**CSEN 901: Artificial Intelligence**

**R2D2 Project**

**Project 2 Report**



**Team:**

**Youssuf Radi 31-2140 T10**

**Hazem El-Agaty 31-1079 T08**

**Mohamed Shokr 31-1802 T09**

**Introduction**

This section will have a **brief description of the problem**. The 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 its 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.

For the robot to use the teleport cell, he must push a rock over each pressure pad first. Still not an easy task giving that the robot has some limitations. The robot cannot 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.

**Implementation**

1. **SYNTAX AND SEMANTICS:**

*teleport(X,Y).*

teleport initial positions declared at cell X,Y

*obstacle(X,Y).*

obstacle initial positions declared at cell X,Y

*pad(X,Y).*

pad initial positions declared at cell X,Y

*w(W).*

grid width declaration every predict defines a column with W for every column

*h(H).*

grid height declaration every predict defines a row with H for every row

*run(Q):-*

main predicate to run the code where Q represent query to be used

*run\_helper(Q, I):-*

helper for the main predicate just to have the initial starting depth called I default 1

*run\_helper2(\_, \_, R):-*

checks if R is not depth\_limit\_exceeded then the agent have found a solution so it stops and returns this solution.

*run\_helper2(Q, I, R):-*

checks if R is depth\_limit\_exceeded then the agent have not found a solution so it increments the depth to search in a deeper level.

*agent(2,2,s0).*

agent intial position at situation s0 at cell X,Y.

*agent(I, J, result(down,S)):-*

agent fluent axioms to move the agent they check if an agent can be in position I,J as a result of an action A in a previous situation S.The possible actions are either down, up, right, left.This predicate is called with every possible action.

*move\_agent(I, J, \_, \_, S):-*

move\_agent is used to check if agent can be in the I,J given the previous state S.moreover this predicate takes two extra parameters K,L they are used to generalize the method where K represent change in I and L represent change in J. Also it has 2 cases, case to check if the agent is moving to a cell I,J which was completely free in situation S and exists inside the grid, or the case to check if the agent is moving to a cell I,J which is not free and if it contains a rock that can be pushed so the agent will be also alowed to move.

*rock(0,1,s0).*

rock intial position at situation s0 at cell X,Y

*rock(I, J, result(down, S)):-*

rock fluent axioms to move the rock they check if an agent can be in position I,J as a result of an action A in a previous situation S. The possible actions are either down, up, right, left. This predicate is called with every possible action.

*move\_rock(I, J, K, L, S):-*

move\_rock is used to check if the rock can be in the I,J given the previous state S. Moreover this predicate takes two extra parameters K,L they are used to generalize the method where K represent change in I and L represent change in J.

Checks the case if the rock is moved to a cell I,J which was completely free in situation S and exists inside the grid and the rock in previous state was not on a pad, also it checks that an agent was just near the rock and did the same movement in order for the rock to move.

1. **SUCCESSOR STATE AXIOMS:**

In the problem, we have two properties that change over time; the agent’s location and the rocks’ locations. Thus, the predicates defining them must have an extra situation argument which is initially s0. Other situations can be generated from the initial situation by doing one or more actions.

A situation example in our case could be something like this:

*S = result(up,(result(left, s0))).*

The previous situation is reached from the initial situation by doing a *left* then *up* actions as illustrated in the example as in our problem we only have four actions defining the four directions that the agent go in.

In order to define such state, we must define successor state axioms for the predicates that change over time. In our problem we defined two successor state axioms; agent(I, J, result(A, S)) and rock(I, J, result(A, S)) where I and J are the column and row location in the grid, A is the action done on a situation and S is the situation. The following shows an illustration on how each successor state axiom work.

*agent(I, J, result(A, S)):*

* For an agent to move from situation S to Situation result(A, S), the A which is the action done on S has to be valid. Since we have four different actions, we will mention the next cell location resulting from doing a movement action on the current cell location as the adjacent cell. In our problem there are two cases for an agent to do a valid movement; either the adjacent cell is free, which means that there exists a cell which is not a rock or an obstacle, or the adjacent cell has a rock that can move. A rock can move if it has an adjacent cell which is not a rock or an obstacle.

This check is done using the *move\_agent(I, J, K, L, S)* where I and J are the column row location of the agent in the grid and K and L are two variables denoting the movement of the agent and S is the situation. If the *move\_agent* predicate’s call succeeds, the agent predicate will recursively call itself with the previous location before doing such an action A on the previous situation S and the previous situation S.

* Since it does not make sense for the agent to stay at the same location for two following situations, the agent’s persistence was not implemented. Thus, if the agent’s movement failed, it will denote that the agent is stuck and a false should be returned.

*rock(I, J, result(A, S)):*

* The rock’s successor state axiom follows the same implementation procedure as the agent’s with few modifications.
* For the rock to move from situation S to Situation result(A, S), the action done on S must also be valid. The movement action of a rock is valid if an adjacent cell of the rock exists and does not have a rock or an obstacle on it. But since the action denotes the movement of the agent and not the movement of the rock, we have to check also that the agent was on the opposite side of the adjacent cell of the rock and the action was done in a direction towards the rock. For example, if the action was *up* and the cell above the rock is free, the agent must be in the cell below the rock in order to push it. The check is done in a separate predicate as before which is *move\_rock(I, J, K, L, S).*
* Here, we must have a persistence for the successor state of the rock since the agent can move a valid movement away from the rock, and thus the rock should stay as it is at the same location. So if the move\_rock check failed, the rock axiom is recursively called with the same location for the previous situation, meaning that the action done by the agent did not affect the rock’s location.

**3. GRID IMPLEMENTATION:**

**Query**

In order to get the agent’s solutions for the problem, we have to query the knowledge base for a situation in which the goal test is satisfied. This is done in prolog using unification. The query checks if there is a situation in which the agent is on the teleport location and all rocks are on pads. The situation S is then unified with the correct result chains which define the set of actions done from the initial situation to the goal. This is illustrated further in the examples.

**Examples**

The Grid:

\_ \_ \_

| p | O | p |

\_ \_ \_

| r | T | r |

\_ \_ \_

| | A | |

\_ \_ \_

?- run((agent(1,1,S),rock(0,0,S),rock(2,0,S))).

S = result(right, result(up, result(left, result(down, result(left, result(up, result(right, s0))))))) ;

S = result(right, result(up, result(left, result(left, result(down, result(up, result(right, s0))))))) ;

S = result(left, result(up, result(right, result(down, result(right, result(up, result(left, s0))))))) ;

S = result(left, result(up, result(right, result(right, result(down, result(up, result(left, s0))))))) ;

S = result(right, result(up, result(left, result(down, result(left, result(up, result(right, s0)))))))

The Grid:

\_ \_ \_

| | | p |

\_ \_ \_

| o | T | r |

\_ \_ \_

| p | r | A |

\_ \_ \_

?- run((agent(1,1,S),rock(2,0,S),rock(0,2,S))).

S = result(up, result(left, result(down, result(up, s0)))) ;

S = result(left, result(up, result(right, result(left, s0)))) ;

S = result(up, result(left, result(down, result(up, s0)))) ;

S = result(left, result(up, result(right, result(left, s0)))) ;

S = result(down, result(up, result(up, result(left, result(down, result(up, s0)))))) ;

S = result(down, result(up, result(left, result(up, result(right, result(left, s0)))))) ;

S = result(up, result(right, result(left, result(left, result(down, result(up, s0)))))) .