Project 2: Wumpus World

CSC 4301: Introduction to Artificial Intelligence

Souleiman Oulmaati

Supervised by : Dr. Rachidi Tajjeeddine

Introduction:

In this project, my teammate (Imane Chtaini) and I (Souleiman Oulmaati) applied the concepts learnt during class and our programming background to implement a program about Wumpus world. We used prolog as it is the most efficient language related to answering questions about the eventualities of performing some actions using logic. The rules of this game are simple. The agent is on a starting position at the beginning, typically ([1,1]), and ideally, it should try to grab the gold and kill the Wumpus. The agent should not be in a room where there is a pit or the Wumpus in order to not die. In our project, we were asked to implement a program where it answers questions related to performing some specific actions. Also, the Wumpus is static. We tested our program with 3 different configurations. We were able to set a program that answer many questions showing if it is possible or not; however, we still encountered some limitations that we could not fix. In the following report we will explain the key predicates and the meaning of the variables, display some snapshots of various experiments, and discuss some of the limitations that make our code fail. Our project was implemented using prolog language through the SWISH prolog software.

A. Key Predicates, and the meaning of the Variables:

location of starting position: **mypositionlist([[]]).**

location of gold: **mygoldlist([[]]).**

location of stench: **mystenchlist([[]]).**

location of Wumpus: **mywumpusloc([[]]).**

location breeze: **mybreezelist([[]]).**

safe rooms: **mysaferooms([[]]).**

list of rooms where there is a pit: **mypitlist([[]]).**

-Predicates to print lists of saferooms,gold,stenches,wumpus, breezes, pits

listofsaferooms(X)-gold(X)-stench(X) -wumpus(X) -breeze(X) -pit(X)

-Predicate that checks whether two lists are identical or not(it returns true if yes and false if not)

same([], []).

-Predicate that takes 2 inputs (two different rooms to verify whether the two rooms are adjacent or not if the rooms are adjacent, it returns true. If not, it returns false)

adjacenthuntercell([X, Y], [A, B]).

The following predicates work if the predicate same([], []) returns true (for example we cannot grab gold if the input position is not the same as the position of the gold in the configuration)

-Predicate that test whether it it possible to grab gold from the position entered as an input(if the user enter the position where the gold is present it will return yes)

grabgoldgold(A,B)

-Predicate that verifies whether there is stench.

doesitstench(A,B)

-Predicate that verifies whether there is wumpus.

doesitcontainwumpus(A,B)

-Predicate that verifies whether there is breeze.

doesitcontainbreeze(A,B)

-Predicate that verifies whether there is a pit.

doesitcontainpit(A,B)

-Predicate that verifies whether it is possible to launch an arrow from a room entered as an input to kill the wumpus. This predicate contains another predicate inside of it which is adjacenthuntercell([X, Y], [A, B]) which compare the adjacency of the input room with the room of the Wumpus and It returns true and kill the wumpus if the adjacency is satisfied. Otherwise, it returns false and the Wumpus is not killed hence we don't hear a scream.

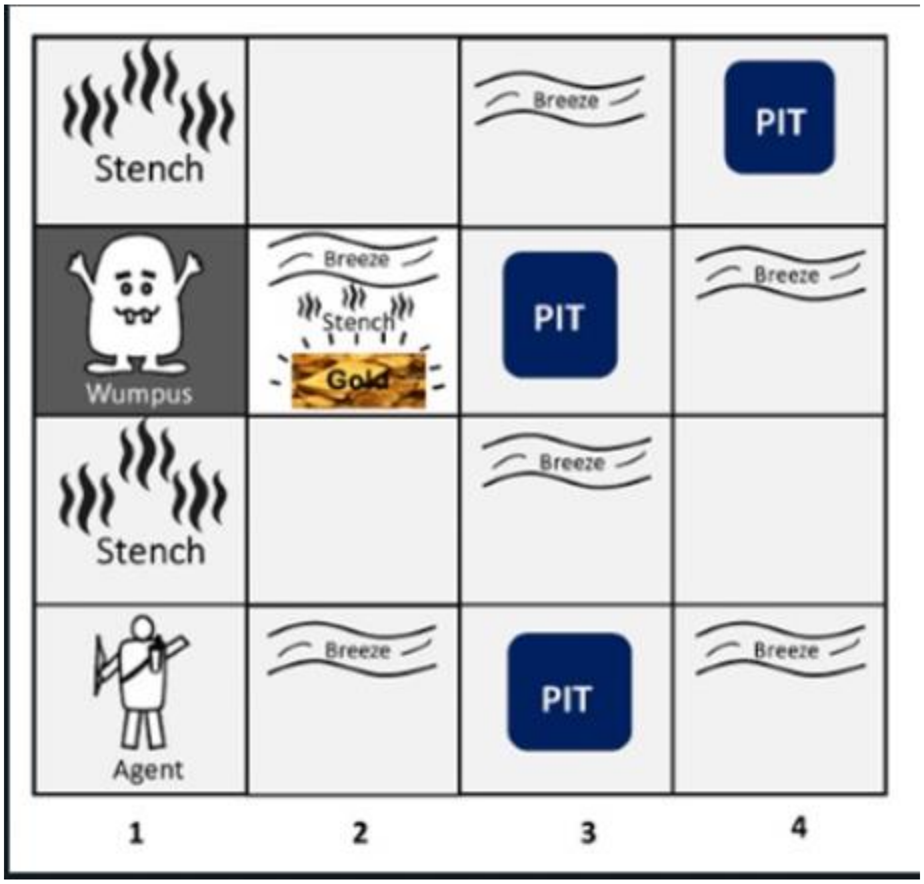
shoot(X,Y).

