NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005 (Artificial Intelligence)

Lab Assignment 2 Submission

By

AlphaBlur

Team Member Name	Matriculation No.
Koh Jia Cheng	U2022450J
Muqaffa Al-Afham Bin Kamaruzaman	U2022554C

Lab Group: TS4

Date of Submission: 18th Apr 2022

1. TASKS

1.1. Agent Code in Prolog

1.1.1. Localization and Mapping

In addition to the terms required by the assignment, several other terms are implemented to improve the agent's localization and mapping, including *no_wumpus/2* and *no_confundus/2*, which imply that a particular grid cell definitely contains no wumpus and/or no confundus portal. The absence of a stench/tingle in a cell triggers the assertion of *no_wumpus/no_confundus* for the 4 neighbouring cells. The required term *safe/2* is completely determined by these 2 terms.

1.1.2. Exploration Capability

In order to support the required the term *explore/1*, another term *dfs/1* is implemented, which applies depth-first-search (DFS) to find a path to all safe, unvisited, accessible cells. This term requires 2 additional terms *sim_current/3* – which marks the agent's position during the DFS simulation – and *sim_visited/2* – which informs whether a particular cell has already been visited in the DFS simulation. The following is the pseudocode:

```
dfs(L):
        sim_current = get_sim_current
        if visited(sim_scurrent)
                add L to explore/1
        end if
        add sim_current to sim_visited/2
        for each neighbour v of sim_current
                move sim_current to v's position
                if safe(sim_current) and !wall(sim_current) and !sim_visited(sim_current)
                        actions = actionsToMoveTo(v)
                        newL = L.append(actions)
                        dfs(newL)
                end if
        end for
```

This implementation ensures that each safe, unvisited cell gets only 1 path (sequence of actions) leading to it, which is intended as there are theoretically infinite paths to a cell (note that turning right 4 times in a row causes no net change to the agent's position). *dfs/1* is called at the end of *move/2* and at the end of *reposition/1*.

1.1.3. Agent's Reset

In *reposition/1*, all facts of the knowledge base are first retracted. The agent's current relative position is then set to (0, 0, rnorth), the knowledge base is updated according to the stench, tingle and glitter sensory inputs, and finally, *dfs/1* is called.

1.2. Wumpus World Driver

1.2.1. Map Layout

Initially, the relative map is of size 3x3, with the relative origin at the centre. Subsequently, each successful forward action towards a boundary cell will increase the size of the relative map by 2 rows and 2 columns, so as to keep the relative origin at the centre of the map.

1.2.2. Flow of programme

In order to maximise flexibility, the driver will let the user choose whether the absolute map is pre-set, randomly generated by the driver, or manually constructed by the user.

Subsequently, the driver will let the user choose one of two game modes, where one executes manual movements controlled by the user, while the other executes automatic movement controlled by the agent's exploration ability as determined by *explore/1*. Manual movement allows the user to test the correctness of entering portal as well as game reset, as due to the nature of *explore/1*, it is unable to step into a portal or wumpus's location.

1.2.3. Testing of Correctness

1.2.3.1. Relative Position

In order to test that the agent correctly represents its relative position, the driver also keeps track of the agent's relative position, by updating it after every *reposition/1* or after every successful *moveforward*, *turnleft* or *turnright* action. The position returned by *current/3* must match this maintained position after every call of *move/2* in order for the agent to pass the test.

1.2.3.2. Evaluating the Knowledge Base

In order to test the correctness of the agent's sensory inference, the driver – after each call of *move/2* – queries **everything** within the agent's knowledge base relating to localisation and mapping, including (but not limited to) *visited/2, wumpus/2, safe/2* and *glitter/2*. In order for the agent to pass the test, the following must be adhered to:

i) All cells returned by *wall/2* must contain a wall.
ii) All cells returned by *tingle/2* must have a tingle.
iii) All cells returned by *glitter/2* must contain a coin.
iv) All cells returned by *stench/2* must have a stench.
v) All cells returned by *visited/2* must have been marked as visited by the driver before.
vi) All cells returned by *wumpus/2* must have at least one neighbouring cell with a stench.
vii) All cells returned by *confundus/2* must have at least one neighbouring cell with a tingle.
viii) All cells returned by *safe/2* must contain neither a wumpus nor a confundus portal.

The driver also applies the above tests to the current relative position of the agent. Additionally, if the last move executed was a successful *pickup*, the driver also checks if *glitter/2* for that cell now returns false.

1.2.3.3. Repositioning of Agent

In the event that the agent is repositioned, the driver checks – in a manner similar to that mentioned in 1.2.3.2 – that the agent's knowledge base has been cleared and now only reflects the initial set of sensory inputs. If the repositioning happens because of a reborn, the driver checks that *hasarrow/0* returns true. Otherwise, it checks that *hasarrow/0* remains unchanged.

1.2.3.4. Exploration

For every list of actions returned by *explore/1*, the driver executes a simulated run of the actions and fails the agent if any of the following holds true:

i) The agent enters a cell inhabited by the wumpus
ii) The agent enters a cell inhabited by a confundus portal
iii) The agent hits a wall
iv) The final cell in the simulation has already been visited (except the relative origin)

If the simulated run is done successfully, the driver calls *move/2* on the agent to execute the list of actions. Once the agent completes the list of actions, the driver will query *explore/1* for the next set of actions.

In the event that *explore/1* returns false or just an empty list, the driver performs a breadth-first-search (BFS) to confirm that there is no more safe, unvisited, accessible cell left, and fails the driver if such a cell exists.

2. CONCLUSION

Through this assignment, our group has learnt and understood – to some extent – the power and capabilities of logical reasoning with the assistance of the Prolog logic programming language. We have learnt how to represent and manipulate a knowledge base using Prolog. With sound logical reasoning, the agent was able to correctly identify safe cells, and potentially unsafe cells.

3. CONTRIBUTIONS OF TEAM MEMBERS

Task	Done By
Driver.py	Koh Jia Cheng
Agent.pl	Muqaffa Al-Afham Bin Kamaruzaman
Report	Koh Jia Cheng, Muqaffa Al-Afham Bin Kamaruzaman

4. REFERENCES

Tutorials Point. (n.d.). Retrieved from Prolog - Different and Not: https://www.tutorialspoint.com/prolog/prolog_different_and_not.ht

Wielemaker, J. (n.d.). *SWI Prolog*. Retrieved from Predicate assert/1: https://www.swi-prolog.org/pldoc/man?predicate=assert/1