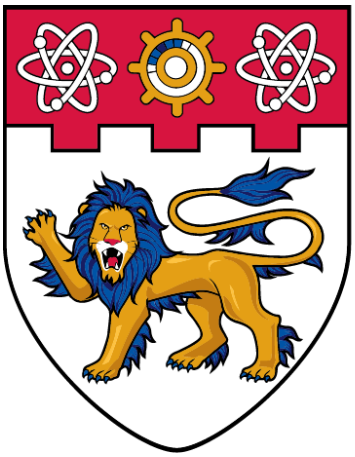

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence
Lab Assignment 2

Group Name: SJY
Lab Group: TS5

Learning to Use Prolog as a Logic Programming Tool

Prepared By	
Seah Jie Yu	U2022275K

1.0 Agent

1.1 reborn/0

reborn/0 is queried by the driver when the agent encounters the Wumpus, and the game needs to be restarted. All sensory inputs are lost, the arrow will be returned to the agent, the Wumpus will be revived (if it was previously dead), and the agent relative position will be back to the relative origin.

1.2 reposition(L) and move(A,L)

reposition(L) will only be called when the game starts, or when the agent walks into a Confundus portal. As such, the Confundus indicator is always on, and bump and scream indicator will always be off. If the stench indicator is on, stench will be asserted at the agent's relative position (i.e. 0,0), and the adjacent cells will be marked as Wumpus cells. The same is done for the tingle indicator and Confundus cells. Since the agent will never spawn on a cell with another NPC, there is no need to check for the glitter indicator.

move(A,L) is similar to reposition(L), with the exception that the agent will be checking the action A which he had previously taken.

A is	Indicator On	Agent Will
shoot	scream	Retract all stench, Wumpus cells. Set wumpusalive as false.
moveforward	bump	Assert wall at current position, then move back and update new current position.
	stench	Assert stench at current position and Wumpus at adjacent cells.
	tingle	Assert tingle at current position and Confundus at adjacent cells.
	glitter	Assert tingle.
	confounded	Retract all sensory information, visited, safe, wall cells, and agent current position. Set the new current position as (0,0,rnorth).

Figure 1: Agent's responses after receiving L.

Note that if stench and tingle indicators are both absent, the agent will set adjacent cells as safe cells. This applies to both reposition(L) and move(A,L). In addition, stench and Wumpus cells will only be asserted if Wumpus is alive.

1.3 Localization and Mapping

visited(X,Y)	True if the agent is currently in a cell which does not contain a portal or the Wumpus.
wumpus(X,Y)	True, if any of the adjacent cells have stench, Wumpus is alive, the cell is not a wall, and the cell was not previously marked as safe.
confundus(X,Y)	True if any of the adjacent cells have tingle, the cell is not a wall, and the cell was not previously marked as safe.
tingle(X,Y)	True if adjacent cells might contain a Confundus portal.
glitter(X,Y)	True if there is a coin in the cell, and the agent has yet to pick it up.
stench(X,Y)	True if adjacent cells might contain the Wumpus.
safe(X,Y)	True if the agent is currently in a cell which does not contain a portal or the Wumpus. Also true if one of the adjacent cells does not have stench or tingle. If the cell was previously marked as a Wumpus or Confundus cell, the agent will retract them (i.e. safe will override).
wall(X,Y)	True if the cell contains the wall. If this cell was previously marked as a safe, Wumpus, or Confundus cell, the agent will retract them.

Figure 2: Situations where predicates will be True, otherwise False.

1.4 Actions

1.4.1 moveforward

When moveforward is queried, the agent uses current/3 to get its current relative position which includes the orientation he is facing. Depending on which direction the agent is facing, new X

and Y coordinates will be calculated. For example, if the agent is currently facing relative north (rnorth), the X coordinate will remain unchanged, and Y will be incremented by 1. The orientation of the agent remains unchanged. The previous current position of the agent will then be retracted, and the new coordinates will be asserted.

1.4.2 turnleft and turnright
For turning actions, the agent uses current/3 to get the direction he is facing currently. If turnleft is queried, the direction of the agent will change by 90 degrees counterclockwise. If turnright is queried, the direction of the agent will change by 90 degrees clockwise. The previous current/3 will be retracted and the new one containing the agent new direction will be asserted.

1.4.3 pickup
When pickup is queried, the agent uses current/3 to get its current X and Y coordinates and uses them to retract glitter at its current position.

