Wumpus World - Prolog Implementation Explained

- Alexandre Dietrich
- Patrick Chabelski
- Rodolfo Vasconcelos

SCS 3547 – Intelligent Agents & Reinforcement Learning

UNIVERSITY OF TORONTO - SCHOOL OF CONTINUING STUDIES

Instructor: Larry Simon

GitHub: https://github.com/ravasconcelos/wumpus_world/ (https://github.com/ravasconcelos/wumpus_world/)

About the Project

Richard O. Legendi has implemented in Prolog the Wumpus World described in Artificial Intelligence : A Modern Approach (Russel - Norvig)

- https://github.com/rlegendi/wumpus-prolog/ (https://github.com/rlegendi/wumpus-prolog/)
- https://www.amazon.ca/Artificial-Intelligence-Modern-Approach-3rd/dp/0136042597 (https://www.amazon.ca/Artificial-Intelligence-Modern-Approach-3rd/dp/0136042597)

This project is an explanation of how each rule in the program works.

The outcome of this project is this Swish Notebook.

Improvements

The improvements of this code can be found in this Notebook:

- https://swish.swi-prolog.org/p/Wumpus%20World%20-%20Improved.swinb (https://swish.swi-prolog.org/p/Wumpus%20World%20-%20Improved.swinb)

Deviations from Wumpus World Rules

The version of the Wumpus World code by Richard O. Legendi deviates from the typical Wumpus World rules in numerous ways. This was done to facilitate an easier version of the game to understand, though has led to a few complications during runtime. These differences (and any associated problems) are as follows:

- Movement:
  - WW Rules: The agent can only move forward, turn left or turn right on each turn, and cannot move diagonally.
  - Legendi: The agent can go to any of the 8 nearest squares in one direct move per turn.
- Perceptions:
  - WW Rules: The agent shall receive a maximum of 5 percepts per turn: “breezy” (if there a pit adjacent), “stench” (if there is a Wumpus adjacent), “glitter” (if the gold is in the current spot), “bump” (if the agent walks into a wall at the boundary), and “scream” (if the agent’s arrow hits the wumpus)
  - Legendi: The agent only receives 3 percepts: breezy, stench, and glitter

        - Each turn, the agent gets information from the KB about adjacent spaces. However, the implementation for KB checking of gold is not right, as it will check space [3,2] each time (instead of checking adjacent squares as it does for the Wumpus and Pit percepts)
            - KB learn Gold (https://raw.githubusercontent.com/ravasconcelos/wumpus_world/master/images/check_gold.png)
            - This can be solved by updating the `add_gold_KB(no)` predicate to the following (which is the same setup as the Wumpus and Pit checks)
            - KB learn Gold - Fixed (https://raw.githubusercontent.com/ravasconcelos/wumpus_world/master/images/fix_gold.png)
        - Each turn, the agent gets information from the KB about adjacent spaces. The code is not very efficient, as it checks spots that are outside of the world. For instance, an agent in location [1,1] will still check [1,0] and [0,1]. There will never be anything in these spots and the agent cannot move to these spots; this is simply a parasitic element of code inefficiency.
- Boundaries:
    - WW Rules: The agent receives a “bump” percept if they attempt to move forward into a wall, when on the boundaries of the world grid
    - Legendi: The agent’s action-taking process allows it to move to any nearby square that has not yet been visited; this, coupled with the inability to turn/move forward automatically inhibits the ability to walk into a wall
- Actions:
    - WW Rules: The agent can fire one arrow, grab the gold (if in the same room) as part of its action in a turn, or climb out (if it is back at the entrance)
    - Legendi: The agent cannot fire an arrow (therefore cannot kill the wumpus), and will automatically obtain the gold once it is detected in the same room as the agent. There is no climb action.
        - As the agent is not allowed to rotate its orientation, the ability to “aim” its arrow is not possible with this implementation
        - Additionally, the fact that the wumpus cannot be killed means that there is no “scream” percept available for the agent
- Ending the Game:
    - WW Rules: The game will end once the agent finds the gold and climbs out of the cave entrance (located at the starting position [1,1]), or if the agent dies via Wumpus or falling into a pit
    - Legendi: The game will end once the agent detects that it is on the same spot as the gold. The agent will still die if it walks into the wumpus or into a pit.
- Object initialization:
    - WW Rules: The locations of the Wumpus, Pits, and Gold are randomized each session.
    - Legendi: The locations of the Wumpus, Pits, and Gold are initialized per Figure 7.2 in the Russell & Norvig textbook.
        - Figure 7.2 specifies that the Wumpus should be initialized in [3,1]. However, this is hard coded as [4,1] in the Legendi code. Reverting this value back to [3,1] results in the agent losing each game; seemingly the author has placed the Wumpus in [4,1] to facilitate completion of the game