-The following predicate checks whether a room is safe(does not contain Wumpus or pit) or not.

isthisroomsafe(X,Y).

B. Snapshots

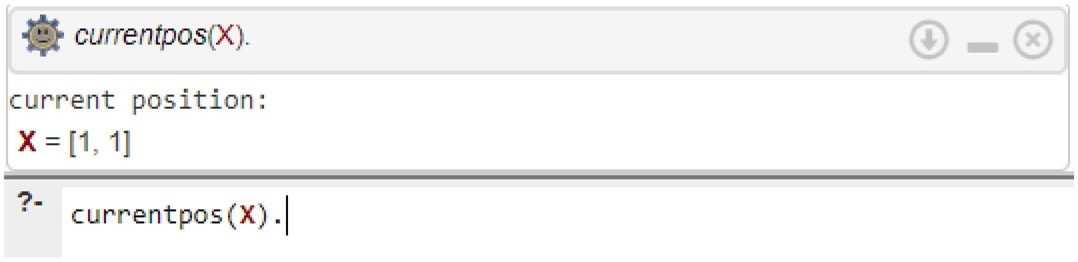
Config 1:



Here is its corresponding configuration

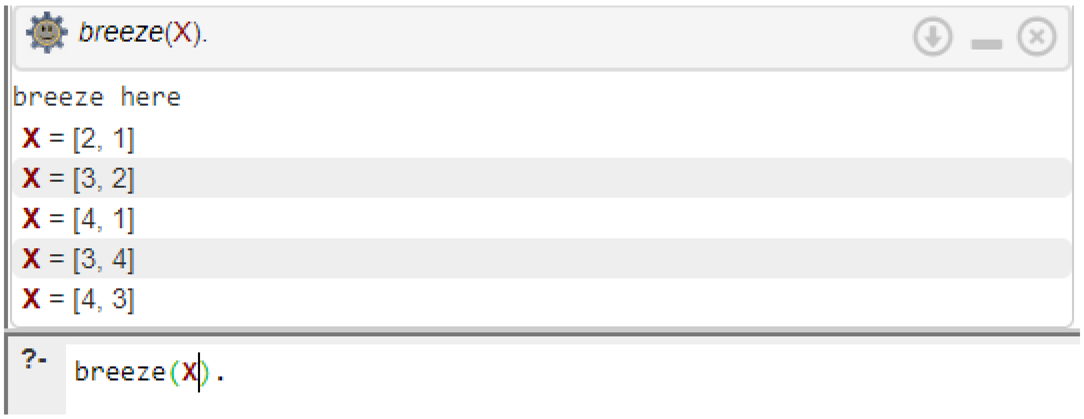
```
%location of starting position
mypositionlist([[1,1]]).
%location of gold
mygoldlist([[3,2]]).
%location of stench
mystenchlist([[1,2],[1,4]]).
%location of wumpus
mywumpusloc([[1,3]]).
%location breeze
mybreezelist([[2,1],[3,2],[4,1],[3,4],[4,3]]).
%safe roooms
mysaferooms([[1,4],[1,2],[1,1],[2,1],[2,2],[2,3],[2,4],[3,2],[3,4],[4,1],[4,2],[4,3]]).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

- Current position location:



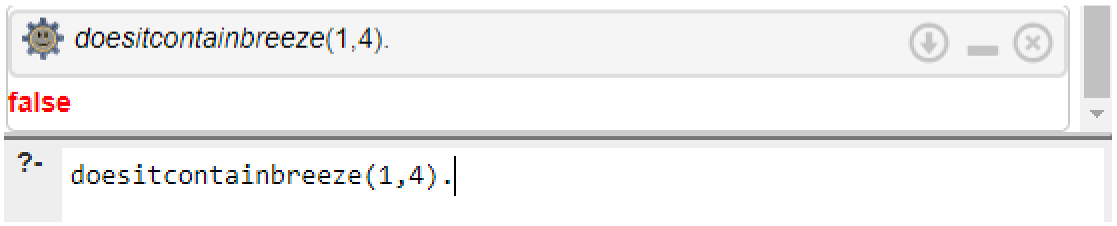
```