1.4.4 shoot
When shoot is queried, the agent first checks if hasarrow/0 is true. If it is true, it will print “Agent shoots.”, and hasarrow/0 will be retracted (set to false). If the agent tries to shoot again, it will print “Agent does not have an arrow.” instead.

1.5 explore(L)
explore(L) is implemented mainly using **exploremap(L)** and **returntoorigin(L),** both of which return a safe path for the agent to take to get from the starting node to the goal node. The starting node will always be the agent’s current relative position. The goal node for exploremap(L) is a safe and unvisited cell, and the goal node for returntoorigin(L) is the relative origin (0,0).

Generating the safe path first begins by using the depth-first search algorithm in Prolog. The algorithm begins at the starting node, and checks if the starting node is the goal node. If it is not the goal node, it will find a safe cell that is adjacent to the node, and call depth-first search recursively until it reaches the goal node. The output of the algorithm is a list of coordinates - the first coordinate being the agent’s current relative position, and the subsequent ones being safe cells which the agent should move to in order to reach the goal node (the last coordinate). An example of the DFS algorithm for exploremap(L): If the agent’s current relative position is (0,2), and the depth-first search algorithm returns [[0,2],[0,1],[0,0],[1,0]], this means that the agent has to move to (0,1) next, then (0,0), before arriving at the safe and unvisited cell at (1,0).

Next, we will need to translate the list of coordinates into a list of actions, so that the driver can iterate through them and query the agent to perform the actions one by one. This is done by looping through the coordinates in Prolog. In the first loop, the first coordinate in the list will be stored in currentcopy/3, along with the direction of the agent which can be found at current/3. Subsequent loops will check what actions are needed for the agent to go from the coordinate in currentcopy/3 to the coordinate in the loop. The actions are asserted into the list movements/1. The coordinate in the loop, along with the new direction will be then asserted as the new currentcopy/3. An example of how the loop works: If the list of coordinates provided by DFS is [[0,2],[0,1],[0,0],[1,0]], agent is facing relative north, and we are in the second iteration of the loop, currentcopy/3 will be (0,2,rnorth), and the coordinate in the current iteration of the loop will be (0,1). For the agent to go from the coordinate in currentcopy/3 to (0,1), the agent will need to turn left, turn left again, then move forward. Hence, these 3 actions are asserted into movements/1. The new direction will now be relative south, and the new currentcopy/3 will be (0,1,rsouth).

For explore(L) to decide between whether to call exploremap(L) or returntoorigin(L), it will use findall/3 to generate a list which contains all the cells in the knowledge base that are known to be safe and unvisited. If this list is not empty, meaning that there are safe and unvisited cells in the relative map to be explored, it will call exploremap(L). If it is empty, it will simply call returntoorigin(L) to get a safe path back to the relative origin, and then the game will end.

I have also made a slight enhancement to exploremap(L) – The DFS algorithm will prioritize choosing a node that is both safe and unvisited, instead of simply choosing the next safe node it can find. This helped to reduce the number of iterations greatly.

2.0 Driver

2.1 World Creation

The driver begins calling the function create_empty_map() which will return an empty map of size 7 x 6, where each cell is represented by a 3 x 3 map cell. All the outer cells in the map are wall map cells, which means that all its symbols are "#". Next, a Wumpus, at least 1 coin, and at least 3 Confundus Portals are inserted into the empty map at random positions, each taking up a unique random inner map cell. Stench and tingle indicators are then introduced into the map, in the adjacent map cells of those which are inhabited by the Wumpus and Confundus Portals correspondingly. Lastly, an agent is inserted into the map at a random safe position (i.e. a map cell that is not inhabited by a Wumpus or Confundus Portal), facing a random direction. This final map will be known as the absolute_map.

2.2 Sensory Interface

If the agent is in a cell with no tingle and no stench, the adjacent cells will be printed with an 's'. For tingle and stench cells, 'O' and 'W' will be printed in the adjacent cells respectively. If both stench and tingle is present, 'U' will be printed instead in the adjacent cells.

