CZ3005 Artificial Intelligence Lab 2 Report

Team StopMug

Team Members and Contribution

Name	Matric Number	Contribution
Xia Tianyi	U2020801C	Report + Agent perception logic + Driver testing
Brendan Ang Wei Jie	U2021332F	Agent code + report editing
Justyn Phoa Zairen	U2022593D	Driver code and testing + report

Agent Tasks:

Approach: Prolog agent will have the ability to move forward, turn left, turn right, pick up coins or shoot arrows. These actions can be carried out with the move(A, L) command with A representing the action while L represents the perceptions of the agent.

reborn/0

Reborn is implemented by moving the agent to a random location of the wumpus world map which is at position (0, 0) on the agent's relative map. The agent's knowledge base of the world will be reset.

move(A, L)

Move(A, L) takes in action and agent perception as the parameter and performs the action stated.
If the action is "shoot", the agent will shoot an arrow if the agent still has an arrow, otherwise, it will display no more arrows message.
Action "turnleft" and "turnright" will change the agent's relative direction and will always return true.
Pickup will display "Got the gold!" and return true when the glitter indicator in the perception L is on, else it will return false.
Moveforward will return true and the display agent's new relation position and direction if the bump indicator in the perception is off. Otherwise when the bump indicator is on it means the cell agent trying to move into is a wall, move(A, L) will return false.

```
?- move(moveforward,[no,no,no,no,no,no]).
I am at: r(0,1)rnorth
true.
```

```
?- move(shoot,[no,yes,no,no,no,no]).
I do not have arrows anymore.
true.
```

reposition(L)

Reposition is implemented for situations where the Agent has walked into a Confundus and caused a reset. This is similar to reborn but takes in perceptions as the parameter which is the initial sensory information of the agent in its new initial position.

```
?- reposition([no,no,no,yes,no,no]).
true.
```

visited(X, Y)

Visited allows the player to check whether the agent has visited a certain cell already, the (x, y) relative coordinate of the cell will be entered as a parameter for the checking. When the agent has entered a certain cell through a "moveforward" action, the cell will be asserted as visited, query visited(X, Y) will return true if the cell(X, Y) has been asserted as visited and returns false otherwise.

wumpus(X, Y)

Wumpus(X, Y) is implemented to check for the condition of whether a wumpus exists in a cell. If an adjacent cell has a stench indicator and the cell in question has not been visited by an agent or does not contain a wall, it will be marked as wumpus, as there is a possibility that the wumpus exists in the cell. When the agent explores, it will not enter any cell that has the wumpus indicator. When the agent can confidently conclude that the wumpus is in a certain cell, the particular cell will be marked as certainWumpus and all other cell wumpus indicators will disappear. If the wumpus has been shot and the agent hears a scream, all wumpus indicators will also be removed.

```
?- wumpus(1,1).
true .
```

```
?- certainWumpus(1,1).
true.
```

```
?- wumpus(2,2).
false.
```

certainWumpus(X, Y) is implemented by focusing on 3 conditions where the wumpus existence in a certain cell can be confirmed.

1. When there are 2 cells marked stench located diagonally on the map with one cell(a,b) and the other cell(a+1,b+1). We will then select the cell(a,b) and remove all the visited adjacent cells as well as any cell that is not adjacent to other stench nodes. If only one cell remains, the wumpus is located and can be marked as certainWumpus.
2. When 3 adjacent cells of a cell(x,y) are marked as stench (i.e. visited as well), the cell(x,y) can be marked as certainWumpus as well.
3. For any cell(x, y) marked as stench, when 3 adjacent cells are marked as safe(x, y) or wall(x, y), then the last adjacent cell can be marked as certainWumpus.

Diagrams demonstrating these 3 scenarios can be found below and in Appendix C.