General Code Overview

1. start
    1. init
        1. init_game
        2. init_land_72 See Fig. 7.2 (https://raw.githubusercontent.com/ravasconcelos/wumpus_world/master/figure_7_2.png)
        3. init_agent
        4. init_wumpus
    2. `take_steps(VisitedLast)`

   1. `make_percept_sentence([Stench,Breeze,Glitter])`
      1. `breezy(Yes/No)`
         1. `isBreezy(Yes/No)`
         2. (adj check)
      2. `smelly(Yes/No)`
   2. `isSmelly(Yes/No)`
      1. (adj check)
      2. `glittery(Yes/No)`
   3. `isGlittery(Yes/No)`
      1. (adj/condition check)
3. `agent_location(AL)`
4. `update_KB([Stench,Breeze,Glitter])`
   1. `addWumpus_KB(No)`
      1. assumeWumpus
   2. `add_pit_KB(Yes/No)`
      1. assumePit
   3. `add_gold_KB(Yes/No)`
      1. assumeGold
5. `ask_KB(VisitedLast, Action)`
   1. permitted
   2. not_member
6. update_time
7. update_score
   1. `agent_location(Aloc)` ,
   2. VL = [Aloc|VisitedList],
   3. standing,
   4. `step_pre(VL)` .
      1. TakeSteps(VL) -> LOOP BACK TO 2. UNTIL WIN/LOSE

Run the application

```
?- start.
```

Header and Copyright

```
%------------------------------------------------------------------------------
% A Prolog Implementation of the Wumpus World described in
% Artificial Intelligence : A Modern Approach (Russel - Norvig)
%
% Mandatory Excercise 2007
% v1.0 - Jan. 31, 2007
% Richard O. Legendi
%
% Copied into prolog-examples with permission Richard O. Legendi
% Original exercise descriped in  Artificial Intelligence : A Modern Approach (Russel
%
% Usage:
```

```
% consult this file
% ?-start.
%
%------------------------------------------------------------------------------
```

Dynamic methods

Define dynamic predicates with associated arities; these are used as methods throughout the code (called numerous times; data within the structures are purged and updated accordingly)

- agent_location: Rule defining agent location
- gold_location: Rule defining gold location
- pit_location: Rule defining pit location
- time_taken: keep track of the time taken
- score: keep track of the score
- visited: Not used. Can be safely removed from here and init_game.
- visited_cells: List of visited cells
- world_size: Dynamically define the size of the world. Set 4x4 in init_land_fig72
- wumpus_location: Rule defining wumpus location
- isPit: 2 data points -> location and Yes/No for if pit is in spot
- isWumpus: 2 data points -> location and Yes/No for if wumpus is in spot
- isGold: 2 data points -> location and Yes/No for if gold is in spot

```
%------------------------------------------------------------------------------
% Declaring dynamic methods

:- dynamic ([
         agent_location/1,
         gold_location/1,
         pit_location/1,
         time_taken/1,
         score/1,
         visited/1,
         visited_cells/1,
         world_size/1,
         wumpus_location/1,
         isPit/2,
         isWumpus/2,
         isGold/2
        ]).
```

Start the Game

- start: this predicate / rule starts the initializes the simulation (start if init and `take_steps(([1,1]))`
- init: initialize all the dynamic methods
- take_steps: Put agent in spot 1,1 and start the game

```
%------------------------------------------------------------------------------
% To start the game

start :-
    format('Initializing started...~n', []),
    init,
    format('Let the game begin!~n', []),
    take_steps([[1,1]]).
```

