Name: Kartik Jolapara **SAPID:** 60004200107

DIV: B/B1

AI

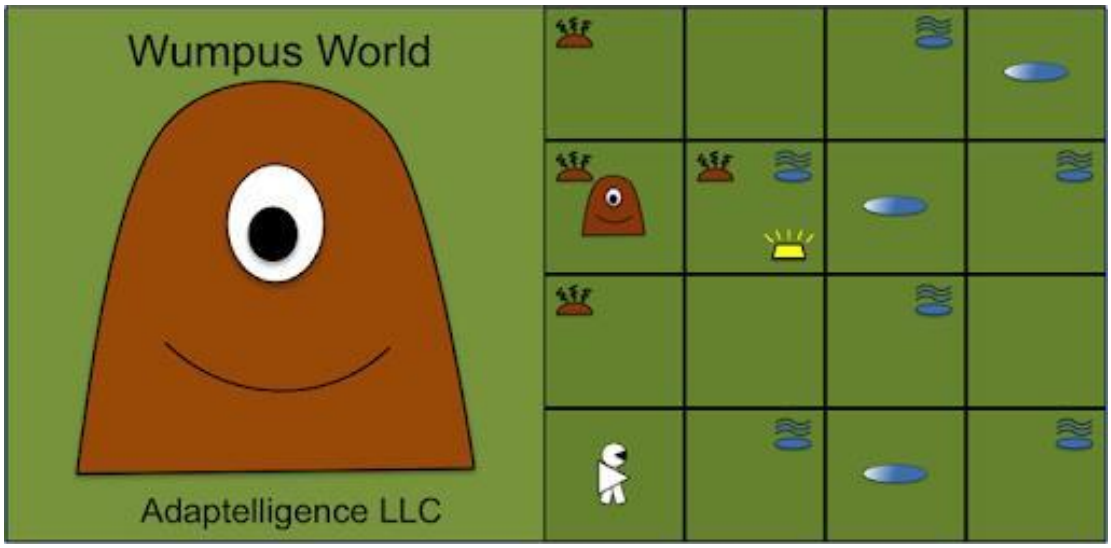
Exp7

Aim: Implementation of Wumpus World Program in Prolog.

Theory:

The Wumpus World's agent is an example of a knowledge-based agent that represents Knowledge representation, reasoning, and planning. Knowledge Based agent links general knowledge with current precepts to infer hidden characters of the current state before selecting actions. Its necessity is vital in partially observable environments.

Wumpus world is a cave with 16 rooms (4×4). Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from Room [1, 1]. The cave has – some pits, a treasure, and a beast named Wumpus. The Wumpus cannot move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or is eaten by the
Wumpus.
Some elements support the agent to explore the cave, like -The Wumpus's adjacent rooms are stench. -The agent is given one arrow which it can use to kill the Wumpus when facing it (Wumpus screams when it is killed). – The adjacent rooms of the room with pits are filled with breeze. -The treasure room is always glittery.



PEAS for Wumpus World:

PEAS represents Performance Measures, Environment, Actuators, and Sensors. The PEAS description helps in grouping the agents. PEAS Description for the Wumpus World problem:

1. Performance measures:
    1. Agent gets the gold and returns back safe = +1000 points
    2. Agent dies = -1000 points
    3. Each move of the agent = -1 point
    4. Agent uses the arrow = -10 points 2. Environment:
    1. A cave with 16(4×4) rooms
    2. Rooms adjacent (not diagonally) to the Wumpus are stinking
    3. Rooms adjacent (not diagonally) to the pit are breezy
    4. The room with the gold glitters
    5. Agent's initial position – Room [1, 1] and facing right side
    6. Location of Wumpus, gold, and 3 pits can be anywhere, except in Room
    [1, 1].
3. Actuators:
    1. Devices that allow the agent to perform the following actions in the environment.
    2. Move forward
    3. Turn right
    4. Turn left
    5. Shoot 6. Grab-Release
4. Sensors:

1. Devices that help the agent in sensing the following from the environment.
2. Breeze
3. Stench
4. Glitter
5. Scream (When the Wumpus is killed)
6. Bump (when the agent hits a wall)