Certain Wumpus	= Stench
= Stench	Safe/Wall

case 1

= Stench	
Certain Wumpus	= Stench
= Stench	

case 2

	Safe/Wall	
Certain Wumpus	= Stench	Safe/Wall
	Safe/Wall	

case 3

confundus(X, Y)

Confundus(X, Y) is implemented to check for possible portals existing in unvisited cells. If there is a tingle in cell(a,b), all adjacent cells which are unvisited cells may contain the portal, passing in the location of any of these adjacent cells as parameters to Confundus(X, Y) will return true. At the same time, a predicate certainConfundus(X, Y) will return true if a cell Tingle indicator is on and all its adjacent cells are explored except 1.

tingle(X, Y)

Tingle will be asserted to a cell if there is a portal in any of its adjacent cells. Tingle(X, Y) is implemented to check if a certain cell contains the tingle indicator. If the location of a cell has the tingle indicator marked as on, Tingle(X, Y) will return true.

glitter(X, Y)

Glitter will be asserted to a cell if there is a coin in the cell. Glitter(X, Y) is implemented to check if a certain cell contains the glitter indicator, which also means that the cell contains a coin. If the location of a cell has the glitter indicator marked as on, Glitter(X, Y) will return true. After the coin is picked up from a glitter cell(a,b), the glitter indicator on cell(a,b) will be unasserted.

stench(X, Y)

Stench will be asserted to a cell if the wumpus is located in any of its adjacent cells. Stench(X, Y) is implemented to check if a cell contains the stench indicator. If the location of a cell containing the stench indicator marked is passed as parameters, the stench(X, Y) will return true. The stench indicator of a cell will be unasserted when certainWumpus is asserted in any cell that is not adjacent to the cell.

safe(X, Y)

A cell is considered safe if there is no confundus or wumpus in the cell. Safe(X, Y) is implemented to check if a cell is safe.

1. If a cell(a,b) has been visited, it is safe.
2. If certainWumpus has been ascertained, the cell is safe if it is not the Wumpus cell and is not adjacent to tingle cells.
3. If certainWumpus is not ascertained, the cell is safe only if it is not adjacent to any tingle and stench indicated cells.

wall(X, Y)

Wall(X, Y) is implemented to check whether there is a wall in a cell by passing in the cell location as parameters, if the cell contains a wall query will return true, otherwise return false.

explore(L)

Explore uses a breadth-first-search approach with a queue structure in order to find a path to a safe and unvisited node. This path is then converted into actions of set { [moveforward], [turnleft, moveforward], [turnright, moveforward], [turnleft, turnleft, moveforward] } corresponding to (move relative North, move relative West, move relative East, move relative South }

1. L is not given: the predicate will return the Actions needed to reach a safe and unvisited node. If no such node is possible to reach from the Agent's current position, it will return False.
2. L is given as a list of actions: the predicate will return True if every node visited via this list of actions is safe, and that the final room is safe and unvisited. Elsewise, it will return False.

```
?- explore(L).
L = [[turnleft, moveforward]].
```

current(X, Y, D)

Current(X, Y, D) is implemented to check for the location and direction of the agent on its relative map. If the agent's relative position and the direction it is facing matches the query parameters passed in, the query will return true, otherwise it will return false.

```
?- current(1,0,rwest).
true.
```

```
?- current(1,0,reast).
false.
```

hasarrow/0

Hasarrow/0 will return true if the agent has arrows and return false if the agent has run out of arrows. The agent will have one arrow at the start of the game and it will use up the arrow after performing the shoot action.

Driver Tasks:

Approach - The driver needs to carry out 5 tests on the Agent, namely:

1. Correctness of Agent's localisation and mapping abilities
2. Correctness of Agent's sensory inference
3. Correctness of Agent's memory management in response to stepping into a Confundus Portal
4. Correctness of Agent's end-game reset in a manner similar to that of Confundus Portal reset
5. Correctness of Agent's exploration capabilities

For all tests, a specially built world is used repeatedly to test the Agent. The layout of this world can be found in Appendix A. The overall approach used in the driver testing is to order the Agent to execute a specific sequence of actions, and at the end, the user will be able to confirm if the Agent has executed its moves properly / inferenced its perceptions properly.

Test 1:

The driver makes the agent perform a non-trivial sequence of actions in the custom world, and then prints out the expected relative end location of the Agent. At the same time, the driver also queries for the agent's current location to do a comparison of the actual relative location to check if it is the same as the expected relative location. The same is done for the relative direction that the Agent is facing.

Test 2:

The driver makes the agent perform a non-trivial sequence of actions in the custom world, and then queries for several perceptions (including portal/2, certainWumpus/2, wumpus/2, etc) to compare with the expected inferences of the Agent.

