# Fr. Conceicao Rodrigues College of Engineering

**Department of Computer Engineering**

EXPERIMENT NO. 4

| | |
|---|---|
| **Practical No:** | 4 |
| **Title:** | Write a program to find a solution to Monkey Banana problem in Prolog. |
| **Date of Performance:** | 12/02/2025 |
| **Date of Submission:** | 27/04/2025 |
| **Roll No:** | 9913 |
| **Name of the Student:** | Mark Lopes |

**Rubrics for Evaluation:**

| Sr. No | Performance Indicator | Excellent | Good | Below Average | Total Score |
|---|---|---|---|---|---|
| 1 | On time Completion & Submission (01) | 01 (On Time ) | NA | 00 (Not on Time) | |
| 2 | Logic/Theory understanding(02) | 02(Correct) | NA | 01 (Tried) | |
| 3 | Coding Standards (03): Comments/indention/Naming conventions Output/Test Cases | 03(All used) | 02 (Partial) | 01 (rarely followed) | |
| 4 | Post Lab Assignment (04) | 04(done well) | 3 (Partially Correct) | 2(submitted) | |

| Academic Year | 2024-25 | Estimated Time | Experiment No. 4 – 02 Hours |
|---|---|---|---|
| Course & Semester | T.E. (CE) – Sem. VI | Subject Name | CSC604: Artificial Intelligence |
| Chapter No. | 03 | Chapter Title | Solving Problems by Searching |
| Experiment Type | Software/Finding Solution | Software | Prolog/Python |

**AIM:** Write a program to find a solution to Monkey Banana problem in Prolog/Python.

## I. OBJECTIVES

• Solving classical problems using Prolog.

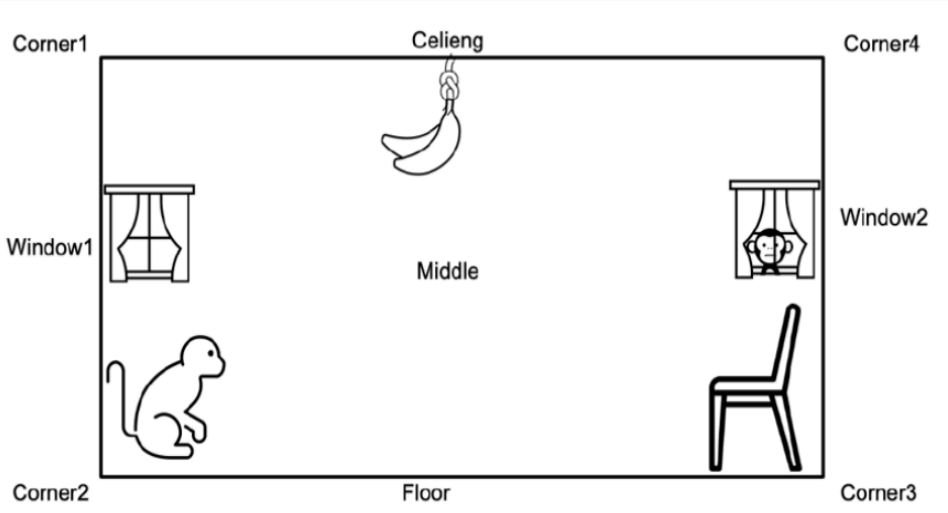
### 2. Finding solution to the Problem

"A monkey is in a room. A bunch of bananas is hanging from the ceiling. The monkey cannot reach the bananas directly. There is a chair/box in the corner of the room. How can the monkey get the bananas?"

**Procedure:**

The solution of the problem is of course that the monkey must push the chair/box under the ananas, then stand on the chair/box and grab the bananas. The purpose of the problem is to raise the question: Are monkeys intelligent? Both humans and monkeys have the ability to use mental maps to remember things like where to go to find shelter or how to avoid danger. They can also remember where to go to gather food and water, as well as how to communicate with each other. Monkeys have the ability not only to remember how to hunt and gather but they also have the ability to learn new things, as is the case with the monkey and the bananas. Even though that monkey may never have entered that room before or had only a chair for a tool to gather the food available, that monkey can learn that it needs to move the chair/box across the floor, position it below the bananas and climb the chair/box to reach for them.

**MONKEY WORLD**



**The action available to the monkey include:**

(1) grasp banana,

(2) climb box,

(3) push box,

(4) walk around.

Not all moves are possible in every possible state of the world. For example, the move 'grasp' is only possible if the monkey is standing on the box directly under the banana (which is in the middle of the room) and does not have the banana yet. Such rules can be formalized in Prolog as a three-place relation narned move: move(Statel, M, State2)

**The three arguments of the relation specify a move thus: Statel --------) State2**

**M**

Statel is the state before the move. M is the move executed and State2 is the state after the move. The move 'grasp', with its necessary precondition on the state before the move, can be defined by the clause:

```
move( state( middle, onbox, middle, hasnot),     % Before move
      grasp,                                     % Move
      state( middle, onbox, middle, has) ).      % After move
```

The other two types of moves, 'push' and 'climb', can be similarly specified. The main kind of question that our program will have to answer is: Can the monkey in some initial state S get the banana? This can be formulated as a predicate

canget( S)

where the argument S is a state of the monkey world. The program for canget can be based on two observations:

(1) For any state S in which the monkey already has the banana, the predicate canget must certainly be true; no move is needed in this case. This corresponds to the Prolog fact:

canget( state( -, -, -, has) ).

