Swi prolog code for wumpus world:

```
/* ---------------------------
   Wumpus World in SWI-Prolog
   --------------------------- */

% --- World Representation ---
% world(Size, Wumpus, PitList, Gold)
world(4, (2,3), [(3,1),(4,4)], (2,2)).

% --- Adjacency Helper ---
adjacent((X,Y),(X1,Y)) :- X1 is X+1.
adjacent((X,Y),(X1,Y)) :- X1 is X-1, X1 > 0.
adjacent((X,Y),(X,Y1)) :- Y1 is Y+1.
adjacent((X,Y),(X,Y1)) :- Y1 is Y-1, Y1 > 0.

% --- Percepts ---
stench(Pos) :-
    world(_, Wumpus, _, _),
    adjacent(Pos,Wumpus).

breeze(Pos) :-
    world(_, _, Pits, _),
    member(Pit,Pits),
    adjacent(Pos,Pit).

glitter(Pos) :-
    world(_, _, _, Gold),
    Pos = Gold.
% --- Safety ---
safe(Pos) :-
```

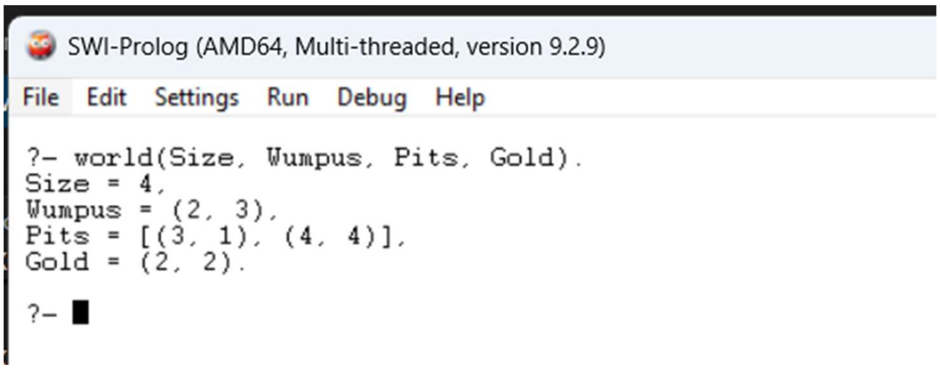
```
  \+ stench(Pos),
  \+ breeze(Pos).

% --- Moves (stay within world bounds) ---
move((X,Y), right, (X1,Y)) :-
  world(Size, _, _, _),
  X1 is X+1,
  X1 =< Size.
move((X,Y), left,  (X1,Y)) :-
  X1 is X-1,
  X1 > 0.
move((X,Y), up,    (X,Y1)) :-
  world(Size, _, _, _),
  Y1 is Y+1,
  Y1 =< Size.
move((X,Y), down,  (X,Y1)) :-
  Y1 is Y-1,
  Y1 > 0.

% --- Safe or goal (allow moving to gold even if near danger) ---
safe_or_goal(Pos) :-
  safe(Pos);
  glitter(Pos).

% --- Planning with visited cells to avoid cycles ---
plan(Start, Plan) :-
  plan(Start, [Start], Plan).

% If gold is here, grab it
plan(Pos, _, [grab]) :-
  glitter(Pos), !.
```

```
% Explore safe neighbors
plan(Pos, Visited, [Move|Rest]) :-
  move(Pos, Move, Next),
  safe_or_goal(Next),
  \+ member(Next, Visited),   % avoid revisiting
  plan(Next, [Next|Visited], Rest).
```

Output:

1. Check the World Setup

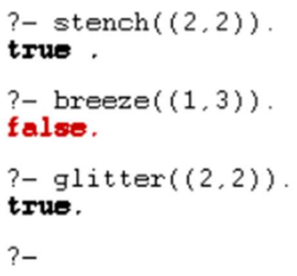
We can inspect the world:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help

```
?- world(Size, Wumpus, Pits, Gold).
Size = 4,
Wumpus = (2, 3),
Pits = [(3, 1), (4, 4)],
Gold = (2, 2).

?- 
```

2. Test Percepts

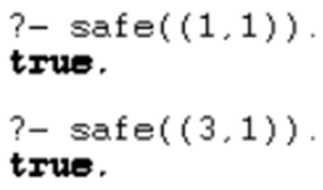
These tell you what the agent would sense in a cell.



```
?- stench((2,2)).
true .

?- breeze((1,3)).
false.

?- glitter((2,2)).
true.

?-
```

3. Check Safety

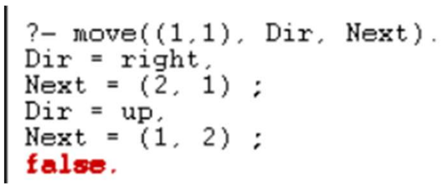
See whether a cell is safe to enter:



```
?- safe((1,1)).
true.

?- safe((3,1)).
true.
```

4. Generate Possible Moves

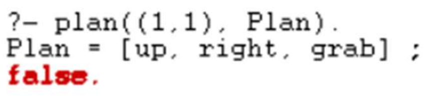
To see where we can go from a position:



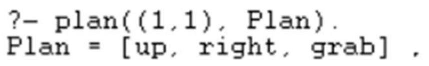
```
?- move((1,1), Dir, Next).
Dir = right,
Next = (2, 1) ;
Dir = up,
Next = (1, 2) ;
false.
```

5. Plan a Route to the Gold

The core feature — find a plan starting from a position:



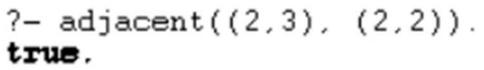
```
?- plan((1,1), Plan).
Plan = [up, right, grab] ;
false.
```



