AI Project 2 Report

1) Predicates

list_eq(L1,L2): this predicate is responsible for recursively checking if two lists are equal by comparing the heads of both L1 and L2 and then recursively calling the list_eq for the rest of both lists excluding the two heads. The predicate will stop if the heads of the two argument lists are not equal which indicates that the original L1 and L2 are not equal or if the two lists are empty, a case in which the original lists L1 and L2 are equal.

mem_2d(Elem, List): this predicate is responsible for checking if the Elem argument, which is in fact a list of elements, exists inside the second argument List, which is infact a list of lists. This is done by checking if the Elem equals the head of the list using the list_eq predicate, if not then the predicate will recursively call itself again with Elem and the List except for its head until it finds a matching element and return true or no elements remain in the list and return false.

first_elem([H1|_], H): this predicate is responsible for getting the first element in a list or checking if an element H is the first element in the list specified in the first argument of the predicate.

last_elem(List, Elem): this predicate is responsible for getting the last element of the argument **List** of elements by checking if an element is the last in the **List** by recursively calling itself with the first arguments being the original list except for the head and the second argument being as it is. The predicate will stop if the list has one element and will try to map/unify the element with the argument Elem.

2) Successor state axioms

A)neo_at(x_location_of_neo, y_location_of_neo, result(A,state)): this axiom is responsible for representing the change of the location of neo after performing a certain action on a given state of the world where the x_location and the y_location both represent the current location of neo in the state that resulted from applying an action on the previous state of the world which encompass the x and y location of neo in the previous state before applying the action.

- **neo_at(X, Y, s0):** the base case of this axiom is the one concerned with the initial state s0, in this case the X and Y which resemble the location of neo should correspond to the initial location of neo in s0 which is defined in the KB as neo_loc(A,B) a case in which A should be unified with X and B should be unified with Y.

- **neo_at(X, Y, result(A,S)):** in this instance of the axiom, the X and Y represent the location of neo in the state that results from applying the action A on the previous state. The actions that could be applied on a state are carry, drop, left, right, up, and down. For this predicate to hold true, certain preconditions regarding the action A have to be satisfied so that the X and Y of the resulting state could be derived from the previous state S when performing action A.

    - For the carry action these preconditions should fulfill that the current X and Y of neo are the same X and the same Y of the previous state S and that the X and Y of the previous state also correspond to the same location of an uncarried hostage.

    - For the drop action the preconditions should satisfy that the current X and Y of neo are the same as the X and Y of the telephone booth and in the previous state S. in this case the X and Y of neo in the resulting state are same as the previous state as the drop action doesn't make neo move in the grid.

    - For the left action the preconditions that should be satisfied are that the Y location of neo in the resulting state is (Y_cur = Y_pre + 1) where Y_pre the y of neo in state S. the current Y of neo should also be smaller the the maximum

width of the grid for the left action applied to be a valid action in the previous state
For the right action the preconditions that should satisfy that the Y location of neo in the resulting state is
(Y_cur = Y_pre - 1).
where Y_pre the y of neo in state S. the current Y location of neo should also be greater than the minimum width of the grid for the right action applied to be a valid action in the previous state.

- For the up action the preconditions that should satisfy that the X location of neo in the resulting state is
(X _cur = X _pre + 1). where X_pre the x of neo in state S. the current X of neo should never be smaller than the minimum height of the grid for the up action to be a valid operator in the previous state.

- For the down action the preconditions that should satisfy that the X location of neo in the resulting state is
(X _cur = X _pre - 1). where X_pre the x of neo in state S. The current X of neo should never be greater than the maximum height of the grid for the down action to be a valid operator in the previous state.

B)hostage_at(X_loc, Y_loc, X_cur_hostage, Y_cur_hostage, X_other_hostage, Y_other_hostage, carried, capacity, result(A,state)): this axiom is responsible for representing the change of the x and y location of a hostage, if the hostage is carried or not, and the capacity of neo in the state that results from performing an action A on the previous state S where the state S encompasses all of this information before applying the action. The first two arguments of the predicate are the x and y location of the

current hostage under investigation in the resulting state, the X_cur_hostage, Y_cur_hostage both represent the original location of the hostage under investigation in the initial state s0. X_other_hostage, Y_other_hostage both represent the original x and y location of the other hostage in the initial state s0. The carried represents if the hostage under investigation is being carried in the resulting state after applying the action A on the previous state S. the capacity represents the number of hostages that neo can carry in the state that result from applying the action A on the previous state S. the last argument represents the state of the world that results from applying action on the state S.

- **hostage_at(X_cur_hostage ,Y_cur_hostage, X_cur_hostage, Y_cur_hostage, _, _, false, Capacity, s0):** the base case of this axiom is concerned with the initial state where in the initial state the location of the hostage under investigation should be the initial location in the world and the hostage shouldn't be carried by neo. In this initial state neo shouldn't be holding any hostages so the capacity is the original capacity of neo in the in the KB (Capacity = capacity(Capacity)).

