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.

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.                                                          🔧 ▶
```

```
Initializing started...
Let the game begin!
I'm in [1,1], seeing: [no,no,no]
KB learn [1,2] - no Wumpus there!
KB learn [1,0] - no Wumpus there!
KB learn [2,1] - no Wumpus there!
KB learn [0,1] - no Wumpus there!
KB learn [1,2] - there's no Pit there!
KB learn [1,0] - there's no Pit there!
KB learn [2,1] - there's no Pit there!
KB learn [0,1] - there's no Pit there!
KB learn [3,2] - there's no gold here!
I'm going to: [1,2]
There's still something to do...
I'm in [1,2], seeing: [no,yes,no]
KB learn [1,3] - no Wumpus there!
KB learn [1,1] - no Wumpus there!
KB learn [2,2] - no Wumpus there!
KB learn [0,2] - no Wumpus there!
KB learn [1,3] - its a Pit!
KB learn [1,1] - its a Pit!
```

```
KB learn [2,2] - its a Pit!
KB learn [0,2] - its a Pit!
KB learn [3,2] - there's no gold here!
I'm going to: [2,1]
There's still something to do...
I'm in [2,1], seeing: [no,no,no]
KB learn [2,2] - no Wumpus there!
KB learn [2,0] - no Wumpus there!
KB learn [3,1] - no Wumpus there!
KB learn [1,1] - no Wumpus there!
KB learn [2,2] - there's no Pit there!
KB learn [2,0] - there's no Pit there!
KB learn [3,1] - there's no Pit there!
KB learn [1,1] - there's no Pit there!
KB learn [3,2] - there's no gold here!
I'm going to: [2,2]
There's still something to do...
I'm in [2,2], seeing: [no,no,no]
KB learn [2,3] - no Wumpus there!
KB learn [2,1] - no Wumpus there!
KB learn [3,2] - no Wumpus there!
KB learn [1,2] - no Wumpus there!
KB learn [2,3] - there's no Pit there!
KB learn [2,1] - there's no Pit there!
KB learn [3,2] - there's no Pit there!
KB learn [1,2] - there's no Pit there!
KB learn [3,2] - there's no gold here!
I'm going to: [3,1]
There's still something to do...
I'm in [3,1], seeing: [yes,no,no]
I'm in [3,1], seeing: [no,no,no]
KB learn [3,2] - no Wumpus there!
KB learn [3,0] - no Wumpus there!
KB learn [4,1] - no Wumpus there!
KB learn [2,1] - no Wumpus there!
KB learn [3,2] - there's no Pit there!
KB learn [3,0] - there's no Pit there!
KB learn [4,1] - there's no Pit there!
KB learn [2,1] - there's no Pit there!
KB learn [3,2] - there's no gold here!
I'm going to: [2,3]
There's still something to do...
I'm in [2,3], seeing: [no,yes,no]
KB learn [2,4] - no Wumpus there!
KB learn [2,2] - no Wumpus there!
KB learn [3,3] - no Wumpus there!
```

```
KB learn [1,3] - no Wumpus there!
KB learn [2,4] - its a Pit!
KB learn [2,2] - its a Pit!
KB learn [3,3] - its a Pit!
KB learn [1,3] - its a Pit!
KB learn [3,2] - there's no gold here!
I'm going to: [3,2]
There's still something to do...
WON!
Score: 995,
 Time: 6
 true
```

1

| Next | 10 | 100 | 1,000 | Stop |

Header and Copyright

```
 1 %----------------------------------------------------------------
 2 % A Prolog Implementation of the Wumpus World described in
 3 % Artificial Intelligence : A Modern Approach (Russel - Norvig)
 4 %
 5 % Mandatory Excercise 2007
 6 % v1.0 - Jan. 31, 2007
 7 % Richard O. Legendi
 8 %
 9 % Copied into prolog-examples with permission Richard O. Legendi
10 % Original exercise descriped in  Artificial Intelligence : A Modern Approach (I
11 %
12 % Usage:
13 % consult this file
14 % ?-start.
15 %
16 %----------------------------------------------------------------
```

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

```
1  %------------------------------------------------------------------
2  % Declaring dynamic methods
3
4  :- dynamic ([
5          agent_location/1,
6          gold_location/1,
7          pit_location/1,
8          time_taken/1,
9          score/1,
10          visited/1,
11          visited_cells/1,
12          world_size/1,
13          wumpus_location/1,
14          isPit/2,
15          isWumpus/2,
16          isGold/2
17         ]).
```

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

```
1  %------------------------------------------------------------------
2  % To start the game
3
4  start :-
5      format('Initializing started...~n', []),
6      init,
7      format('Let the game begin!~n', []),
8      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.

```
1  %-----------------------------------------------------------------------------
2  % Scheduling simulation:
3
4  step_pre(VisitedList) :-
5      agent_location(AL),
6      gold_location(GL),
7      wumpus_location(WL),
8      score(S),
9      time_taken(T),
10
11     ( AL=GL -> writeln('WON!'), format('Score: ~p,~n Time: ~p', [S,T])
12     ; AL=WL -> format('Lost: Wumpus eats you!~n', []),
13                 format('Score: ~p,~n Time: ~p', [S,T])
14     ; take_steps(VisitedList)
15     ).
16
17 take_steps(VisitedList) :-
18     make_percept_sentence(Perception),
19     agent_location(AL),
20     format('I\'m in ~p, seeing: ~p~n', [AL,Perception]),
21
22     update_KB(Perception),
23     ask_KB(VisitedList, Action),
24     format('I\'m going to: ~p~n', [Action]),
25
```

```
26       update_time,
27       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

