be avoided.
  3. When the goal position of a moved moving box is on the path of other moving boxes, our program can avoid the collision between them under this scenario, because they are treated as obstacles.
  4. Under the scenarios that there are numerous available solutions, the broad-first search method we applied can always guarantee our solution has lowest cost which is determined by the nature of BFS.
  5. Our program will perform well if the distance between initial place and target place is short enough because we use basic BFS algorithm and further distance means larger search space that each point in will be searched.

# Under what situation do you think your program will fail? Please explain your answer.

* 1. The moving boxes or robotic arm are surrounding by static obstacles, in that case, there is no solution no matter how this program is designed.
  2. There is limited space between static obstacles and it only has solution when robotic arm rotates between 0 to 90 degree. This is because in our program, the arm can only be moving under 0 degree or 90 degrees.
  3. The arm is possible to collide with other object while rotation. The rotation space that the robotic arm traveled is not considered in our program.
  4. It will fail if there are solutions exist only if the movable obstacles have to be moved firstly. In our program, the movable obstacles and static obstacles are treated as same objects.

# Improvement

* 1. If we have more time to improve our algorithm, we suppose that we could use larger moving step, such as 0.01 as one unit, to find the path and then make the step smaller such that the results meet the requirement(0.001), because BFS needs such a long time(O(n2)) to search.
  2. Depending on the target direction, set possibility on each direction of next step could accelerate the search.
  3. Using other algorithms such as Dijkstra algorithm, A-star algorithm and B-star algorithm is much more efficient than BFS.
  4. Before implement our algorithm to the moving boxes, we can consider whether the moving obstacle can be moved away from the best path of the moving boxes, to shorten the path of moving box.

# Conclusion

As data science students, it is our pleasure to join the class and enjoy the assignment. As we are both beginners of java, there is a limit to build such a large program, but to be honest, we have done our best. We complete the Bread-Fist Search (BFS) algorithm although it costs a long time to search the path in this program and we have a deep understanding of this algorithm. We should have known that there exist several better well-known navigation algorithms, such as A-star algorithm and B-star algorithm, which are much better than what we used in this program.

Thanks to those who helped us.