- **hostage_at(X, Y, X_cur_hostage, Y_cur_hostage, X_other_hostage, Y_other_hostage, Carried, Capacity, result(A,S)):** for this instance of the predicate to hold true certain preconditions has to be fulfilled so that the current X, Y, Carried, and capacity in the resulting state could be derived from the previous state S after applying the action A. These preconditions depend on the action that was applied where the possible actions that could be applied are carry, drop, left, right, up, and down.

    - In case the action that was performed on the previous state S was up and the hostage is carried in the resulting state, the x location of the hostage in the previous state should be the current x location +1 and the current x is smaller than the maximum x dimension for this action to be applicable on the previous state. This case corresponds to the situation in which neo is carrying a hostage and moving across the grid

so the location of the hostage should be the same as the location of neo.

- In case the action was down and the hostage hostage is carried in the resulting state, the previous x location of the hostage is the current x location - 1 and that the and and the current x is greater than the minimum x dimension.This case corresponds to the situation in which neo is carrying a hostage and moving across the grid so the location of the hostage should be the same as the location of neo.

- In case the action was left and the hostage hostage is carried in the resulting state, the previous y location of the hostage is the current y location + 1 and that the and the current y is smaller than the maximum y dimension.This case corresponds to the situation in which neo is carrying a hostage and moving across the grid so the location of the hostage should be the same as the location of neo.

- In case the action was right and the hostage hostage is carried in the resulting state, the previous y location of the hostage is the current y location - 1 and that the and the current y is greater than the minimum y dimension.This case corresponds to the situation in which neo is carrying a hostage and moving across the grid so the location of the hostage should be the same as the location of neo.

- In these previous actions and their preconditions, if neo is carrying a hostage and he decides to move up, down, left, right, then not only the location of neo will change but also the location of all the carried hostage. This explains why we update the x and the y of the hostage under investigation.

- In case the hostage is not carried in the resulting state and if the action that was applied was either up, down, left, or right, then all the arguments of the predicate which resemble some properties of the world in the state that results from applying

the action on the previous state remains the same. The reason is that if Neo was not carrying the hostage under investigation and decided to do a movement action, then the hostage for sure won't be carried in the resulting state and so his location. Also Neo's capacity won't be affected as the action being applied is a movement action.

- In case the action that was applied is drop and the hostage is not carried in the resulting state, then the x and y locations of the hostage in the previous state should be the location of the telephone booth. Regarding the capacity of neo, if neo was carrying one hostage then his capacity in the previous state = capacity in the current state - 1. If neo was carrying two hostages then the capacity of neo in the previous state = capacity of neo in the current state - 2. The intuition of this action is that neo can only drop if he is at the telephone booth and in this case, the carried hostages should also be at the telephone booth and if neo dropped all carried hostages then the capacity of neo should increase by a number equal to the dropped hostages.

- The drop action can also be performed on the previous state S if the hostage is being carried and his location is the telephone booth or neo is carrying another hostage and not carrying the hostage under investigation. In the first case, the capacity of neo in the resulting state = capacity in previous state + 1 and the location of the hostage under investigation is at the telephone booth and the hostage was carried. In the second case the capacity of neo in the resulting state also = capacity in previous state + 1 but the hostage is not carried and is not at the telephone booth.

- For the carry action that was applied in a previous state there exist two cases. The first case is that the carry action was applied to carry the hostage under investigation. In this case the hostage should be carried in the resulting state, in

addition, the capacity of the current state = capacity of the previous state - 1 and the x and y of the resulting state is the same as the previous state. The second case of the carry action deals with the possibility that the action was applied on the previous state to carry the other hostage. If it's the case then the x and the y location in the resulting state should correspond to the x and y location of the other hostage, this is because the carry action is applied on the hostage under investigation which in this case is the other hostage. If it's not the case, then the hostage under investigation in the current state should be carried. In either case the capacity of neo in the resulting state = capacity in the previous state - 1.

3) goal, goal_help, and ids predicates

- **goal(S)** predicate is used for both verification and generation of a solution of the matrix problem given an initial state indicated by the Knowledge base. For the verification, the goal(S) calls a helper predicate called goal_help(S) that is used to check whether the given sequence of actions carried on the initial state S0 is correct and actually leads to a goal state. For the generation, the goal(S) predicate relies on the ids predicate that uses depth limited search to find a goal state.

- **goal_help(S)** predicate is used to verify that a given state S which is a consecutive action carried in the initial state s0 is a goal state. This is done by verifying that neo is at the telephone booth location, all the hostages are at the telephone booth and not carried ,if there exists any hostages in the problem, and neo's capacity is full.
- **ids(N,S):** this predicate is used to generate a solution of the matrix problem with the employment of *call_with_depth_limit* Predicate where N is the maximum depth of the search and S is placeholder for the goal state.
  If *call_with_depth_limit* Predicate exceeds the depth limit, the ids predicate will increase the depth and will call itself

recursively afterwards. The predicate will keep repeating the process till it reaches a goal state.