PROJECT MANUAL

Project Title: Wumpus World Game Logical Agent

Aim:

To simulate the Wumpus World environment using a logical agent that interacts with a 4x4 grid, avoiding hazards (Wumpus and pits), collecting gold, and returning safely to the starting position. The implementation uses Prolog for logical reasoning and Tkinter for a graphical user interface.

Objective:

1. Develop an interactive GUI-based Wumpus World game.
2. Utilize Prolog to manage game logic, state, and percepts.
3. Enable player actions such as moving, shooting, grabbing gold, and climbing out.
4. Provide real-time feedback on percepts (breeze, stench, glitter) and game metrics (score, moves).

Facts:

1. **Grid Layout:** 4x4 grid with the agent starting at (1,1).
2. **Hazards:**
    - Wumpus at (4,4).
    - Pits at (1,4) and (3,1).
    - Gold at (3,3).
3. **Adjacency:** Squares are connected horizontally and vertically (e.g., (1,1) is adjacent to (1,2) and (2,1)).
4. **Percepts:**
    - breeze/1: True if the current square is adjacent to a pit.
    - stench/1: True if the current square is adjacent to the Wumpus.
    - glitter/1: True if the gold is in the current square.

Rules:

1. **Movement:**
    - The agent can move to adjacent squares.
    - Entering a pit or the Wumpus's square results in death.
2. **Shooting:**
    - The agent can shoot an arrow in one direction (up, down, left, right).
    - Killing the Wumpus removes all stench percepts.
3. **Gold Collection:**
    - Grabbing gold increases the score.
    - Climbing out at (1,1) with the gold wins the game.
4. **Scoring:**
    - Collecting gold: +1000 points.
    - Dying: -1000 points.
    - Each move: -1 point.
    - Shooting an arrow: -10 points.

Query:

```
:- dynamic([
   visited/1, breeze/1, stench/1, glitter/1,
   wumpus_location/1, pit_location/1, gold_location/1,
   agent_location/1, timer/1, score/1, wumpus_final_location/1
]).
% Adjacent squares (up, down, left, right)
adjacent([X,Y], [X1,Y]) :- X1 is X+1, X1 =< 4.  % Right
adjacent([X,Y], [X1,Y]) :- X1 is X-1, X1 >= 1.  % Left
adjacent([X,Y], [X,Y1]) :- Y1 is Y+1, Y1 =< 4.  % Up
adjacent([X,Y], [X,Y1]) :- Y1 is Y-1, Y1 >= 1.  % Down
% Border check
border([X,Y]) :- X < 1; X > 4; Y < 1; Y > 4.
% Update percepts when entering a square
makestatement([X,Y]) :-
   (pit_location(Pit), adjacent([X,Y], Pit)) ->
      (format('You feel a breeze in ~p!~n', [[X,Y]]), assert(breeze([X,Y]))) ; true,
   (wumpus_location(Wumpus), adjacent([X,Y], Wumpus)) ->
```