Scheduling simulation

- step_pre: This is the recursive element of the code; will call take_steps using the updated VisitedList, until the game is won
  - AL=GL: Agent Location is the same as Gold Location, therefore you won - print out the score and time, the code will not call the recursive element and will stop here.
  - AL=WL: agent is in the same spot as the wumpus; you lost - print score and time, code will not call recursive element and will stop here
  - take_steps: If neither WIN or LOSE conditions are met, continue (recursively)
- take-steps: Number of steps involved in viewing percepts, updating the knowledge base, deciding on an action, updating time/score/board state accordingly. VisitedLast is an [[X,Y]] coordinate.
  - make_percept_sentence: check where you are and what is around you. Returns [yes/no,yes/no,yes/no] list corresponding to the 3 percepts the agent can sense (smell, breeze, glitter)
  - agent_location: rule for the agent location (update it)
  - format: print out agent location & percept sentence
  - update_KB: updates the knowledge base with the current perception. The KB are composed by three arrays: isWumpus, isPit and isGold. These arrays are updated with locations adjacent to the current location of the agent with information of yes or no (denotes the presence or absence of Wumpus, a pit or gold).
  - ask_KB: based on where you are and what is around, decide what to do
  - update_time: for each turn, add 1 to the time counter
  - update_score: update the score each turn based on wumpus location, agent location, gold location
  - agent_location and VL: Update the visited list based on the agent location
  - standing: This checks to see if the agent is on a wumpus spot, gold spot, or pit spot
  - step_pre: This is a call to a predicate that begins a recursive chain (to continue the game until you win or lose). VL is the visited list updated with the current agent location.

```
%------------------------------------------------------------------------------
% Scheduling simulation:

step_pre(VisitedList) :-
    agent_location(AL),
    gold_location(GL),
    wumpus_location(WL),
    score(S),
    time_taken(T),

    ( AL=GL -> writeln('WON!'), format('Score: ~p,~n Time: ~p', [S,T])
    ; AL=WL -> format('Lost: Wumpus eats you!~n', []),
               format('Score: ~p,~n Time: ~p', [S,T])
    ; take_steps(VisitedList)
    ).

take_steps(VisitedList) :-
    make_percept_sentence(Perception),
    agent_location(AL),
    format('I\'m in ~p, seeing: ~p~n', [AL,Perception]),

    update_KB(Perception),
    ask_KB(VisitedList, Action),
    format('I\'m going to: ~p~n', [Action]),

    update_time,
    update_score
```

Updating states

- update_time:
  - Initialized as T = 0, it will be updated each iteration.
  - NewTime: set the NewTime variable
  - assert: this line updates time_taken to the NewTime
- update_score:
  - `update_score(AL, GL, WL)` : update the score based on the 3 locations confirmed here
- `update_score(P)` : Each turn the score will decrease by 1 (very similar to time updating)
- `update_score(AL, AL, _)` : this is the criteria for winning (gold and agent locations are the same)
- `update_score(_,_,_)` : when the locations of the wumpus, agent, and gold are still unique, the game continues & your score decreases by 1
- update_agent_location: update the the agent location with the location defined after asking the KB for a valid location
- is_pit: Check to see if the pit location will be the same as the agent location at every call of the "standing" predicate

```
%------------------------------------------------------------------------------
% Updating states

update_time :-
    time_taken(T),
    NewTime is T+1,
    retractall( time_taken(_) ),
    assert( time_taken(NewTime) ).

update_score :-
    agent_location(AL),
    gold_location(GL),
    wumpus_location(WL),
    update_score(AL, GL, WL).

update_score(P) :-
    score(S),
    NewScore is S+P,
    retractall( score(_) ),
    assert( score(NewScore) ).

update_score(AL, AL, _) :-
    update_score(1000).

update_score(_,_,_) :-
    update_score(-1).
```