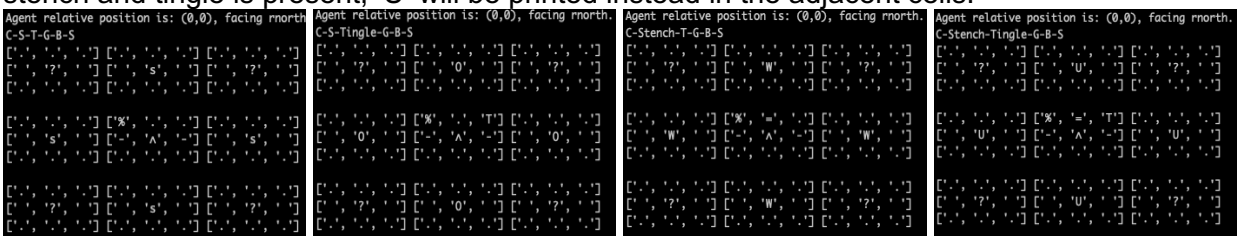

Figure 3: Agent in Non-Tingle+Stench, Tingle, Stench, Tingle+Stench cells.

If a cell is safe and visited, it will be printed with a 'S' instead. Bump, scream, and confounded indicators are transitory; hence they are not permanent.

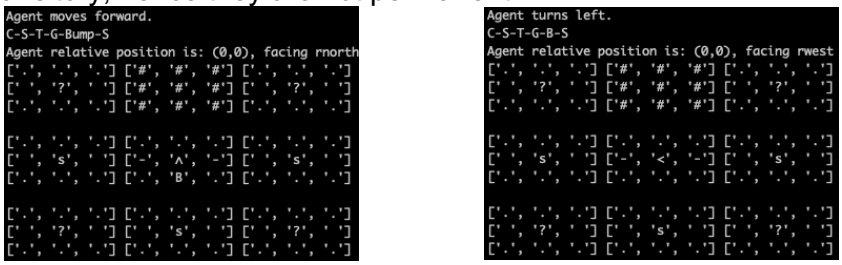

Figure 4: Bump disappears after making another action. Before on the left, after on the right.

2.3 Relative Map

The relative map perceived by the agent is always the size of 2k+1 by 2k+1, with the center of the map being the relative origin (0,0). The value of k will depend on whether the map is big enough to display all the information that could be perceived by the agent. For example, the first

relative map will always be 3x3 (i.e. k is 1), which is big enough to display the agent in the center, and all of the possible senses it might perceive at that location. If the agent chooses to move to the edge of the relative map, for example, by moving forward, k will be incremented by 1, making the new relative map to be of size 5x5.

2.3 Shooting the Wumpus, Walking into a Confundus Portal

If the agent has the arrow and driver queries shoot, the driver will check if the Wumpus has successfully been killed (i.e. Wumpus is in the same direction the agent is facing). If so, the driver will update the agent using move(A,L), and the agent will remove all stench indicators and Wumpus cells.

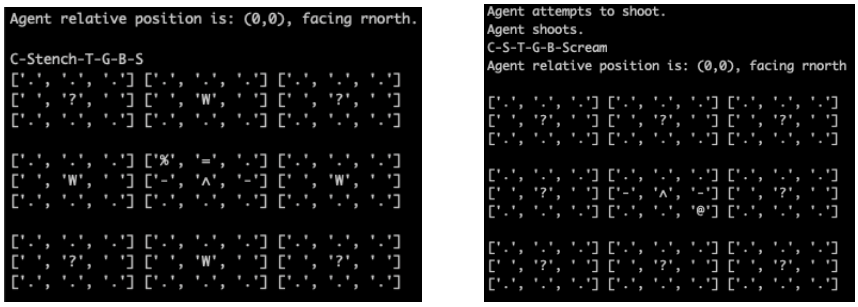

Figure 5: Wumpus successfully killed. Before on the left, after on the right.

If the agent happens to walk into a Confundus Portal, the driver will query move(A,L) with the Confounded indicator on, so that the agent can retract all sensory inputs and reset the relative position back to (0,0). The agent is then placed into another random but safe location in the same absolute map. The driver will then query reposition(L) to update the agent with the sensory inputs found in the new cell. Lastly, the driver will print the new absolute map followed by the new 3x3 relative map.

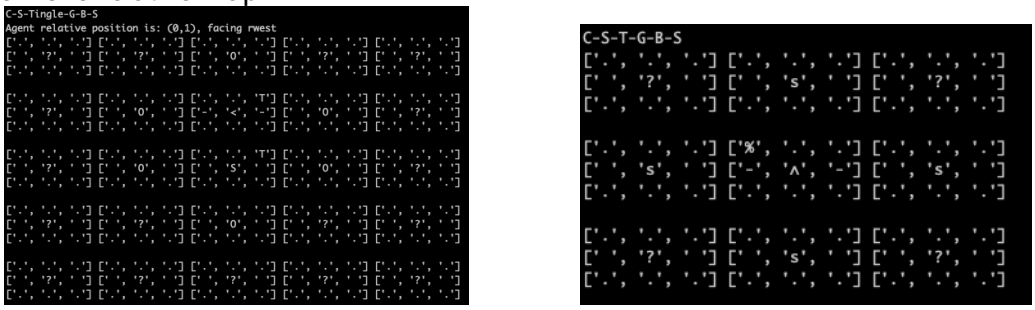

Figure 6: Agent walks into a Confundus Portal. Before on the left, after on the right.