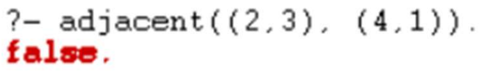
```
?- plan((1,1), Plan).
Plan = [up, right, grab] .
```

6. Test Adjacency

To see if two positions are neighbors:



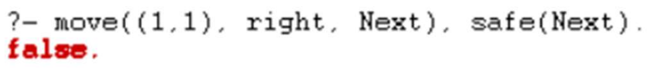
```
?- adjacent((2,3), (2,2)).
true.
```



```
?- adjacent((2,3), (4,1)).
false.
```

7. Manual Exploration

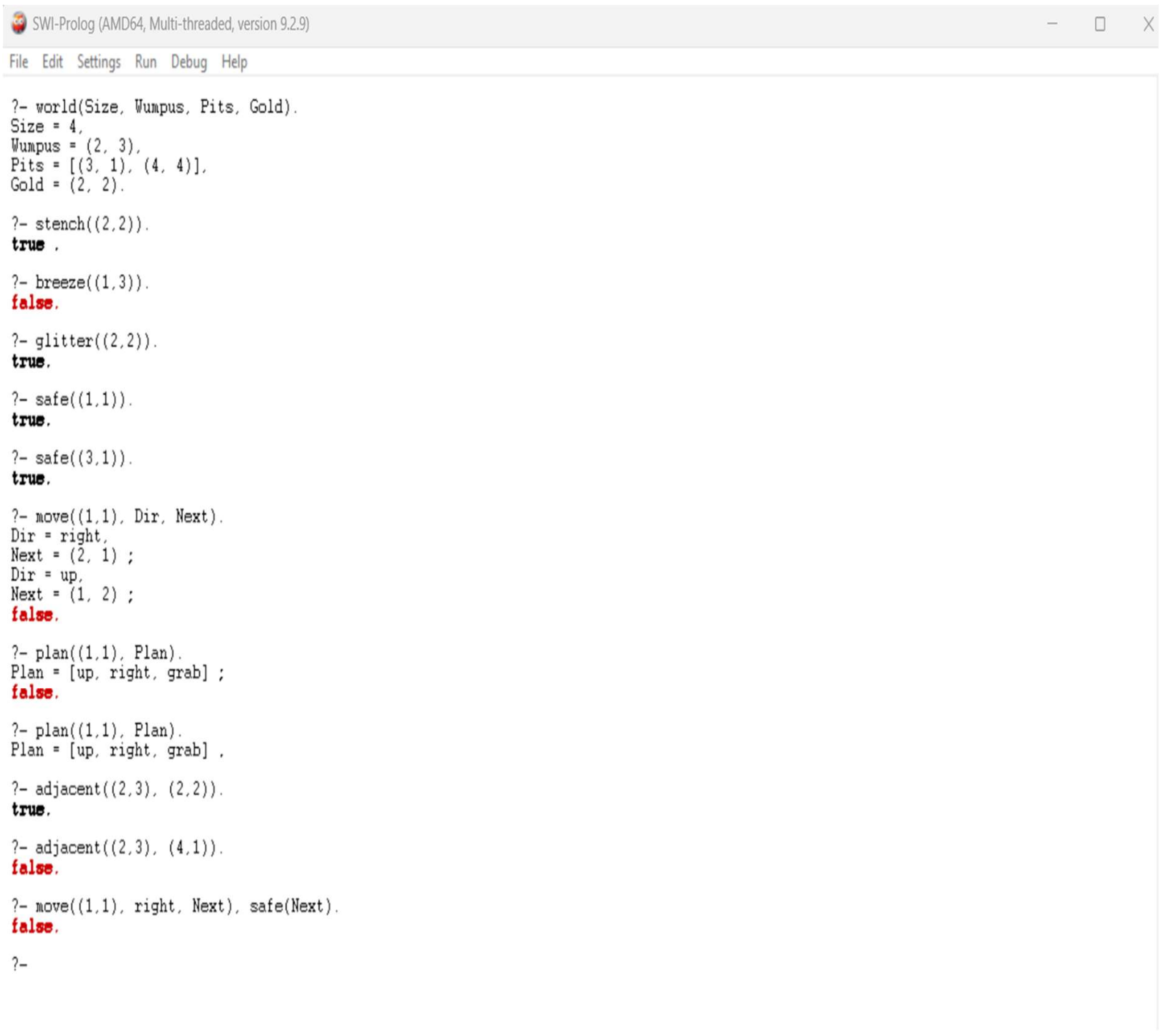
We can combine rules manually:



```
?- move((1,1), right, Next), safe(Next).
false.
```

This will tell "if the right cell is safe to move to."

Overall Output:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help

```
?- world(Size, Wumpus, Pits, Gold).
Size = 4,
Wumpus = (2, 3),
Pits = [(3, 1), (4, 4)],
Gold = (2, 2).

?- stench((2,2)).
true .

?- breeze((1,3)).
false.

?- glitter((2,2)).
true.

?- safe((1,1)).
true.

?- safe((3,1)).
true.

?- move((1,1), Dir, Next).
Dir = right,
Next = (2, 1) ;
Dir = up,
Next = (1, 2) ;
false.

?- plan((1,1), Plan).
Plan = [up, right, grab] ;
false.

?- plan((1,1), Plan).
Plan = [up, right, grab] .

?- adjacent((2,3), (2,2)).
true.

?- adjacent((2,3), (4,1)).
false.

?- move((1,1), right, Next), safe(Next).
false.

?-
```