Display standings

- standing: this updates the user every turn about the agent status
    - (`is_pit(yes, AL)` : if pit location and agent location are the same, let the player know the agent has fallen and the game is over
- `stnd(_, _, _)` : Agent is not located in the gold or Wumpus location. If the locations of the agent, gold, and wumpus are unique, then you can still do something (game is not done). This rule could have been tested after the rules for location of wumpus and gold.
- `stnd(AL, _, AL)` : If the locations of the agent and wumpus are the same, you've been eaten and the game is over
- `stnd(AL, AL, _)` : If the locations of the agent and gold are the same, you found the gold and the game is won

```
%------------------------------------------------------------------------------
% Display standings

standing :-
    wumpus_location(WL),
    gold_location(GL),
    agent_location(AL),

    ( is_pit(yes, AL) -> format('Agent was fallen into a pit!~n', []),
      fail
    ; stnd(AL, GL, WL)
      %\+ pit_location(yes, Al),
    ).

stnd(_, _, _) :-
    format('There\'s still something to do...~n', []).

stnd(AL, _, AL) :-
    format('YIKES! You\'re eaten by the wumpus!', []),
    fail.

stnd(AL, AL, _) :-
    format('AGENT FOUND THE GOLD!!', []),
    true.
```

Perception

- make_percept_sentence: this will return a list of [yes/no,yes/no,yes/no] corresponding to whether or not the agent can detect the percepts in adjacent spaces
- make_perception and test_perception: Not used. They can safely be removed from the code.

```
%------------------------------------------------------------------------------
% Perceptotion

make_perception([_Stench,_Bleeze,_Glitter]) :-
    agent_location(AL),
    isStinky(AL),
    isBleezie(AL),
    isGlittering(AL).

test_perception :-
    make_percept_sentence(Percept),
    format('I feel ~p, ',[Percept]).

make_percept_sentence([Stench,Bleeze,Glitter]) :-
    smelly(Stench),
    bleezy(Bleeze),
    glittering(Glitter).
```

Initializing

- init: call the following 4 predicates as part of initialization
- init_game: predicate is true if the following rules hold (which they will per initialization)
  - `retractall(time_taken(_))` : remove blank values in time_taken and update the structure to have time = 0 (initialization)
  - `retractall(score(_))` : remove blank score value, update the counter to be zero (initial score once the game starts)
  - retractall/assert (`visited()`): Not used. Can be safely removed.
- init_land_fig72: Create the Wumpus World See Fig. 7.2 (https://raw.githubusercontent.com/ravasconcelos/wumpus_world/master/figure_7_2.png)
  - clear and initialize world_size to 4x4. world_size is defined as a dynamic methods.
  - initialize gold location (3,2).
  - initialize the 3 pit locations (4,4), (3,3), (1,3).
- init_agent: initialize agent location (1,1).
  - `visit([1,1])` : the agent is technically visiting [1,1] to start
- init_wumpus: initialize the wumpus location (4,1).
- `visit(Xs)` : this predicate sets the previously visited coordinates as Ys, blanks out the current visited_cells structure, and then appends the previous lis (Ys) to the new target-visit coordinate (Xs); running the code with printing,

```
%------------------------------------------------------------------------------
% Initializing

init :-
    init_game,
    init_land_fig72,
    init_agent,
    init_wumpus.

init_game :-
    retractall( time_taken(_) ),
    assert( time_taken(0) ),

    retractall( score(_) ),
    assert( score(0) ),

    retractall( visited(_) ),
    assert( visited(1) ),

    retractall( isWumpus(_,_) ),
    retractall( isGold(_,_) ),

    retractall( visited_cells(_) ),
    assert( visited_cells([]) ).

% To set the situation described in Russel-Norvig's book (2nd Ed.),
```

Perceptors

- adj: set up the adjacency rules (x,y coordinates that will equate to adjacency)
- adjacent: the adjacent function will take two coordinates (Agent and Pit/Wumpus/Gold) and hold true if either x or y coordinates are the same (indicating agent and object are in same row OR column) AND if the other coordinates are adjacent, per the adjacent rules set previously in the KB
- bleezy: Ls1 is the current agent location, pit location is Ls2; check if they are adjacent to each other; if they are, the predicate holds true, otherwise it does not, resulting in `breezy(no)`
- smelly: Check if Ls1 and Ls2 are adjacent to each other; if they are, the predicate holds true, otherwise it does not, resulting in `smelly(no)`
- glittering: Based on the current agent location (AL), check if it is glittering then the agent and the gold are in the same location and the predicate holds true returning yes. If it is not, the predicate fails returning no.

```
%------------------------------------------------------------------------------
% Perceptors

%%% Institiation error!!!

%adj(X,Y) :-
%    world_size(WS),
%    ( X is Y+1, Y   < WS
%    ; X is Y-1, Y-1 > 0
%    ).

adj(1,2).
adj(2,1).
adj(2,3).
adj(3,2).
adj(3,4).
adj(4,3).

adjacent( [X1, Y1], [X2, Y2] ) :-
    ( X1 = X2, adj( Y1, Y2 )
    ; Y1 = Y2, adj( X1, X2 )
    ).

%adjacent([X1,Y],[X2,Y]) :-
%    adj(X1,X2).

