

جامعة الأخوين

جامعة الأخوين

AL AKHAWAYN
UNIVERSITY

Project #2:

Wumpus game

By : Imane Chtaini

Supervised by: Dr Tajjedine Rachidi

I. Introduction:

For this project, my teammate Souleiman Oulmaati and I (Imane Chtaini) tried to put the concept learnt in class into practice and make it real. For this matter, we used the language prolog to implement the wampus game, which is a well-known game. We can add that this game 'the wampus world' is one example to actually see the representation of a knowledge base and the Knowledge base-agent.

A knowledge-based agent is an agent that uses logic to represent knowledge. A knowledge base agent is an intelligent agent by process of reasoning that operates on the representation of knowledge. A knowledge base is the key component of a knowledge base agent.

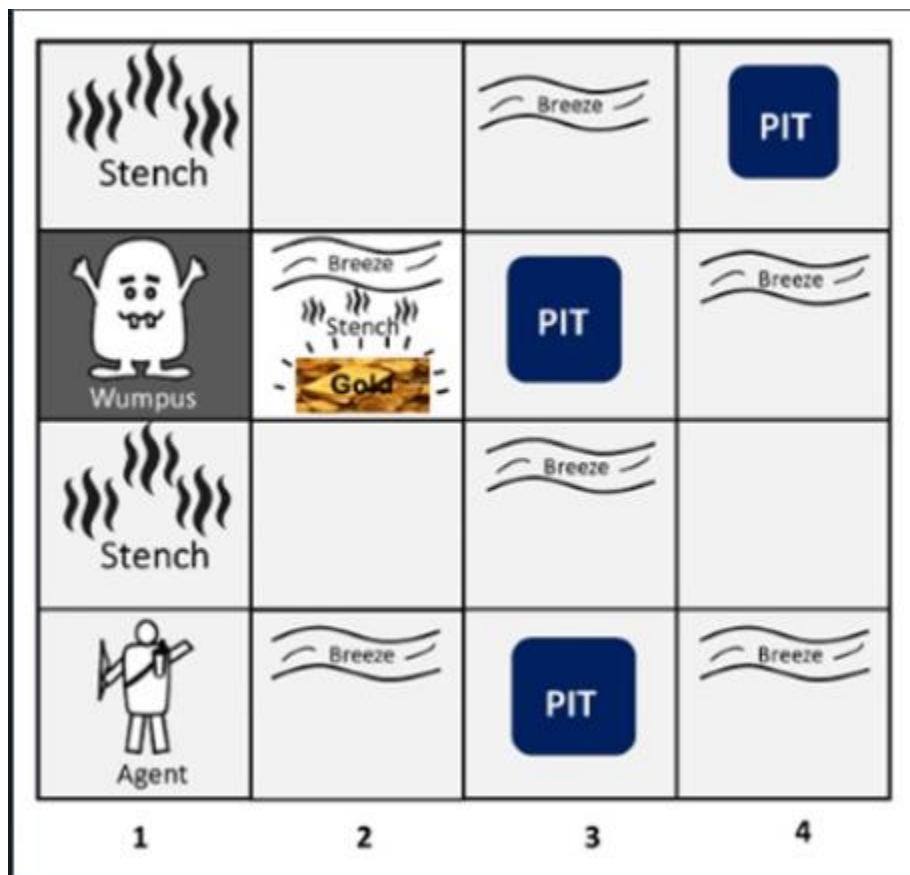
For a project we were asked to create or build a single iteration of a logical agent, given a starting position and asked to do a single action (check rooms' safety, grabbing gold, kill wampus...). Our agent that we created will only be able to answer questions asked by a human being given a set of rules.

For this matter, we tested or tried about 3 configurations on our program, in order to evaluate its performance.

➔ We implemented our project using prolog language via the SWISH prolog software.