Test 3:

The driver makes the agent perform a sequence of actions in the custom world to build up a non-trivial map within the Agent's knowledge. After which, the driver asks the agent to reposition (simulating the Agent stepping through a Confundus Portal). The driver then queries the agent to check for various sensory inferences (if they have been reset properly), as well as whether the agent has the arrow (should be consistent with before repositioning).

Test 4:

Similar to test 4, the driver makes the agent perform a sequence of actions in the custom world to build up a non-trivial map within the Agent's knowledge. After which, the driver asks the agent to "reborn" (simulating the Agent stepping into the Wumpus cell, hence restarting the game). The driver then queries the agent to check for various sensory inferences (if they have been reset properly), as well as whether the agent has the arrow (should be true).

Test 5:

In test 5, the driver repeatedly calls the explore function of the Agent to obtain a list of moves that would lead to a safe and unvisited cell. The driver then feeds these moves back to the agent with the related percepts. This continues until there are no more moves left to execute, at which point the driver will stop the test. Our team believes that comparing the final relative Agent map to the expected map would be sufficient to determine the correctness of the Agent's exploration capabilities. The expected map can be found in Appendix B.

Correctness:

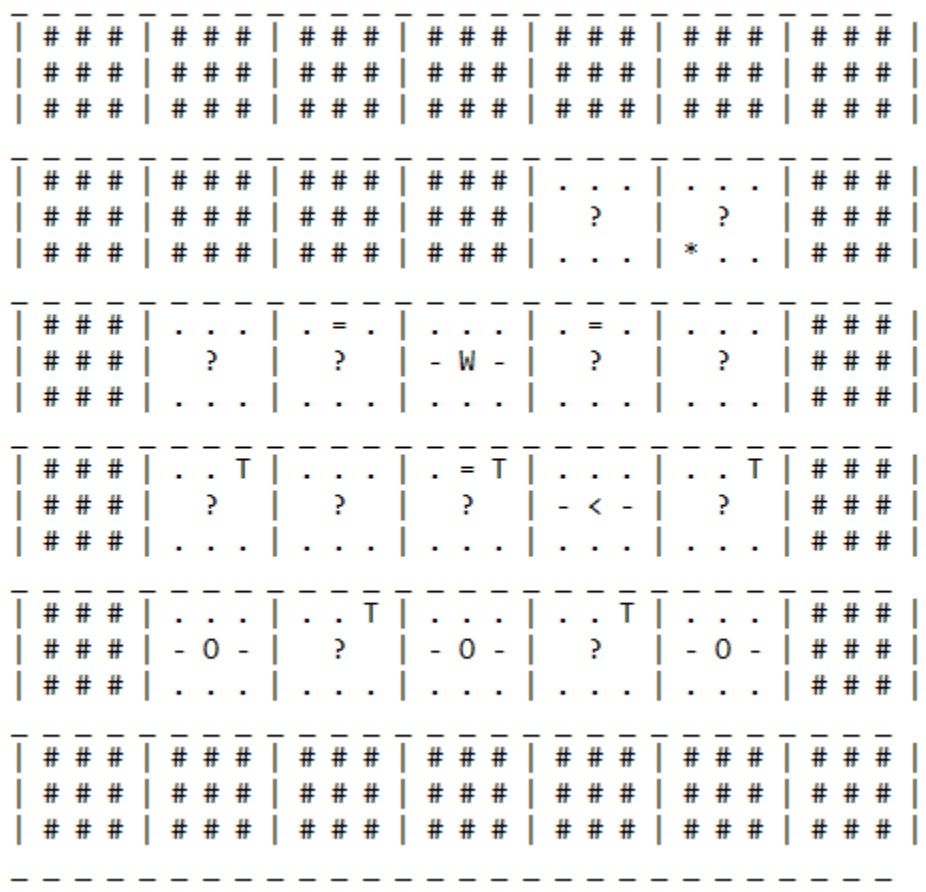
In this project, our team did not automate any tests to prove the correctness of the Agent's capabilities, rather, the user has to manually verify if the Agent's actual results match the expected results. As long as the expected inferences and results are consistent with the Driver's, then we can conclude that