```

Knowledge Base

- update_KB: According to the perception list [Stench, Bleeze, Glitter], with values of "yes" or "no", create elements for four adjacent agent locations with the respectively values. This predicates updates the knowledge base of board positions with information where the Wumpus, Pits and Gold are as the agent moves and the it is repeatedly called
- add_wumpus_KB: Get the agent location and for the four adjacent agent locations, create an array element [x, y] in the dynamic predicate isWumpus with "no".
    - check if the wumpus is adjacent to 4 spots of the agent. For the four adjacent agent locations, create an array element [x, y] in the dynamic predicate isWumpus with "no".
- `add_pit_KB(no)` : Get the agent location and for the four adjacent agent locations, create an array element [x, y] in the dynamic predicate isPit with "no".
    - check if the pit is adjacent to 4 spots of the agent
- `add_pit_KB(yes)` : Get the agent location and for the four adjacent agent locations, create an array element [x, y] in the dynamic predicate isPit with "yes".
- `add_gold_KB(no)` : Get the gold location and call the predicate to update with "no" the lcoation of gold.
- `add_gold_KB(yes)` : Get the gold and agent locations. If the lcoations are the same, call predicate to update with "yes" and the location the dynamic predicate isGold.
- `assume_wumpus(no, L)` : For a specify location L, remove blank values in isWumpus and update the array element with "no". Print out the location where there is no Wumpus.
- `assume_wumpus(yes, L)` : For a specify location L, remove blank values in isWumpus and update the array element with "yes". Print out the location where the Wumpus is.
- `assume_gold(no, L)` : For a specify location L, remove blank values in isGold and update the array element with "no". Print out the location where there is no Gold.
- `assume_gold(yes, L)` : For a specify location L, remove blank values in isGold and update the array element with "yes". Print out the location where the Gold is.
- `permitted([X,Y])` : ensure new move does not breach world limits
- ask_KB: predicate that checks if there is no wumpus nearby, no pit nearby. Checks if the move (L, ie coordinates X,Y) is legal (does not go past world grid definition). Check to see if the move is legal (has not been selected already). Update the agent location accordingly to the new spot; the action is the location where there is no pit, wumpus, and is a permissible target.
    - isWumpus: no wumpus in target spot
    - isPit: no pit in target spot
    - permitted: spot is permissible (within world boundaries)
    - not_member: spot has not yet been selected
    - update_agent_location: update to the according location
    - Action = Lthe action taken will be this spot

```
%-----------------------------------------------------------------------------
% Knowledge Base:

update_KB( [Stench,Bleeze,Glitter] ) :-
    add_wumpus_KB(Stench),
    add_pit_KB(Bleeze),
    add_gold_KB(Glitter).

% if it would be 'yes' -> it would mean the player is eaten ;]
add_wumpus_KB(no) :-
    %agent_location(L1),
    %adjacent(L1, L2),
    %assume_wumpus(no, L2).
    agent_location([X,Y]),
    world_size(_),

    % Checking needed!!
    % adj will freeze for (4,_) !!

    Z1 is Y+1, assume_wumpus(no,[X,Z1]),
    Z2 is Y-1, assume_wumpus(no,[X,Z2]),
    Z3 is X+1, assume_wumpus(no,[Z3,Y]),
    Z4 is X-1, assume_wumpus(no,[Z4,Y]).

add_pit_KB(no) :-
    agent_location([X,Y]),
    Z1 is Y+1, assume_pit(no,[X,Z1]),
```

Utils

- not_member: check if the new move location [x,y] has already been visited (need to elaborate a bit here; dig into what the semicolon does / review lists and appending).

```
%-----------------------------------------------------------------------------
% Utils

not_member(_, []).
not_member([X,Y], [[U,V]|Ys]) :-
    ( X=U,Y=V -> fail
    ; not_member([X,Y], Ys)
    ).
```