II. Analysis or the Wampus game:

- As known rules of this game, we know that if a room's adjacent rooms are stench (smelly) it means that this room contains a wampus.
- The hunter can only shoot one single arrow only
- If a room is adjacent, cells are breezes, which leans that there is a pit there.
- If a room contains glitter it means that that room has gold
- ➔ These rules gives us a way to predict the probability of knowing if the next move of the hinter is safe or not. Which means that we will be able to know if the room can contain a pit or a wampus and pay attention to it In order to grab or retrieve the gold and win the game.



1. Predicates used to know the position of hunter, safe room, stench, breeze, stench, wampus, pit:

mypositionlist([[]]). -> To know the location of the starting position.

mysaferooms([[]]).-> to know the safe rooms that don't contain a pit or a wampus.

mygoldlist([[]]).-> to know the exact position of the gold.

mybreezelist([[]]).-> to know where the breezes are.

mystenchlist([[]]).-> to know where are the locations of different stench.

mywumpusloc([[]]).-> to know where the wampus is.

mypitlist([[]]).-> to know the rooms that contains a pit.

2. Predicates used to get all possible position for safe rooms, gold, stench, breeze and pits:

listofsaferooms(X)-> to know the list of safe rooms that don't contain a pit or a wampus.

gold(X)-> to know the list of positions of the gold.

breeze(X)-> to know the list of breezes.

stench(X)-> to know the list of stench.

wumpus(X)-> to know where the wampus .

pit(X)-> to know the rooms that contains a pit.

➔ One of the important predicate to know or predict the probability of knowing if a room contains a pit or wampus is by using the adjacent room predicate:

adjacenthuntercell([X, Y], [A, B]).-> give the adjacent room of the hunter's current position, and see if 2 rooms are adjacent.

➔ In order to win the game or grab the gold we need to know if a room contains gold or not hence the following predicate:

grabgold(A,B) -> return True if it's possible to grab the gold from its position (the hunter is in the gold room).

shoot(X,Y) -> basically this predicate the localization of the wampus, and shoots a arrow (since the agent can only shoot once) in order to kill the wampus. If we hear a scream then the wampus is dead if not then the wampus is still alive.

3. *Predicates that check if there is a stench, breeze, pit, wampus in a given room:*

doesitstench(A,B) -> this predicate see if there is a stench.

doesitcontainwampus(A,B) -> see if there is a wampus.

doesitcontainbreeze(A,B) -> see if there is a breeze in the position (A,B).

doesitcontainpit(A,B) -> see if there is a pit in the room.

isthisroomsafe(X,Y) -> see if the room is safe or not.

➔ To help our code or case we added the predicate **same([], [])**:

Which checks if 2 lists are equal or not it was mainly used in safe rooms, breeze....

III. Configurations used:

For this part, we run or tried all predicates on 3 different configuration as shown in the screenshots below:

➤ Configuration 1:

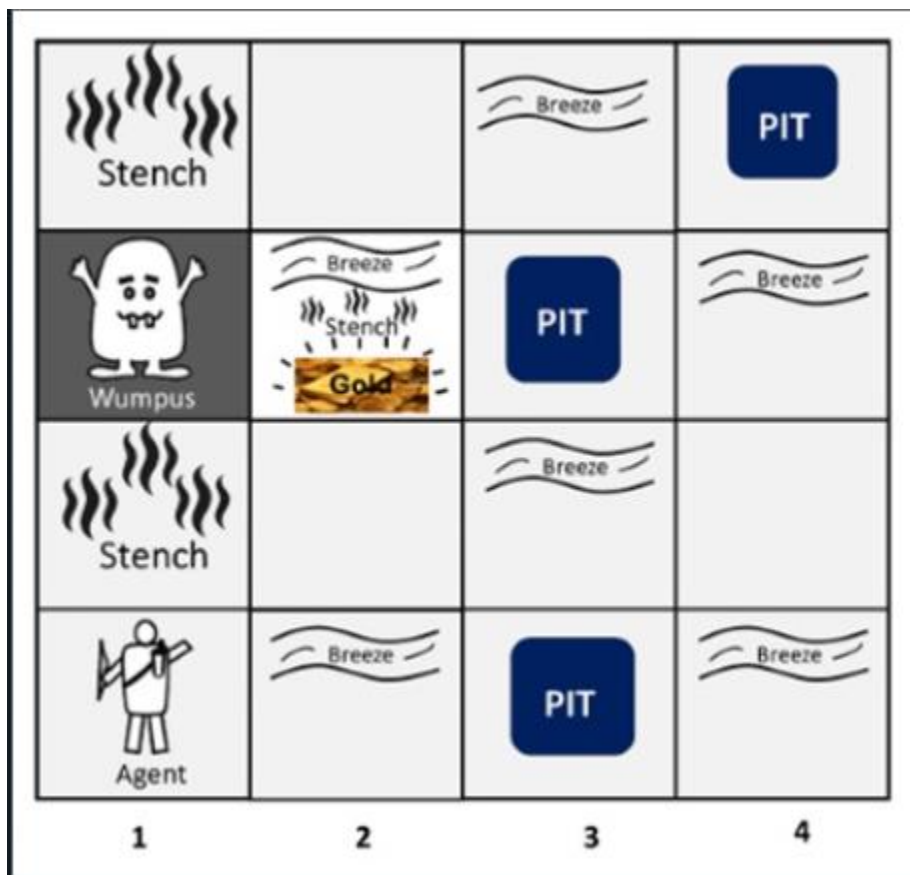
For this configuration the hunter starting position is at [1,1]

Stench: [1,4], [1,2]

Breeze: [2,1], [2,3], [3,4], [3,2], [4,1], [4,3]

Pit: [3,1], [3,3], [4,4]

Gold: [2,3]



➔ So we implemented this configuration using prolog in the SWISH prolog software:

```
%location of starting position
mypositionlist([[1,1]]).
%location of gold
mygoldlist([[3,2]]).
%location of stench
mystenchlist([[1,2],[1,4]]).
%location of wumpus
mywumpusloc([[1,3]]).
%location breeze
mybreezelist([[2,1],[3,2],[4,1],[3,4],[4,3]]).
%safe rooms
mysaferooms([[1,4],[1,2],[1,1],[2,1],[2,2],[2,3],[2,4],[3,2],[3,4],[4,1],[4,2],[4,3]]).
%list of rooms where there is a pits
mypitlist([[3,1],[3,3],[4,4]]).
```

1. We implemented the current position predicate to know the starting position or the current position of the hunter which is [1,1]:



2. We implemented the predicate to get the list of rooms where there is a breeze:

```
⚙ breeze(X).
breeze here
X = [2, 1]
X = [3, 2]
X = [4, 1]
X = [3, 4]
X = [4, 3]

?- breeze(X).
```

3. We implemented a predicate that only tell us if a given room contains a breeze or not. As in the examples below for [1,4] and [4,3]:

```
⚙ doesitcontainbreeze(1,4).
false

?- doesitcontainbreeze(1,4).
```

```
⚙ doesitcontainbreeze(4,3).
true

?- doesitcontainbreeze(4,3).
```

4. This predicate gives us the list of rooms that contain pits:

```
⚙ pit(X).
pit here
X = [3, 1]
X = [3, 3]
X = [4, 4]

?- pit(X).
```


5. This predicates gives us if a given room contains a pit:

```
doesitcontainpit(4,4).  
true  
?- doesitcontainpit(4,4).|
```

```
doesitcontainpit(1,4).  
false  
?- doesitcontainpit(1,4).|
```

6. This predicates gives us the position of the wampus:

```
wumpus(X).  
wumpus here  
X = [1, 3]  
?- wumpus(X).|
```

7. This predicates if in a given room there is a wampus or not:

```
doesitcontainwumpus(1,3).  
true  
?- doesitcontainwumpus(1,3).|
```

```
doesitcontainwumpus(2,3).  
false  
?- doesitcontainwumpus(2,3).|
```