2.4 Setting Up the Game

At the start of the game, the driver will print the absolute map. Next, the driver calls the agent_sense function to sense the cell the agent is in, in the absolute map, to get a list of senses L. Then, the driver uses this list to query reposition(L) to update the agent's knowledge base. Finally, the relative map is printed, which is simply a reflection of the information the agent has. The driver will now query explore(L) to get a list of actions that will lead the agent to the next safe and unvisited cell, which will be executed in the main loop.

2.5 Main Loop

The main loop consists of a while(true) loop which will only exit if explore(L) returns false (i.e. agent has finished exploring the map and returned back to the relative origin) or if the agent has encountered the Wumpus. In this while loop, there is a for loop, which iterates over each action provided by explore(L).

In the for loop, the driver will first query the agent to perform the action. Next, the driver calls update_agent_abs_cell to get the updated cell the agent is in, in the absolute world. With the updated absolute cell the agent is in, the driver will call agent_sense on it to get the new list of sensory inputs L. Then, the driver queries move(A,L) to the agent, so that the agent knows what new information (from L) which it has gotten after performing A, the action. Next, the driver will check if the action performed by the agent will require the relative map to be expanded. The driver does so by checking if the new agent position is at the edge of the current relative map. Finally, the driver calls update_relative_map, and prints the updated relative map, displaying all the latest information the agent has gotten in the knowledge base. After the end of each for loop, the driver will query explore(L) again to get the next set of actions.

With reference to my printout file SJY-testPrintout-Self-Self.txt, at the end of iteration 15, since there is still one more safe and unvisited cell in the relative map, the agent prints “Agent will move to the next safe and unvisited cell” and explore(L) returns a safe path for the agent to take in order to reach that cell. The driver makes the agent take the path in iteration 16, and at the end of this iteration, the agent prints “There are no more safe and visited cells. Agent will now return to the relative origin cell” and explore(L) returns a safe path back to the relative origin. The driver makes the agent take the path in iteration 17, where the agent finally prints “There are no more safe and visited cells. Agent is now back to the relative origin cell”, concluding the end of the game.

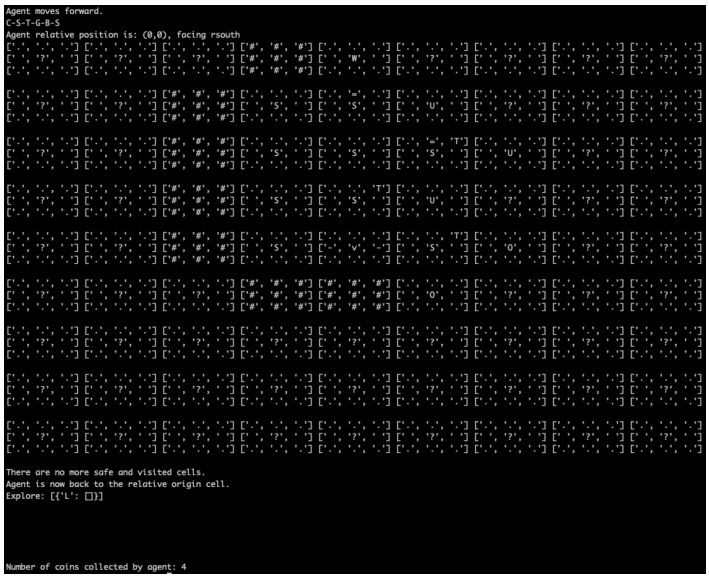

Figure 7: End of the game where there are no more safe and unvisited cells to explore, and agent is back to his relative origin.

3.0 Conclusion

This project has given me a deeper understanding into writing and understanding proposition logics. I have learned the basics of coding in Prolog, and even learned how to implement more advance techniques such as recursion and loops with it. I am now more confident working with other AI projects in the future.

Friend Driver and printout of Friend Agent test are provided by Catsong. There is also a **README.txt** file in the folder explaining the purposes of each file.

References: https://www.javatpoint.com/prolog