currentpos(X).
current position:
X = [1, 1]
?- currentpos(X).
```

- List of rooms where we have breeze:



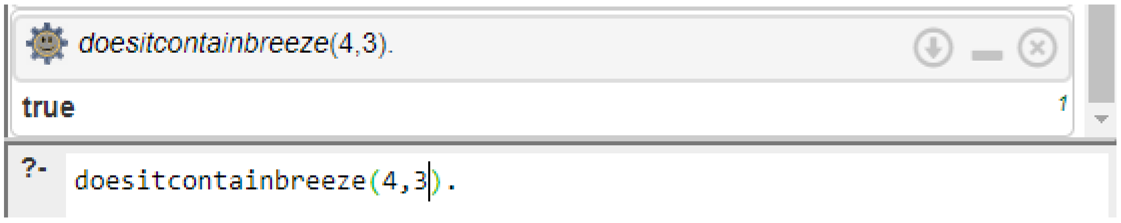
```
breeze(X).
breeze here
X = [2, 1]
X = [3, 2]
X = [4, 1]
X = [3, 4]
X = [4, 3]
?- breeze(X).
```

- There is also another predicate that takes as an input the position of a room and checks if there is a breeze there(returns true) or not (returns false)



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

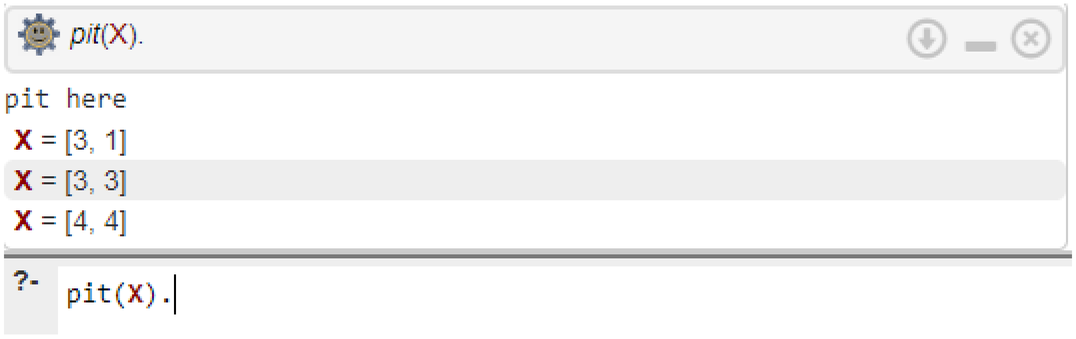
Indeed there is not breeze in (1,4)



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

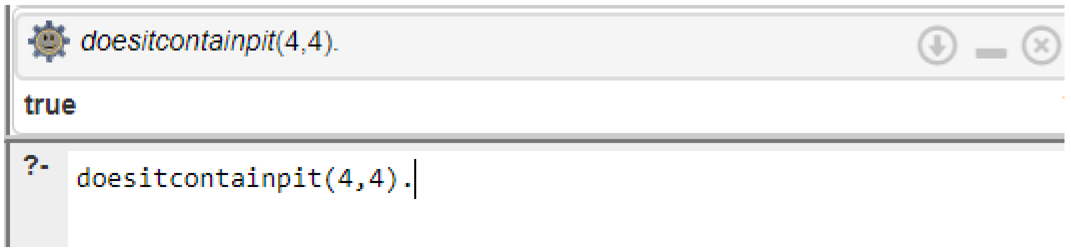
Indeed there is a breeze in (4,3)

- List of rooms where there is a pit.

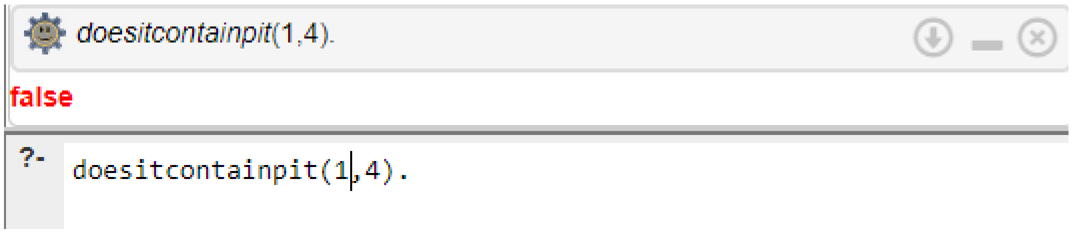