8. Gives us where the stench locations are:

```
stench(X).  
stench here  
X = [1, 2]  
X = [1, 4]  
?- stench(X).
```

9. This predicate check is there is a stench in a given position or not. (

```
doesitstench(1,2).  
true
```

```
doesitstench(4,2).  
false  
?- doesitstench(4,2).
```

10. Give us all possible rooms where the gold is:

```
gold(X).  
gold here  
X = [3, 2]  
?- gold(X).
```

11. Then there the grab gold predicate it gives us true if we can grab the gold and false if we can't:

```
grabgoldgold(1,2).  
false  
?- grabgoldgold(1,2).
```

```
⚙️ grabgoldgold(3,2).  
true  
?- grabgoldgold(3,2).
```

12. This predicate check if room are adjacent to each other or not:

```
⚙️ adjacenthuntercell([3,2],[4,2]).  
true  
⚙️ adjacenthuntercell([2,1],[2,3]).  
false
```

13. This predicate see if a room is safe or not:

```
⚙️ isthisroomsafe(1,3).  
false  
⚙️ isthisroomsafe(2,4).  
true
```

14. This predicate gives us the list of safe rooms in this configuration:

```
listofsaferooms(X).  
list of safe rooms:  
X = [1, 4]  
X = [1, 2]  
X = [1, 1]  
X = [2, 1]  
X = [2, 2]  
X = [2, 3]  
X = [2, 4]  
X = [3, 2]  
X = [3, 4]  
X = [4, 1]  
X = [4, 2]  
X = [4, 3]
```

15. This predicate see if we can actual kill the wampus in a certain position and say if the wampus screamed or not. Since if a wampus screamed it means that the wampus is dead. If no scream heard then the wampus is still alive. So for this predicate it will return True if he wampus was killed and also if he screamed. False if the wampus is not dead.

```
shoot(1,2).  
wumpus was killed. Loud scream!!!!  
true
```

```
shoot(3,2).  
false
```

ANALYSIS:

As mentioned earlier to predict if a room contains a pit, wampus or gold we need to check the adjacent rooms.

For this case, we have breezes in the following positions: [2,1], [2,3], [3,4], [3,2], [4,1], [4,3]

- We see that the cells [2,1], [3,2], [4,1] are 3 cells that share the adjacency with one cell which is [3,1].
 - ➔ We can conclude that this room probably contain a pit. Something that was proven by our code, since we got the same answer that this room contains a pit.

We see that the cells [2,3], [3,4], [3,2], [4,3] are 4 cells that share the adjacency with one cell which is [3,3].

- ➔ We can conclude that this room probably contain a pit. Something that was proven by our code, since we got the same answer that this room contains a pit.

=>For the first case of k knowing if a room contains a pit or no we can say that our agent was efficient and correct in this configuration.

➤ Configuration 2:

For this configuration we:

Hunter: [1,1]

Gold : [1,4]


Wumpus: [4,4]

Pit: [1,3], [2,3], [3,2],[4,2]

GOLD			WAMPUS
PIT	PIT		
		PIT	PIT
HUNTER			

1. Current position predicate:


```

 currentpos(X).
current position:
X = [1, 1]

```

2. Breeze list predicate:

```

 breeze(X).
breeze here
X = [1, 2]
X = [1, 4]
X = [2, 2]
X = [2, 4]
X = [3, 1]
X = [3, 3]
X = [4, 1]
X = [4, 3]

```




3. If the room contain a breeze predicate:


 <code>doesitcontainbreeze(2,4).</code>
true
 <code>doesitcontainbreeze(1,1).</code>
false
 <code>doesitcontainbreeze(3,4).</code>
false


