**The Working Pattern for the Agent**

The agent works as the following pseudocode:

At an square:

Add\_perception\_facts\_to\_KB()

if glitter perceived in this square

Pick\_gold() // This method will pick gold if there is one.

Neighbours = calculate\_neighbours()

Push unexplored neighbours to the unexploredList

Do\_inference() //If safe & unexplored square is found, push it

//to the unexploredList

Choose a safe square to move onto:

1. The unexploredList is empty:

Switch to “Search for goal” mode.

1. Else:

--Pick the first square in unexploredList which is safe.

* 1. If this square is adjacent to the position now: Move to it directly
  2. Else: Use dijkstra algorithm to search the shortest safe path to get there.

--If it cannot locate a safe square, see if the wumpus is still alive:

a) The wumpus is alive, if killing the wumpus can explore more area, kill it.

If not, search ends and switch to the “search for goal” mode.

b)The wumpus is dead, search ends and switch to the “search for goal” mode.

Search for goal mode:

Use dijkstra algorithm to search the shortest safe path to get there.

**Data Structures**

1. World:

This structure is used by the referee and the agent to hold the information they learn about the wumpus world. The referee can know all the information of this world.

1. Referee:

This structure is used by the referee to store the world and communicate with the agent.

1. Agent:

This structure is used by the agent to store the information it gathers and there are methods for the agent to use to do inference, communicate with the referee and find paths.

**Engineering the Rules**

1. Adjacency rules:

Because prover9 does not support functions returns values other than boolean values, the agent will generate Adjacent(s,r) for each square to represent the adjacency in this world.

1. Because there are two kinds of rules in FOL,

“Diagonostic rules” and “Causal rules”, I create two rules for deducing pits:

all x exists y (B(x) -> (Adjacent(x,y) & P(y))). [Diagonostic rule]

all x all y ((P(x) & Adjacent(x,y)) -> B(y)). [Causal rule]

The same for the wumpus.

1. I’m afraid those two rules are not enough, so I add:

all x all y ((-B(x) & Adjacent(x,y)) -> -P(y)).

all x all y ((-S(x) & Adjacent(x,y)) -> -W(y)).

After adding these two rules, I think it’s enough to infer a whether square is safe or not.

1. Finally, a square is safe iff you can prove that it is not a pit and not a wumpus.

So: all x ((-W(x) & -P(x)) <-> Safe(x)).

**FOL rules used**

all s all r (Adjacent(s,r) <-> Adjacent(r,s)).

all x ((-W(x) & -P(x)) <-> Safe(x)).

all x (-Safe(x) <-> (W(x) | P(x))).

all x exists y (B(x) -> (Adjacent(x,y) & P(y))).

all x exists y (S(x) -> (Adjacent(x,y) & W(y))).

all x all y ((-B(x) & Adjacent(x,y)) -> -P(y)).

all x all y ((-S(x) & Adjacent(x,y)) -> -W(y)).

all x all y ((P(x) & Adjacent(x,y)) -> B(y)).

all x all y ((W(x) & Adjacent(x,y)) -> S(y)).

But there is a problem: These rules cannot deduce the position of the wumpus. So the inference of wumpus is done by the agent itself (inside the agent’s method start\_exploration()).

When there are no more safe square can move on and the wumpus is still alive, the agent will search for safe squares, which has two stench neighbours, and it can tell that the wumpus is at the diagonal opposite side.

After the wumpus is located, if killing the wumpus can enlarge searching area, then kill it.

**Experiment**

In the buggy version 2.0, the agent luckily passed map1 and map2 given by the TA, because both of these two maps don’t need to locate the wumpus and kill it.

But the world “wumpus\_world\_fixed.txt” under the directory “wumpus\_worlds” can reveal the bug, the world is described as follow:

M44

A11

B21

P31

B41

B32

G23

B23

P33

B43

B34

P44

GO11

G32

W22

S21

S12

S23

S32

There are just slight differences from the map2.txt. There is one more piece of gold, at (3,2) and the wumpus is moved to (2,2), so the stench will differ as well.

Because the agent, using the rules I defined in “init\_rule.txt” under the directory “source code”,cannot deduce the location of the wumpus so it cannot infer that the square (1,2) is safe. Thus, the only thing it can do now is to end search.

Here is the process of the buggy version 2.0 exploring this world:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Start the game\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Now the agent is exploring for gold.

At (1,1).

Move to (2,1).

Stench in (2,1).

Breeze in (2,1).

At (2,1).

Stench in (2,1).

Breeze in (2,1).

Move to (1,1).

At (1,1).

Move to (1,2).

Stench in (1,2).

At (1,2).

Stench in (1,2).

Search for gold ends.

Move to (1,1).

Final Score: -4

Total Gold Number: 2

Remain Gold Number: 2

Remain Arrow Number: 1

Total Moves: 4

**The Method to Fix the Bug**

1. Because I have no way to use FOL to deduce the position of the wumpus, I add another method to the agent:

cal\_wumpus\_pos(self, square, stenchSet),

which is able to return the position of the wumpus, given the square who has two stench neighbours(passed as stenchSet).

1. And also, repetitive facts can cause failure in prover9, so I revise the add\_perception\_facts(stench,breeze,glitter) to avoid adding repetitive facts into self.fact.