(2) In other cases one or more moves are necessary. The monkey can get the banana in any state S1 if there is some move M from state S1 to some state S2, such that the monkey can then get the banana in state S2 (in zero or more moves). A Prolog clause that corresponds to this rule is:

canget( Sl) :- move( 51, M, S2),canget( S2).

**CODE:**

```
% move/4 defines the actions

% Grasp: If the monkey is on the box and the box is in the middle, it can grasp the banana.
move(state(middle, onbox, middle, hasnot), grasp, state(middle, onbox, middle, has), [grasp]).

% Climb: The monkey climbs onto the box if they are at the same horizontal position.
move(state(P, onfloor, P, H), climb, state(P, onbox, P, H), [climb]).

% Push: The monkey can push the box from one position to another.
move(state(P1, onfloor, P1, H), push(P1, P2), state(P2, onfloor, P2, H), [push(P1,P2)]).

% Walk: The monkey can move horizontally from one position to another.
move(state(P1, onfloor, B, H), walk(P1, P2), state(P2, onfloor, B, H), [walk(P1,P2)]).

% Goal: The goal state is when the monkey has the banana.
canget(state(_, _, _, has), [], []).

% Recursively find the sequence of actions to reach the goal and display the intermediate states.
canget(State1, Sequence, [State1 | Path]) :-
    move(State1, Move, State2, Action),
    canget(State2, SubGoal, Path),
    append(Action, SubGoal, Sequence).

% Base case: If the monkey already has the banana, the sequence is empty, and we stop.
canget(state(_, _, _, has), [], []).
```