Conclusion

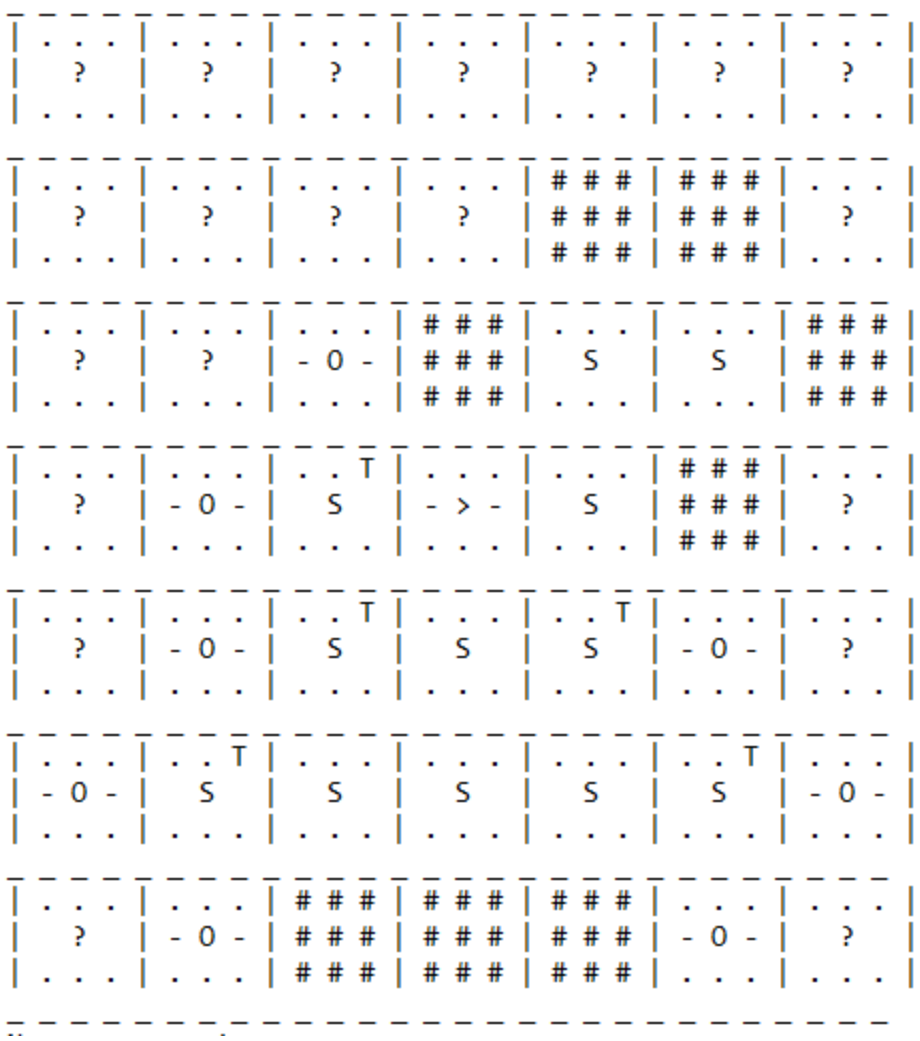
The team learned the application of propositional logic contextually by applying the logic to create a game. The prolog language allows us to code logical statements to build an agent which makes decisions based on logical reasoning and deduction. We then learned to create a map and build a driver to connect to prolog using the pyswip Prolog package and query the prolog agent. During the testing and querying using the driver, we discovered a lot of conditions that we did not consider initially and found the logic from our test result, which we then implemented into the agent. We also discovered that using a BFS method for the explore function will be a better approach than using DFS as we will have a higher chance to discover the closest unvisited node instead of constantly traveling through the visited cells.

Appendix

Appendix A: Driver Test World



Appendix B: Expected Exploration map by Agent



Appendix C: Certain Wumpus

Case 1:

Certain Wumpus	= Stench
= Stench	Safe/Wall

Case 2:

= Stench	
Certain Wumpus	= Stench
= Stench	

Case 3:

	Safe/Wall	
Certain Wumpus	= Stench	Safe/Wall
	Safe/Wall	