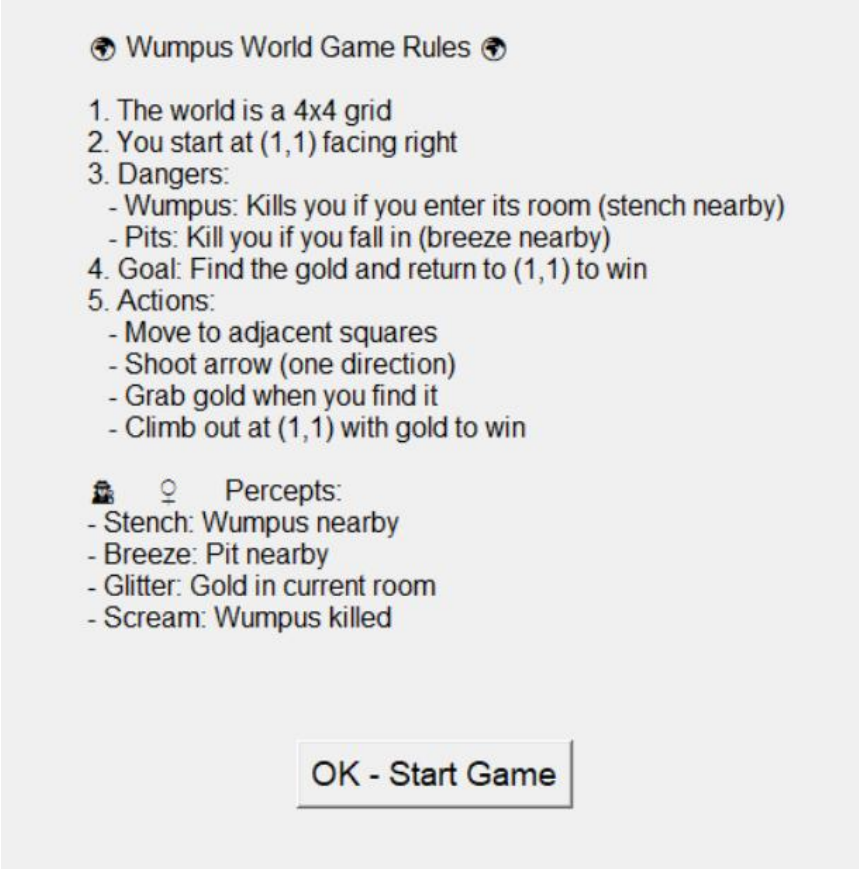
```
    (format('You smell a terrible stench in ~p!~n', [[X,Y]]), assert(stench([X,Y]))) ; true,
  (gold_location(Gold), [X,Y] == Gold) ->
    (format('** You found the GOLD! (+500 points) **~n'), assert(glitter([X,Y])),
    score(S), NewScore is S + 500, retract(score(_)), assert(score(NewScore))) ; true.
% Pit detection rules
pit([X,Y]) :-
  forall(adjacent([X,Y], L), (breeze(L); border(L))).
pit([X,Y]) :-
  adjacent([X,Y], L), visited(L), breeze(L),
  forall(adjacent(L, L2), (L2 == [X,Y]; psafe(L2); border(L2))).
% Wumpus detection rules
wumpus([X,Y]) :-
  forall(adjacent([X,Y], L), (stench(L); border(L))),
  retractall(wumpus_final_location(_)), assert(wumpus_final_location([X,Y])).
wumpus([X,Y]) :-
  adjacent([X,Y], L), visited(L), stench(L),
  forall(adjacent(L, L2), (L2 == [X,Y]; wsafe(L2); border(L2))),
  retractall(wumpus_final_location(_)), assert(wumpus_final_location([X,Y])).
% Safety checks
psafe([X,Y]) :- adjacent([X,Y], L), visited(L), \+ breeze(L).
wsafe([X,Y]) :- adjacent([X,Y], L), visited(L), \+ stench(L).
% Dangerous squares (pit or Wumpus)
fail_agent([X,Y]) :- pit([X,Y]); wumpus([X,Y]).
% Possible safe moves
maybe([X,Y]) :-
  \+ visited([X,Y]), \+ fail_agent([X,Y]),
  (adjacent([X,Y], L), (breeze(L); stench(L))).
safe([X,Y]) :- visited([X,Y]).
safe([X,Y]) :- psafe([X,Y]), wsafe([X,Y]).
good([X,Y]) :- safe([X,Y]), \+ visited([X,Y]).
% Find possible moves (without duplicates)
existgood(A) :-
  visited(V), adjacent(V, A), good(A), \+ visited(A).
existmaybe(A) :-
```

```
    visited(V), adjacent(V, A), maybe(A), \+ visited(A).
exist(X) :- existgood(X); existmaybe(X).
% Remove duplicate moves
unique_moves(Moves, Unique) :-
    sort(Moves, Unique).
% Game over if agent steps into pit/Wumpus
failure(X) :-
    (wumpus_location(W), X == W) ->
        format('You were eaten by the Wumpus! GAME OVER.~n'), halt;
    (pit_location(P), X == P) ->
        format('You fell into a pit! GAME OVER.~n'), halt;
    true.
% Main game loop
acte(X) :-
    retractall(agent_location(_)),
    assert(agent_location(X)),
    update_score(-1),
    update_timer(1),
    format('You are now in ~p.~n', [X]),
    failure(X),
    assert(visited(X)),
    makestatement(X),
    findall(A, exist(A), PossibleMoves),
    unique_moves(PossibleMoves, UniqueMoves),
    (UniqueMoves = [] ->
        format('No more moves left!~n'),
        false;
        format('Possible moves: ~p~n', [UniqueMoves]),
        format('Enter your next move as [X,Y]: '),
        read(NextMove),
        (member(NextMove, UniqueMoves) ->
            acte(NextMove);
            format('Invalid move! Try again.~n'),
            acte(X)
```