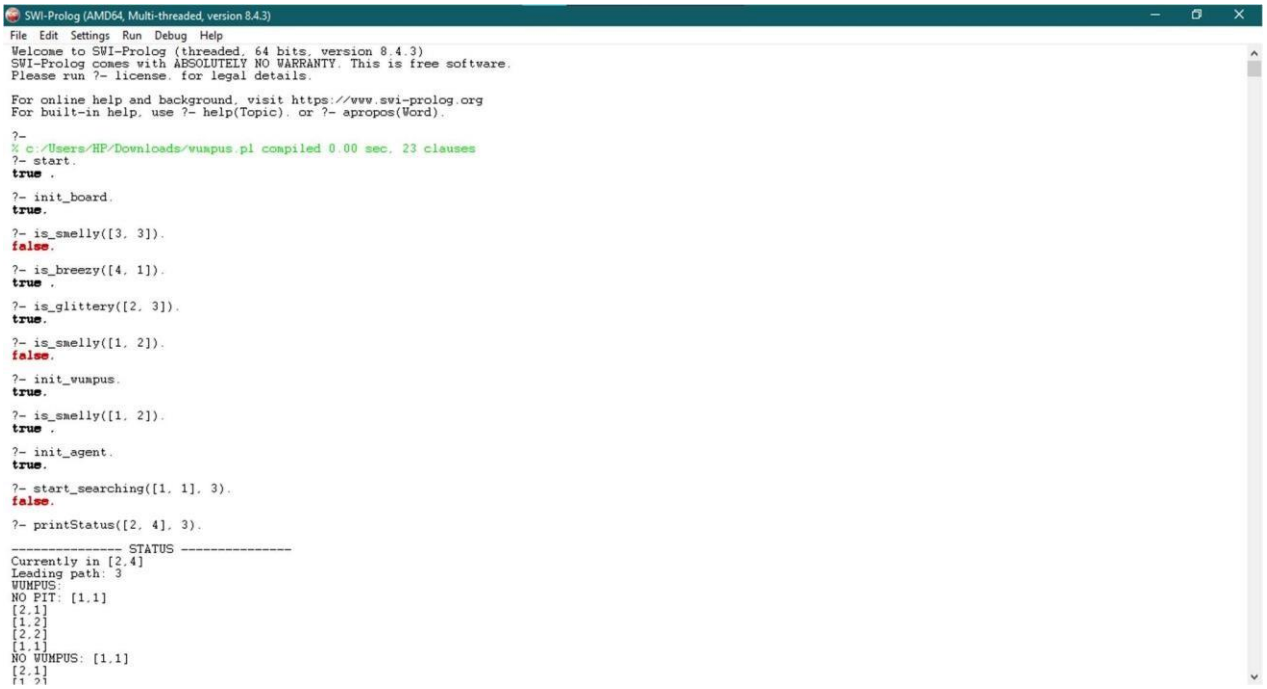
Code:

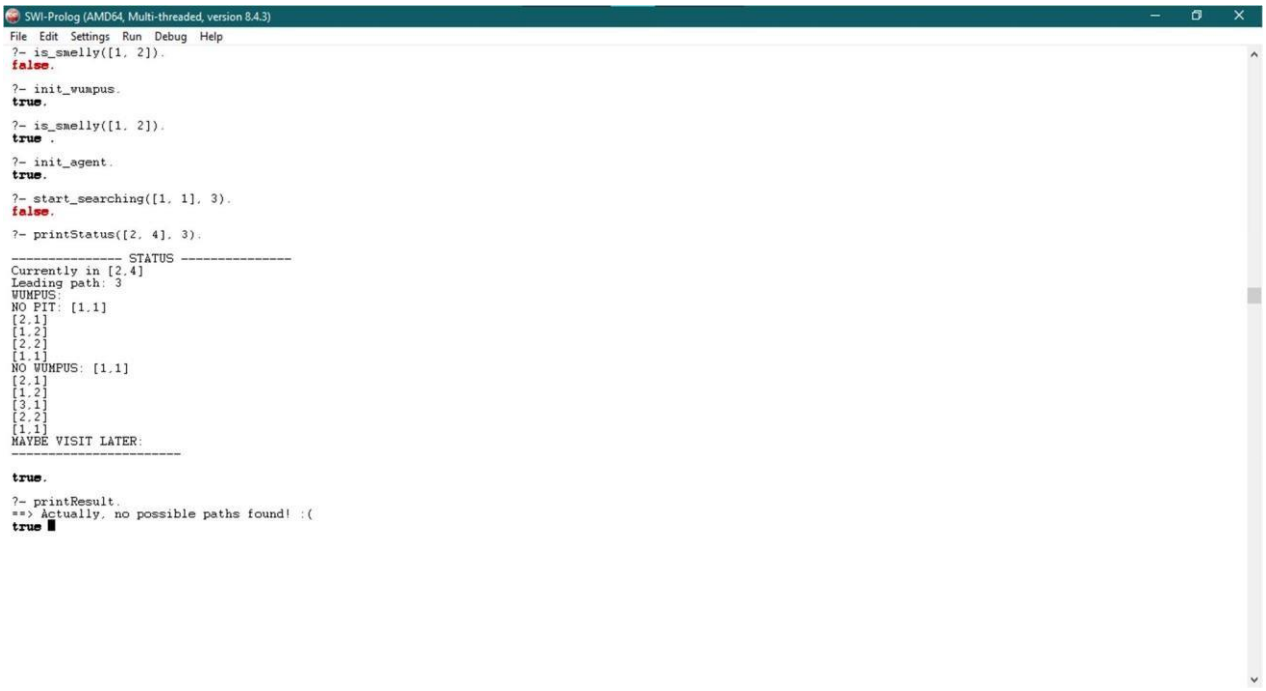
```
dynamic([
world_size/1,
position/2,
wumpus/1, noPit/1,
noWumpus/1, maybeVisitLater/2,
goldPath/1
]).
start:- retractall(wumpus(_)), retractall(noPit(_)),
retractall(noWumpus(_)),
retractall(maybeVisitLater(_,_)),
retractall(goldPath(_)), init_board, init_agent,
init_wumpus, start_searching([1, 1], []),
maybeVisitLater(PausedCell, LeadingPath),
retract(maybeVisitLater(PausedCell, _)),
start_searching(PausedCell, LeadingPath).
init_board:- retractall(world_size(_)),
assert(world_size([5, 5])),
retractall(position(_, _)),
assert(position(gold, [2, 3])),
assert(position(pit, [3, 1])),
assert(position(pit, [5, 1])),
assert(position(pit, [3, 3])),
assert(position(pit, [4, 4])),
assert(position(pit, [2, 5])), assert(noPit([1,
1])).
init_agent:- assert(position(agent, [1, 1])). init_wumpus:-
assert(position(wumpus, [1, 3])), assert(noWumpus([1, 1])).
valid_position([X, Y]):- X>0, Y>0, world_size([P, Q]), X@=<P, Y@=<Q. adjacent([X,
Y], Z):- Left is X-1, valid_position([Left, Y]), Z=[Left, Y]. adjacent([X, Y], Z):- Right is
X+1, valid_position([Right, Y]), Z=[Right, Y]. adjacent([X, Y], Z):- Above is Y+1,
```

```
valid_position([X, Above]), Z=[X, Above]. adjacent([X, Y], Z):- Below is Y-1,
valid_position([X, Below]), Z=[X, Below].
is_smelly([X, Y]):-   position(wumpus, Z), \+
noWumpus(Z),   adjacent([X, Y], Z). is_breezy([X, Y]):-
adjacent([X, Y], Z), position(pit, Z). is_glittery([X, Y]):-
position(gold, Z), Z==[X, Y].
moreThanOneWumpus:-   wumpus(X),
wumpus(Y), X\=Y.
killWumpusIfPossible(AgentCell):-
  wumpus([Xw, Yw]), \+ moreThanOneWumpus,
  AgentCell=[Xa, Ya],   (Xw==Xa; Yw==Ya),   assert(noWumpus([Xw, Yw])),
format('~nAgent confirmed Wumpus cell to be ~w and shot an arrow from cell  ~w.~nThe
WUMPUS has been killed!~n', [[Xw, Yw], AgentCell]),   retractall(wumpus(_)).
start_searching(Cell,      LeadingPath):-   is_glittery(Cell),
append(LeadingPath, [Cell], CurrentPath),
  \+ goldPath(CurrentPath), assert(goldPath(CurrentPath)).
start_searching(Cell, _):-
is_breezy(Cell).
start_searching(Cell, _):-   \+
is_breezy(Cell),
adjacent(Cell, X),
  \+ noPit(X), assert(noPit(X)).  start_searching(Cell,
            _):-
is_smelly(Cell),   adjacent(Cell,
X),
  \+ noWumpus(X), assert(wumpus(X)).
start_searching(Cell, _):-   \+
is_smelly(Cell),   adjacent(Cell,
X),
  \+ noWumpus(X), assert(noWumpus(X)),   wumpus(Y),
X==Y, retract(wumpus(Y)). start_searching(CurrentCell,
LeadingPath):-        (killWumpusIfPossible(CurrentCell);
format('')),          append(LeadingPath,    [CurrentCell],
CurrentPath),
  \+ is_glittery(CurrentCell),   adjacent(CurrentCell, X), \+
member(X, LeadingPath),
  (( noWumpus(X), noPit(X)) -> write('');
    (\+ maybeVisitLater(CurrentCell, _) -> assert(maybeVisitLater(CurrentCell, LeadingPath)); write(''))
  ),
  noWumpus(X), noPit(X),   start_searching(X, CurrentPath).
printResult:-
  \+ goldPath(_), write("==> Actually, no possible paths found! :("). printResult:-
goldPath(_), !, format('The following paths to the Gold are found: ~n'),   forall(goldPath(X),
writeln(X)).  printStatus(Cell,  LeadingPath):-      format('~n---------- STATUS ----------
```

~nCurrently in ~w~nLeading path: ~w~n', [Cell, LeadingPath]), write('WUMPUS: '), forall(wumpus(X), writeln(X)),nl, write('NO PIT: '), forall(noPit(Y), writeln(Y)), write('NO WUMPUS: '), forall(noWumpus(Z), writeln(Z)), write('MAYBE VISIT LATER: '), forall(maybeVisitLater(M, _), writeln(M)), format('~n----------~n~n').

Output:





Conclusion:

Thus, we successfully implemented Wumpus World in SWI-Prolog.