R2-D2 is a robot that is trying to escape from an m x n grid. The problem is that the robot’s only way out is a teleport laying on one of the cells of the grid which can send it back to it’s planet. Yet, it is not as easy as just finding that cell and use it to get back. The grid also consists of three more things; pressure pads, obstacles and rocks. In order for the robot to use the teleport cell, he has to push a rock over each pressure pad first. Still not an easy task giving that the robot has some limitations. The robot can not push more than one rock at a time in any given direction neither push any of the prison’s obstacles. After all the pressure pads have a rock over them, the teleport cell is finally activated and the robot can head directly to it, use it and go back to his home planet.

To make use of the OOP concepts, we used java to implement this project. The project consists of 5 packages:

1. Search Package:

This package contains all the general and abstract classes to solve any general problem. It has 5 classes:

* State Abstract Class:

The State object denotes the state of a given Node. Since each problem defines it’s own state, the State class is an empty Abstract class that has only one abstract method which is the *heuristic(int n) method.* We will talk about what that method does later on.

* Node Class:

It defines a Node object of a general search tree. The Node object has a 5 tuple instance variables; parent of type Node which indicates the parent of this node, currentState of type State which denotes the state of the node, depth of type int indicating the level of the node in a given search tree, pathCost of type int which is the summation of the nodes’ costs till reaching that node and operator of type String indicating the operator that was made by the parent node to reach that node. Then there is one final variable which is the order of type int, it is used to order the node in the priority queue in case of greedy or A\* algorithms.

* Problem Abstract Class:

The Problem Class defines the 5 tuple of any general search problem. It has three instance variables; an array of type String operators which consists of all the possible operators of the problem, an array of type State stateSpace which is the finite state space of the problem and finally the initState of type State which defines the first state of the problem. Then there are the two missing items of the 5 tuple which are two abstract methods, the *goalTest(Node node)* which returns a Boolean of whether the goal test is passed or not and the *pathCost(Node n)* which defines the pathCost of a given node. Finally, we have three important abstract methods; the Expand(Node node) which returns an ArrayList of type Node containing all the reachable nodes from the given node, pastState(Node node) which returns a Boolean denoting whether the given node has a state that has been explored before or not, this method enhances the search performance and speed by eliminating redundant states and lastly the clearPastState() which clears the ArrayList holding the past states for a purpose that would be explained later on in this report.

* QueuingFunction Enum: Enumerators that differentiate between the different search algorithms.
* GeneralSearch Class:

This class holds most of the logic in this project. The class is defined by four instance variables; the problem of type Problem, the qingFunc which is an enum of type QueuingFunction stating which queuing function will be used in the search and two data structures. The reason we used two data structures in the search is that we have different search algorithms that need different data structures, we divided the algorithms into two sets. BFS, DFS and IDS use a double ended queue which eases the process of adding the nodes in either side depending on the algorithm. UC, A\* and Greedy algorithms use a priority queue which also eases the process of sorting the nodes inside the queue. Then we have the search() method which either returns the node that passed the goal test or null indicating that no solution is found. Depending on the qingFunc, the search() calls one of the six different search methods to handle the tree traversing and returns back the solution. We will dig deeper into how each algorithm is implemented. Finally we have the main method of the class and the project which is the search(Grid grid, QueuingFunction strategy, boolean visualization) method; as shown, the method takes a grid object of type Grid which defines the initial grid of the problem, a queuing function, and visualization of type Boolean which indicates of the user wants to visualize the problem or not.