4. List of pits predicate:

 <code>pit(X).</code>
pit here
X = [1, 3]
X = [2, 3]
X = [3, 2]
X = [4, 2]

5. Check if the room contain pit predicate:

 <code>doesitcontainpit(1,3).</code>
true
 <code>doesitcontainpit(2,3).</code>
true
 <code>doesitcontainpit(3,2).</code>
true


 `doesitcontainpit(4,2).`
true

 `doesitcontainpit(1,4).`
false

6. Wumpus predicate:

 `wumpus(X).`
wumpus here
X = [4, 4]


7. See if there is a wumpus in the room predicate:

 `doesitcontainwumpus(4,4).`
true


8. List of stench predicate:

 `stench(X).`
stench here
X = [3, 4]
X = [4, 3]


9. See if there is a stench predicate:

 `doesitstench(3,4).`


true

 `doesitstench(4,3).`


true

 `doesitstench(1,1).`


false

 `doesitstench(1,2).`


false

 `doesitstench(1,3).`

false


 `doesitstench(2,3).`

false


 `doesitstench(2,4).`

false


10. Gold predicate:

 <code>gold(X).</code>
<code>gold here</code>
<code>X = [1, 4]</code>


11. Then grabgoldgold predicate:

 `grabgoldgold(1,4).`


true

 `grabgoldgold(1,1).`

false

 `grabgoldgold(1,2).`

false

 `grabgoldgold(1,3).`

false


 `grabgoldgold(3,3).`

false

 `grabgoldgold(4,3).`

false


12. Adjacent hunter cell predicate:

 `adjacenthuntercell([3,3],[3,2]).`


true

 `adjacenthuntercell([3,1],[3,2]).`


true

 `adjacenthuntercell([1,1],[2,1]).`

true

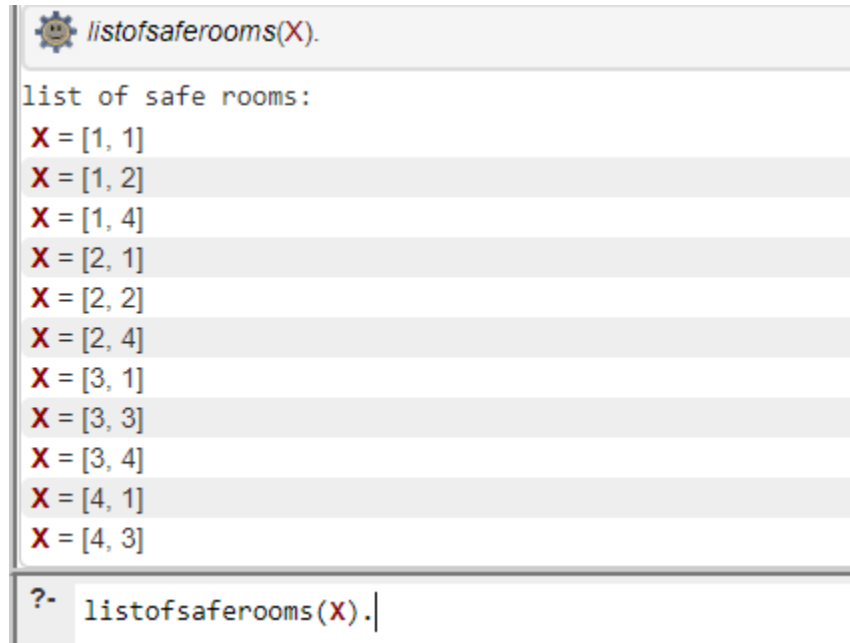
 `adjacenthuntercell([1,1],[2,2]).`

false

 `adjacenthuntercell([1,1],[2,4]).`

false







13. List of safe rooms predicate:



```
listofsaferooms(X).  
list of safe rooms:  
X = [1, 1]  
X = [1, 2]  
X = [1, 4]  
X = [2, 1]  
X = [2, 2]  
X = [2, 4]  
X = [3, 1]  
X = [3, 3]  
X = [3, 4]  
X = [4, 1]  
X = [4, 3]  
?- listofsaferooms(X).|
```

