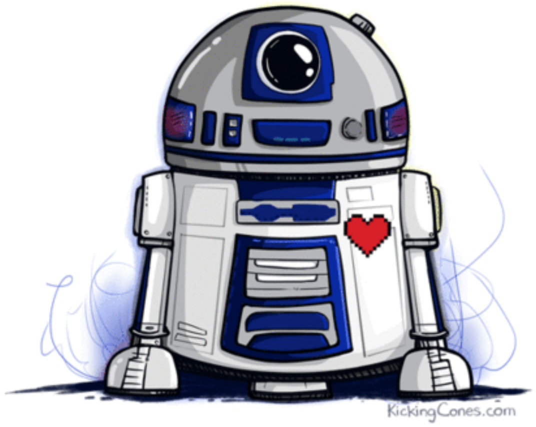
Project 2 Report
Logic-based Agent



By:

Narihan Ellaithy 31-8027

Safa Ads 31-6338

- A brief description of your understanding of the problem.

  In the project, the design was of a logic-based version of HelpR2D2 implemented in the previous project. A logic-based agent stores its knowledge about the world in the form of facts in the knowledge base and infers new sentences. The agent takes an action based on the sentences in the knowledge base. The language used was Prolog logic programming language.

- A description of the modified implementation of GenGrid.

  The modified part in generate grid is adding agent, teleportal, rocks and pressure pads position in the form of predicate(Ex: agent(X,Y,S).), where X and Y are the row and column where the agent, teleportal, rock or pressure pad is randomly allocated and then writing these predicates to (.pl) file.

- A discussion of the syntax and semantics of the action terms and predicate symbols you employ.

  The actions used are either forward, backward, left or right. These represent the values in the first argument of result.

1. wall(X,Y): The limits of the grid are checked so that the agent or rock don't move outside the grid.

2. agent(X, Y, result(forward/backward/left/right,S)): moves agent within the boundaries of the wall where two predicates will be found, one for when the agent moves to an empty cell and the other for when the agent moves towards a rock.

3. rock(X,Y, result(forward/backward/left/right,S)): agent moves rock within boundaries of the wall where two predicates will be found, one for when the agent pushes the rock to an empty cell and the other for the persistence of the rock where an agent moves away from the rock and didn't push it therefore the moving of the rock failed.

4. push(X,Y,I,J,S): this predicate is used as a helper for the movement of the rock. Agent pushes rock within the boundaries of the wall by checking on X and Y. I and J are used as constants to determine the previous state position of the agent (two cells away from the current position of the rock) depending on the type of action, for example if the rock moved forward in X and Y then the rock was in X+1 in situation S(before moving forward) and the agent was in X+2(I) in situation S.

Result of moving forward explanation grid

	Rock(X,Y, result(forward,S))	
	Agent(X,Y, result(forward,S)) / Rock(X+1,Y,S)	
	Agent(X+2,Y,S)	

- A discussion of your implementation of the successor-state axioms.

Actions are performed on Agent and rocks in this project, and are defined in terms of their effects. In this implementation, successor-state axioms were written for each of agent and rock since they are the two changing elements in the implementation. The axioms written represent the current state after the actions were already done and explain the path the agent might have taken in the previous state to come to the current one.

Agent:

Agent Effect: The agent can do four actions which are moving forward, backward, left or right, for each action there were two possible ways were the agent can move, the first way if the cell that the agent moved to is an empty cell and this is known by checking if the next cell isn't obstacle or rock in the current state, and the second way is if the cell that the agent moved to contains rock however the cell that is next to the cell contacting rock that the rock will be pushed to is empty which is also known by checking if this cell has no obstacles or no wall in the current state, so the agent pushes the rock and the movement of the rock is handled in the successor-state axiom of the rock.

Agent Frame: There was no need to handle frames because if any of the effect conditions were not satisfied the agent won't move.

Rock:

Rock Effect: Rock can either be pushed forward, backward, left or right. It can be pushed to any of these directions if two conditions are satisfied, which are that the cell that the rock is pushed to is empty in the current state which is known by checking if this cell has no rocks and obstacles, the second condition is that the cell behind the cell containing rock is agent in the current state to make sure that the rock will be pushed by the agent.

Rock Frame: The rock frame is negating all the previous conditions, if the cell that the rock will be pushed to the next state contains rock or obstacle, or the cell behind the rock is not agent, the rock will not be pushed in the next state. Therefore the agent wasn't able to push the rock either because it wasn't around the rock to even push it or the rock simply can't be pushed to the next cell because of its conditions.

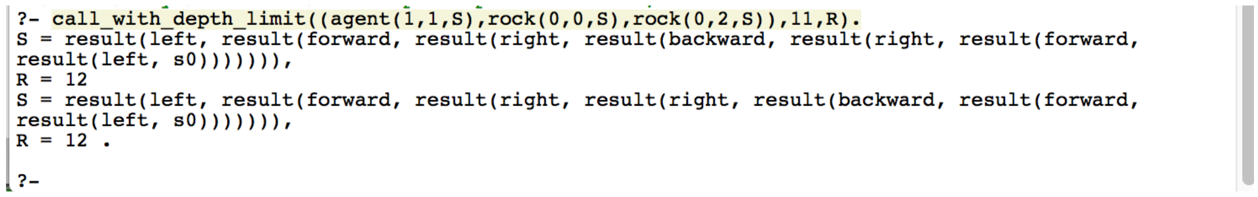
- A description of the query used to generate the plan.