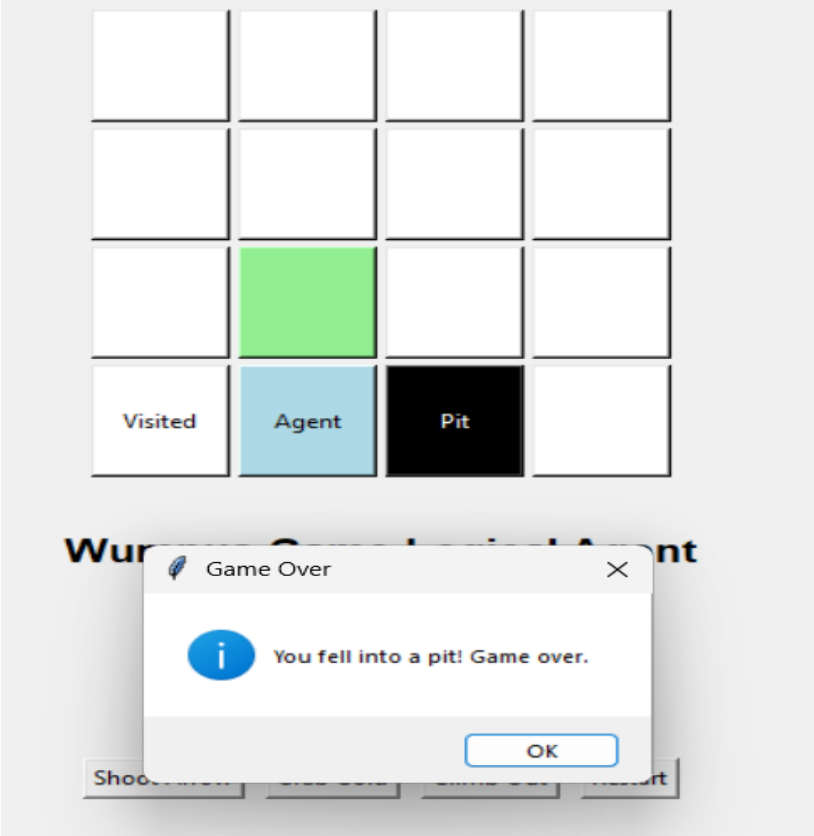
```
    )).
% Score and timer updates
update_score(X) :-
    score(S), NewS is S + X,
    retractall(score(_)), assert(score(NewS)).
update_timer(X) :-
    timer(T), NewT is T + X,
    retractall(timer(_)), assert(timer(NewT)).
% Initialize game
init :-
    retractall(visited(_)), retractall(breeze(_)), retractall(stench(_)), retractall(glitter(_)),
    retractall(wumpus_location(_)), retractall(pit_location(_)), retractall(gold_location(_)),
    retractall(agent_location(_)), retractall(timer(_)), retractall(score(_)),
retractall(wumpus_final_location(_)),
    assert(timer(0)), assert(score(30)),
    assert(gold_location([3,3])), assert(wumpus_location([4,4])),
    assert(pit_location([1,4])), assert(pit_location([3,1])),
    assert(agent_location([1,1])), assert(wumpus_final_location([-1,-1])).
% Start game with replay option
start :-
    init,
    format('~n=== WUMPUS WORLD GAME ===~n'),
    format('Find the gold and kill the Wumpus to win!~n'),
    agent_location(AL),
    \+ acte(AL),
    wumpus_final_location(Z),
    (Z = [-1,-1] ->
        format('~nYou failed to find the Wumpus!~n'),
        ask_replay;
        format('~nYou killed the Wumpus at ~p!~n', [Z]),
        score(S), timer(T),
        format('Final Score: ~p~n', [S]),
        format('Total Moves: ~p~n', [T]),
        format('*** YOU WIN! ***~n'),
```

```
        ask_replay
    ).
% Ask to replay or exit
ask_replay :-
    format('~nPlay again? (yes/no): '),
    read(Choice),
    (Choice == yes ->
        start;
        format('Thanks for playing!~n'),
        halt
    ).
% Run the game
:- initialization(start).
```

Output:




Visited
Agent
Wumpus Game Logical Agent
Percepts: Breeze
Wumpus: Alive
Score: 29
Moves: 1
Shoot Arrow
Grab Gold
Climb Out
Restart


Visited
Agent
Pit
Game Over
You fell into a pit! Game over.
OK