* **Prolog: Lab Search**

For this final prolog exercise, you will implement a breadth first search to find a path to an oracle (shown as a red square) located somewhere on the grid. A breadth first search is a slow algorithm but will always find the shortest path. This exercise may be tough, but assuming you have completed the other exercises, then you will be able to do it! For this exercise you will be editing lab\_search\_1234567.pl.

**Possible Queries**

The exported queries for this exercise are the same as Lab Grid, except map\_adjacent is now also exported. See below for a list:

| **Query Name** | **Description** |
| --- | --- |
| start | Initiate the web server |
| shell | Open the command shell |
| my\_agent(-A) | Returns the ID of your Agent |
| ailp\_grid\_size(-N) | Returns the size of the grid |
| get\_agent\_position(+A,-Pos) | Returns the position of Agent A |
| agent\_do\_moves(+A,+L) | Makes Agent A perform the list of moves L |
| map\_adjacent(+Pos,?AdjPos,?OID) | Returns positions adjacent to Pos and their OID |

**How to run**

Run using swipl ./ailp.pl lab search. The file lab\_search\_1234567.pl will automatically be consulted and should be the place that your work is done.

This time you will need to follow the steps from exercise 1 to run the webserver (start., click run on the website, shell., setup.). You can use search. while in the shell as a shortcut for typing search\_bf outside of the shell.

**Exploring map\_adjacent/3**

After you have loaded the exercise you will notice that you are now in a more complicated version of the world from last week's lab. The world is randomly generated at runtime but will have at most 25 walls (in black) and exactly 1 oracle (in red). Not every world will be solvable, and if it's not then your predicate should just fail!

The query map\_adjacent(+Pos,?AdjPos,?OID) must receive a Pos although it is optional whether or not to give an AdjPos/OID. OID will always be a member of [agent, empty, t(X), o(X)].

For this section, try to write queries in your terminal that:

* 1. Find all the adjacent cells around position P (giving multiple solutions)
  2. Find all the adjacent cells around position P (as a List)
  3. Find **ONLY** the empty cells around position P
  4. Find the empty cells next to your agent
  5. Find the OID of the oracle

NOTE: These queries don't need to work generally, it is enough that they only work in your current version of the grid (i.e. it is ok to instantiate things like P).

**Create predicate complete/1**

As the goal of the task is to move next to an oracle, the predicate complete(+Pos) must be defined in order to know when the task is done. This predicate should be true if the position is adjacent to an oracle.

**Create the predicate search\_bf/0**

You need to implement search\_bf/0 as well as any other supporting predicates that you want to use. search\_bf. should use a breadth first search to navigate the agent to the oracle. This is a more complex task than previous but these tips should help guide you:

* 1. You may want to find the path to the oracle and then make all the moves at once using agent\_do\_moves/2.
  2. Remember that move lists given to agent\_do\_moves/2 **DO NOT** include the current position at the start
  3. Keeping track of which cells have already been visited will make sure that you don't check cells multiple times.
  4. Test often, And use trace/0 if you aren't sure why something isn't working.
  5. Don't be afraid to ask for help!
* **Week 3 - Lab Hints**

[Week 3 - Lab Hints](https://www.ole.bris.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_257207_1&content_id=_8044791_1&mode=reset)

In today's lab you will implement a breadth-first search method to find a shortest path from your agent's location to an oracle at an initially uknown location which must navigate around any intervening walls. You are given a skeleton answer with stubs for two wrapper predicates that yo need to write:

* 1. **complete**/1 (that should be true when the given cell is next to an oracle)
  2. **search\_bf**/0 (which finds your agent's initial position and either does nothing (if your agent happens to start off next to an oracle) or else calls a recursive helper predicate that performs a bfs, and then executes agent\_do\_moves (if necessary after first reversing and chopping the head off the computed path)

Your implementation of the bfs helper predicate will essentially follow the abstract agenda-based formulation of bfs given in the lecture slides:

* 1. search\_bf([Goal|Rest],Goal):-  
          goal(Goal).
  2. search\_bf([Current|Rest],Goal):-  
          children(Current,Children),  
          append(Rest,Children,NewAgenda),  
          search\_bf(NewAgenda,Goal).

However, you will need to make four main changes:

* 1. the call to goal/1 will be replaced by a call to complete/1.
  2. you will need to replace the call to children/2 by some code that uses one of the lab queries to find the empty cells adjacent to your the current position.
  3. to avoid any possibility of getting stuck in cycles you will probably want to include an extra argument to store the list of visited nodes (i.e. the nodes you have removed from the agenda and replaced by their children). Note that, in textbook definitions, the list of visited nodes is sometimes called the **closed set** and the agenda is sometimes called the **open set**.
  4. in order to actually return the computed path, you will probably need to add an extra argument for that as well. As explained and illustrated in **section 5.2 of Simply Logical**, this is most easily achieved by storing (reverse) paths on the agenda instead of just nodes.

Good luck!