The image shows a Prolog IDE window. At the top, there is a gear icon followed by the predicate `listofsaferooms(X).`. Below this, the text "list of safe rooms:" is displayed. A list of 12 pairs of values for `X` is shown, each on a new line and preceded by a red `X`. The pairs are: `[1, 1]`, `[1, 2]`, `[1, 4]`, `[2, 1]`, `[2, 2]`, `[2, 4]`, `[3, 1]`, `[3, 3]`, `[3, 4]`, `[4, 1]`, and `[4, 3]`. At the bottom of the window, there is a prompt `?-` followed by `listofsaferooms(X).` and a cursor.

14. Check is a room is safe predicate:

 <i>isthisroomsafe(1,1).</i>
true
 <i>isthisroomsafe(2,1).</i>
true
 <i>isthisroomsafe(2,4).</i>
true
 <i>isthisroomsafe(4,1).</i>
true
 <i>isthisroomsafe(3,2).</i>
false
 <i>isthisroomsafe(4,4).</i>
false

15. Shoot predicate:

```
⚙️ shoot(3,4).
wumpus was killed. Loud scream!!!!
true

⚙️ shoot(4,3).
wumpus was killed. Loud scream!!!!
true

⚙️ shoot(1,3).
false

⚙️ shoot(1,2).
false

⚙️ shoot(1,3).
false

⚙️ shoot(3,3).
false

?- shoot(3,3).
```

ANALYSIS:

For this configuration we already know where the pit, gold, breeze, and hinter are so to rate the performance of our agent in this case we will see if the logic behind what he found as rooms for breeze, and stench are correct.

- Stenches:
 - ➔ Since we have the wumpus in room [4,4] we can say that its adjacent room contain stenches. In this case the rooms are: [4,3] and [3,4]
- Pit:
 - ➔ We know that when there is a pit in a room [X,Y] its adjacent rooms contain breezes for this case they are: [1,2], [3,1],[4,1],[4,2],[3,3],[2,2],[1,2],[2,4]
- We also found that there is a glitter in room [1,4]

=>After seeing this result we can notice that they are the same as the one that our agent got so we can say that in this case our agent was also efficient and correct.

➤ **Configuration 3:**

For the third configuration we:

Hunter: [1,1]

Gold : [4,1]


Wumpus: [3,2]

Pit: [1,3], [4,3].

PIT			PIT
		wampus	
HUNTER			GOLD

1. Current position predicate:


```

 currentpos(X).
current position:
X = [1, 1]

```




2. List of breezes predicate:

```


 breeze(X).
breeze here
X = [1, 2]
X = [1, 4]
X = [2, 3]
X = [3, 3]
X = [4, 2]
X = [4, 4]

```




3. See if there is a breeze predicate:

```
 doesitcontainbreeze(1,4).  
true  
 doesitcontainbreeze(4,2).  
true  
 doesitcontainbreeze(3,1).  
false
```

4. List of pits predicate:

```
 pit(X).  
pit here  
X = [1, 3]  
X = [4, 3]
```






5. See if a room contains a pit predicate

```
 doesitcontainpit(1,3).  
true  
 doesitcontainpit(4,3).  
true  
 doesitcontainpit(4,4).  
false  
?- doesitcontainpit(4,4|).
```

6. Wumpus predicate:


```
 wumpus(X).  
wumpus here  
X = [3, 2]  
?- wumpus(X).
```

7. Check is a current room contains a wumpus predicate:

```
 doesitcontainwumpus(3,2).  
true  
 doesitcontainwumpus(1,2).  
false  
 doesitcontainwumpus(1,3).  
false  
 doesitcontainwumpus(4,3).  
false  
 doesitcontainwumpus(4,4).  
false  
?- doesitcontainwumpus(4,4).
```