```
1  %--------------------------------------------------------------------------
2  % Updating states
3
4  update_time :-
5      time_taken(T),
6      NewTime is T+1,
7      retractall( time_taken(_) ),
8      assert( time_taken(NewTime) ).
9
10 update_score :-
11     agent_location(AL),
12     gold_location(GL),
13     wumpus_location(WL),
14     update_score(AL, GL, WL).
15
16 update_score(P) :-
17     score(S),
18     NewScore is S+P,
19     retractall( score(_) ),
20     assert( score(NewScore) ).
21
22 update_score(AL, AL, _) :-
23     update_score(1000).
24
25 update_score(_,_,_) :-
26     update_score(-1).
27
```

```
28 undate agent location(NewAL) :-
```

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

```prolog
 1 %----------------------------------------------------------------------
 2 % Display standings
 3
 4 standing :-
 5     wumpus_location(WL),
 6     gold_location(GL),
 7     agent_location(AL),
 8
 9     ( is_pit(yes, AL) -> format('Agent was fallen into a pit!~n', []),
10       fail
11     ; stnd(AL, GL, WL)
12       %\+ pit_location(yes, AL),
13     ).
14
15 stnd(_, _, _) :-
16     format('There\'s still something to do...~n', []).
17
18 stnd(AL, _, AL) :-
19     format('YIKES! You\'re eaten by the wumpus!', []),
20     fail.
21
22 stnd(AL, AL, _) :-
23     format('AGENT FOUND THE GOLD!!', []),
24     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.

```
1  %------------------------------------------------------------------------
2  % Perceptotion
3
4  make_perception([_Stench,_Bleeze,_Glitter]) :-
5      agent_location(AL),
6      isStinky(AL),
7      isBleezie(AL),
8      isGlittering(AL).
9
10 test_perception :-
11     make_percept_sentence(Percept),
12     format('I feel ~p, ',[Percept]).
13
14 make_percept_sentence([Stench,Bleeze,Glitter]) :-
15     smelly(Stench),
16     bleezy(Bleeze),
17     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,

```
1  %------------------------------------------------------------------------
2  % Initializing
3
4  init :-
5      init_game,
6      init_land_fig72,
7      init_agent,
```

```
 8      init_wumpus.
 9
10 init_game :-
11     retractall( time_taken(_) ),
12     assert( time_taken(0) ),
13
14     retractall( score(_) ),
15     assert( score(0) ),
16
17     retractall( visited(_) ),
18     assert( visited(1) ),
19
20     retractall( isWumpus(_,_) ),
21     retractall( isGold(_,_) ),
22
23     retractall( visited_cells(_) ),
24     assert( visited_cells([]) ).
25
26 % To set the situation described in Russel-Norvig's book (2nd Ed.),
27 % according to Figure 7.2
```

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.

```
 1 %-----------------------------------------------------------------------
 2 % Perceptors
 3
 4 %%% Institiation error!!!
 5
 6 %adj(X,Y) :-
 7 %    world_size(WS),
 8 %    ( X is Y+1, Y   < WS
 9 %    ; X is Y-1, Y-1 > 0
10 %    ).
11
12 adj(1,2).
13 adj(2,1).
```

```
14  adj(2,3).
15  adj(3,2).
16  adj(3,4).
17  adj(4,3).
18
19  adjacent( [X1, Y1], [X2, Y2] ) :-
20      ( X1 = X2, adj( Y1, Y2 )
21      ; Y1 = Y2, adj( X1, X2 )
22      ).
23
24  %adjacent([X1,Y],[X2,Y]) :-
25  %     adj(X1,X2).
26
27  %adjacent([X,Y1],[X,Y2]) :-
```

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

```
1  %-----------------------------------------------------------------------------
2  % Knowledge Base:
3
4  update_KB( [Stench,Bleeze,Glitter] ) :-
5      add_wumpus_KB(Stench),
6      add_pit_KB(Bleeze),
7      add_gold_KB(Glitter).
8
9  % if it would be 'yes' -> it would mean the player is eaten ;]
10 add_wumpus_KB(no) :-
11     %agent_location(L1),
12     %adjacent(L1, L2),
13     %assume_wumpus(no, L2).
14     agent_location([X,Y]),
15     world_size(_),
16
17     % Checking needed!!
18     % adj will freeze for (4,_) !!
19
20     Z1 is Y+1, assume_wumpus(no,[X,Z1]),
21     Z2 is Y-1, assume_wumpus(no,[X,Z2]),
22     Z3 is X+1, assume_wumpus(no,[Z3,Y]),
23     Z4 is X-1, assume_wumpus(no,[Z4,Y]).
24
25 add_pit_KB(no) :-
26     agent_location([X,Y]),
27     Z1 is Y+1, assume_pit(no,[X,Z1]),
28     Z2 is Y-1, assume_pit(no,[X,Z2]),
```

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).

```
1  %-----------------------------------------------------------------------------
2  % Utils
3
4  not_member(_, []).
5  not_member([X,Y], [[U,V]|Ys]) :-
6      ( X=U,Y=V -> fail
7      ; not_member([X,Y], Ys)
```

```
8      ).
```