**OUTPUT:**

```
?- trace.
true.

[trace]  ?- canget(state(atdoor, onfloor, atwindow, hasnot), Sequence, Path).
   Call: (8) canget(state(atdoor, onfloor, atwindow, hasnot), _2574, _2576) ? creep
   Call: (9) move(state(atdoor, onfloor, atwindow, hasnot), _2862, _2864, _2866) ? creep
   Exit: (9) move(state(atdoor, onfloor, atwindow, hasnot), walk(atdoor, _2848), state(_2848, onfloor, atwindow, hasnot), [walk(atdoor, _2848)]) ? creep
^  Call: (9) format('Exploring: ~w -> ~w using action: ~w~n', [state(atdoor, onfloor, atwindow, hasnot), state(_2848, onfloor, atwindow, hasnot), walk(atdoor
, _2848)]) ? creep
Exploring: state(atdoor,onfloor,atwindow,hasnot) -> state(_2848,onfloor,atwindow,hasnot) using action: walk(atdoor,_2848)
^  Exit: (9) format('Exploring: ~w -> ~w using action: ~w~n', [state(atdoor, onfloor, atwindow, hasnot), state(_2848, onfloor, atwindow, hasnot), walk(atdoor
, _2848)]) ? creep
   Call: (9) canget(state(_2848, onfloor, atwindow, hasnot), _2908, _2842) ? creep
   Call: (10) move(state(_2848, onfloor, atwindow, hasnot), _2914, _2916, _2918) ? creep
   Exit: (10) move(state(atwindow, onfloor, atwindow, hasnot), climb, state(atwindow, onbox, atwindow, hasnot), [climb]) ? creep
^  Call: (10) format('Exploring: ~w -> ~w using action: ~w~n', [state(atwindow, onfloor, atwindow, hasnot), state(atwindow, onbox, atwindow, hasnot), climb])
 ? creep
Exploring: state(atwindow,onfloor,atwindow,hasnot) -> state(atwindow,onbox,atwindow,hasnot) using action: climb
^  Exit: (10) format('Exploring: ~w -> ~w using action: ~w~n', [state(atwindow, onfloor, atwindow, hasnot), state(atwindow, onbox, atwindow, hasnot), climb])
 ? creep
   Call: (10) canget(state(atwindow, onbox, atwindow, hasnot), _2948, _2894) ? creep
   Call: (11) move(state(atwindow, onbox, atwindow, hasnot), _2954, _2956, _2958) ? creep
   Fail: (11) move(state(atwindow, onbox, atwindow, hasnot), _2954, _2956, _2958) ? creep
   Redo: (10) canget(state(atwindow, onbox, atwindow, hasnot), _2948, _2894) ? creep
   Fail: (10) canget(state(atwindow, onbox, atwindow, hasnot), _2948, _2894) ? creep
   Redo: (10) move(state(_2848, onfloor, atwindow, hasnot), _2914, _2916, _2918) ? creep
   Exit: (10) move(state(atwindow, onfloor, atwindow, hasnot), push(atwindow, _2900), state(_2900, onfloor, _2900, hasnot), [push(atwindow, _2900)]) ? creep
^  Call: (10) format('Exploring: ~w -> ~w using action: ~w~n', [state(atwindow, onfloor, atwindow, hasnot), state(_2900, onfloor, _2900, hasnot), push(atwind
ow, _2900)]) ? creep
Exploring: state(atwindow,onfloor,atwindow,hasnot) -> state(_2900,onfloor,_2900,hasnot) using action: push(atwindow,_2900)
^  Exit: (10) format('Exploring: ~w -> ~w using action: ~w~n', [state(atwindow, onfloor, atwindow, hasnot), state(_2900, onfloor, _2900, hasnot), push(atwind
ow, _2900)]) ? creep
   Call: (10) canget(state(_2900, onfloor, _2900, hasnot), _2960, _2894) ? creep
   Call: (11) move(state(_2900, onfloor, _2900, hasnot), _2966, _2968, _2970) ? creep
   Exit: (11) move(state(_2900, onfloor, _2900, hasnot), climb, state(_2900, onbox, _2900, hasnot), [climb]) ? creep
^  Call: (11) format('Exploring: ~w -> ~w using action: ~w~n', [state(_2900, onfloor, _2900, hasnot), state(_2900, onbox, _2900, hasnot), climb]) ? creep
Exploring: state(_2900,onfloor,_2900,hasnot) -> state(_2900,onbox,_2900,hasnot) using action: climb
^  Exit: (11) format('Exploring: ~w -> ~w using action: ~w~n', [state(_2900, onfloor, _2900, hasnot), state(_2900, onbox, _2900, hasnot), climb]) ? creep
```

```
^  Exit: (11) format('Exploring: ~w -> ~w using action: ~w~n', [state(_2900, onfloor, _2900, hasnot), state(_2900, onbox, _2900, hasnot), climb]) ? creep
   Call: (11) canget(state(_2900, onbox, _2900, hasnot), _3000, _2946) ? creep
   Call: (12) move(state(_2900, onbox, _2900, hasnot), _3006, _3008, _3010) ? creep
   Exit: (12) move(state(middle, onbox, middle, hasnot), grasp, state(middle, onbox, middle, has), [grasp]) ? creep
^  Call: (12) format('Exploring: ~w -> ~w using action: ~w~n', [state(middle, onbox, middle, hasnot), state(middle, onbox, middle, has), grasp]) ? creep
Exploring: state(middle,onbox,middle,hasnot) -> state(middle,onbox,middle,has) using action: grasp
^  Exit: (12) format('Exploring: ~w -> ~w using action: ~w~n', [state(middle, onbox, middle, hasnot), state(middle, onbox, middle, has), grasp]) ? creep
   Call: (12) canget(state(middle, onbox, middle, has), _3040, _2986) ? creep
   Exit: (12) canget(state(middle, onbox, middle, has), [], []) ? creep
   Call: (12) lists:append([grasp], [], _3042) ? creep
   Exit: (12) lists:append([grasp], [], [grasp]) ? creep
   Exit: (11) canget(state(middle, onbox, middle, hasnot), [grasp], [state(middle, onbox, middle, hasnot)]) ? creep
   Call: (11) lists:append([climb], [grasp], _3048) ? creep
   Exit: (11) lists:append([climb], [grasp], [climb, grasp]) ? creep
   Exit: (10) canget(state(middle, onfloor, middle, hasnot), [climb, grasp], [state(middle, onfloor, middle, hasnot), state(middle, onbox, middle, hasnot)])
? creep
   Call: (10) lists:append([push(atwindow, middle)], [climb, grasp], _3054) ? creep
   Exit: (10) lists:append([push(atwindow, middle)], [climb, grasp], [push(atwindow, middle), climb, grasp]) ? creep
   Exit: (9) canget(state(atwindow, onfloor, atwindow, hasnot), [push(atwindow, middle), climb, grasp], [state(atwindow, onfloor, atwindow, hasnot), state(mi
ddle, onfloor, middle, hasnot), state(middle, onbox, middle, hasnot)]) ? creep
   Call: (9) lists:append([walk(atdoor, atwindow)], [push(atwindow, middle), climb, grasp], _2574) ? creep
   Exit: (9) lists:append([walk(atdoor, atwindow)], [push(atwindow, middle), climb, grasp], [walk(atdoor, atwindow), push(atwindow, middle), climb, grasp]) ?
 creep
   Exit: (8) canget(state(atdoor, onfloor, atwindow, hasnot), [walk(atdoor, atwindow), push(atwindow, middle), climb, grasp], [state(atdoor, onfloor, atwindo
w, hasnot), state(atwindow, onfloor, atwindow, hasnot), state(middle, onfloor, middle, hasnot), state(middle, onbox, middle, hasnot)]) ? creep
Sequence = [walk(atdoor, atwindow), push(atwindow, middle), climb, grasp],
Path = [state(atdoor, onfloor, atwindow, hasnot), state(atwindow, onfloor, atwindow, hasnot), state(middle, onfloor, middle, hasnot), state(middle, onbox, mi
ddle, hasnot)] .
```

## Postlab:

1. Which are the actions that the monkey must perform to eat bananas?

2. Draw and explain the Monkey search tree for the banana eating problem.