8. List of stench predicate:

```
stench(X).  
stench here  
X = [2, 2]  
X = [3, 1]  
X = [3, 3]  
X = [4, 2]  
?- stench(X).
```

9. See if a current room contains a stench predicate:

```
doesitstench(2,2).  
true  
doesitstench(3,1).  
true  
doesitstench(3,3).  
true  
doesitstench(4,2).  
true  
doesitstench(4,4).  
false  
doesitstench(4,1).  
false
```

10. Gold predicate:

```
 gold(X).  
gold here  
X = [4, 1]
```

11. Grabgoldgold predicate:

```
 grabgoldgold(4,1).  
true  
 grabgoldgold(4,2).  
false  
 grabgoldgold(4,3).  
false  
 grabgoldgold(4,4).  
false  
 grabgoldgold(2,2).  
false  
 grabgoldgold(2,3).  
false
```




12. Adjacenthuntercell predicate:

```
adjacenthuntercell([1,2],[1,3]).  
true  
adjacenthuntercell([1,2],[1,4]).  
false  
?- adjacenthuntercell([1,2],[1,4]).
```





13. List of safe rooms predicate:

```
listofsaferooms(X).  
list of safe rooms:  
X = [1, 4]  
X = [1, 2]  
X = [1, 1]  
X = [2, 1]  
X = [2, 2]  
X = [2, 3]  
X = [2, 4]  
X = [3, 2]  
X = [3, 4]  
X = [4, 1]  
X = [4, 2]  
X = [4, 3]
```

14. See if a current room is safe predicate:

 <code>isthisroomsafe(1,1).</code>
true
 <code>isthisroomsafe(1,2).</code>
true
 <code>isthisroomsafe(1,3).</code>
false

15. Shoot predicate:

 <code>shoot(3,1).</code>
wumpus was killed. Loud scream!!!!
true
 <code>shoot(2,2).</code>
wumpus was killed. Loud scream!!!!
true
 <code>shoot(3,3).</code>
wumpus was killed. Loud scream!!!!
true
 <code>shoot(4,2).</code>
wumpus was killed. Loud scream!!!!
true



ANALYSIS:

For this configuration we also already know where the pit, gold, breeze, and hunter are, so to rate the performance of our agent in this case we will see if the logic behind what he found as rooms for breeze, and stench are correct.

- Stenches:
 - ➔ Since we have the wampus in room [3,2] we can say that its adjacent room contain stenches. In this case the rooms are: [3,1], [2,2], [4,2], [2,3]
- Pit:
 - ➔ We know that when there is a pit in a room [X,Y] its adjacent rooms contain breezes for this case they are: [4,2], [3,3],[4,4],[1,2],[1,4],[2,3]
- We also found that there is a glitter in room [4,1]

=>After seeing this result we can notice that they are the same as the one that our agent got so we can say that in this case our agent was also efficient and correct.

IV. Conclusion:

After trying these 3 configuration, we can say that our agent was efficient and right in all its answer. Its performance was good in these different situations. We can say that our logical agent worked well and did well to represent its knowledge base or world. Our logical agent used sensors perceived from the environment well since he answered the questions about where the pit, breeze, stench, wampus and gold are correctly.

V. Limitations of our solution:

For the limitation of our code me and my teammate souleiman Oulmaati: is first the count of how many time the hunter used an arrow to attempt to kill the wampus. In the rules of the game, the hunter can only use his arrow once no more than one time. In our code, this was one of the limitations since the hunter can use more than one arrow in his attempts

In addition to that, we do not have a score counter that counts the scores of the hunter or the performance measures of the hunter as in the game:

- ➔ +1000 reward points if the agent comes out of the cave with the gold.
- ➔ -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- ➔ One for each action, and -10 for using an arrow.
- ➔ The game ends if either agent dies or came out of the cave.

In addition to that, we did not set a rule for when the agent is dead the game is over