  The predicate used to generate the plan is iterative(C,S) and inside it call_with_depth_limit is called with the query. C is the starting depth and S is the state variable. Iterative uses call_with_depth_limit(Q,C,R). If R isn't depth_limit_exceeded then it breaks, otherwise Cis incremented with 1 and iterates recursively with the incremented C. Q is the query on agent and rock, for example call_with_depth_limit((agent(2,2,S),rock(0,2,S)),C,R).

  The query in the .pl file is for the following grid:

Pressure Pad	Obstacle	Pressure Pad
Rock	Teleportal	Rock
	Agent	

The result of this query would be:



```
?- call_with_depth_limit((agent(1,1,S),rock(0,0,S),rock(0,2,S)),11,R).
S = result(left, result(forward, result(right, result(backward, result(right, result(forward,
result(left, s0))))))),
R = 12
S = result(left, result(forward, result(right, result(right, result(backward, result(forward,
result(left, s0))))))),
R = 12 .

?-
```

This means that the agent moved Left -> forward -> right -> backward -> right -> forward -> left yielding this grid:

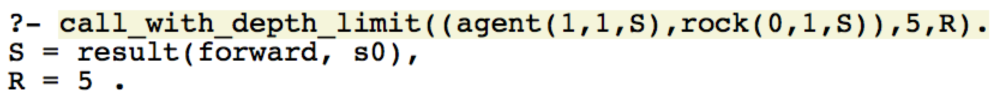
Rock on Pressure Pad	Obstacle	Rock on Pressure Pad
	Agent on Teleportal	

To query a different combination by hard coding it (not using the facts that Java yielded), change the initial place of the agent and rock(s) then in iterative rule change the Q(query) in call_with_depth_limit (first argument) to the desired query.

- At least two running examples from your implementation.

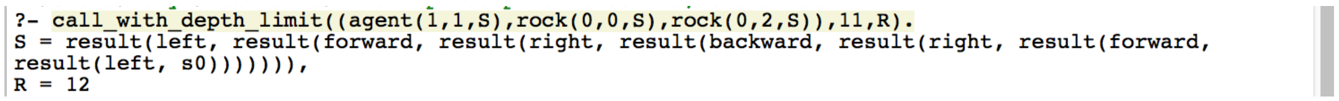
  Note: call_with_depth_limit is used instead of iterative(C,S) just to show the queries done for the examples.

Initially: agent(2,1,s0) and rock(1,1,s0):



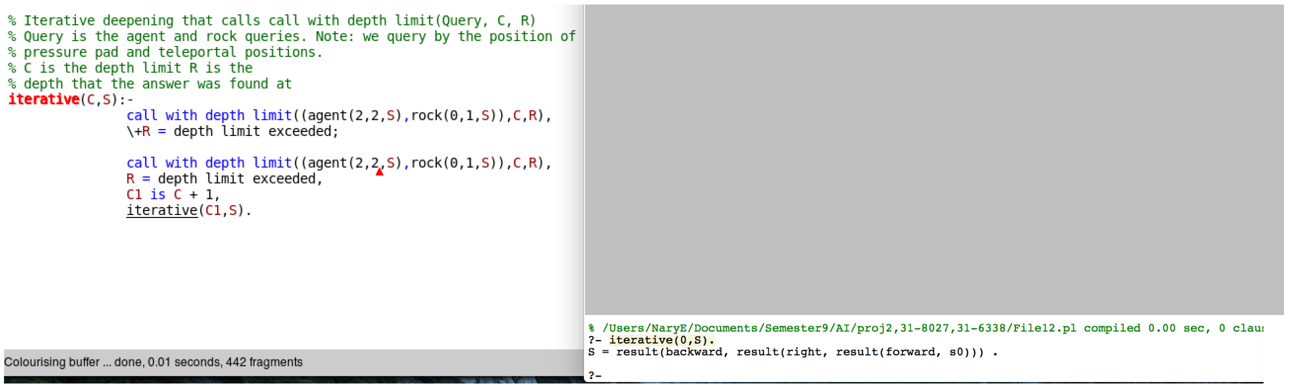
```
?- call_with_depth_limit((agent(1,1,S),rock(0,1,S)),5,R).
S = result(forward, s0),
R = 5 .
```

Initially: agent(2,1,s0), rock(1,0,s0) and rock(1,2,s0):



```
?- call_with_depth_limit((agent(1,1,S),rock(0,0,S),rock(0,2,S)),11,R).
S = result(left, result(forward, result(right, result(backward, result(right, result(forward,
result(left, s0))))))),
R = 12
```

Initially: rock(1,1,s0) and agent(2,1,s0):



```
% Iterative deepening that calls call with depth limit(Query, C, R)
% Query is the agent and rock queries. Note: we query by the position of
% pressure pad and teleportal positions.
% C is the depth limit R is the
% depth that the answer was found at
iterative(C,S):-
        call with depth limit((agent(2,2,S),rock(0,1,S)),C,R),
        \+R = depth limit exceeded;

        call with depth limit((agent(2,2,S),rock(0,1,S)),C,R),
        R = depth limit exceeded,
        C1 is C + 1,
        iterative(C1,S).
```

Colourising buffer ... done, 0.01 seconds, 442 fragments

```
% /Users/NaryE/Documents/Semester9/AI/proj2,31-8027,31-6338/File12.pl compiled 0.00 sec, 0 clau
?- iterative(0,S).
S = result(backward, result(right, result(forward, s0))) .

?-
```