```
pit(X).
pit here
 X = [3, 1]
 X = [3, 3]
 X = [4, 4]
?- pit(X).
```

- Here is another predicate to see if there is a pit in a room with some given inputs

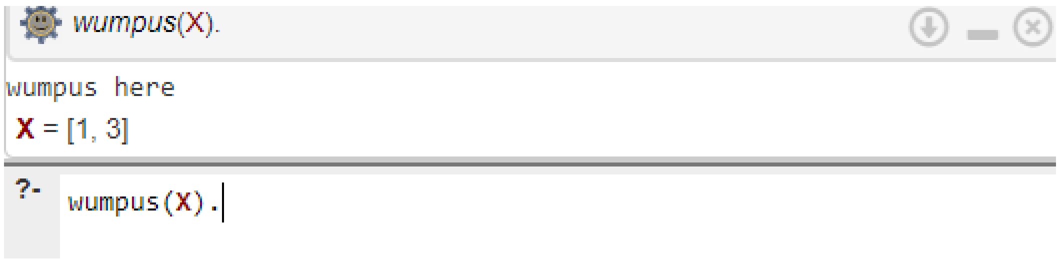


```
doesitcontainpit(4,4).
true
?- doesitcontainpit(4,4).
```



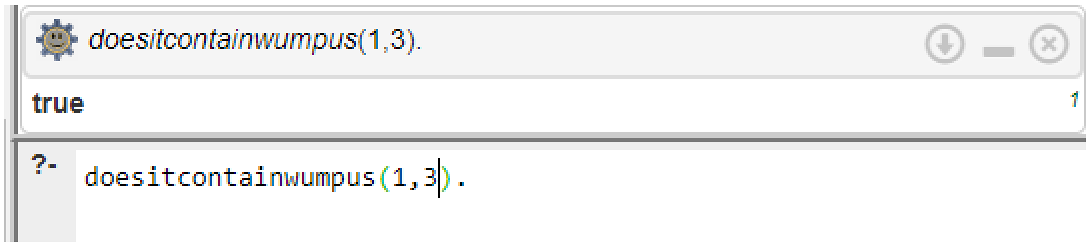
```
doesitcontainpit(1,4).
false
?- doesitcontainpit(1,4).
```

- List of rooms where there is the wumpus



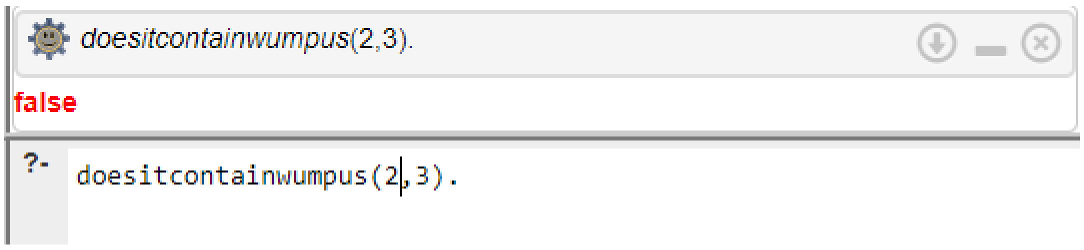
```
wumpus(X).
wumpus here
X = [1, 3]
?- wumpus(X).
```

- The other predicate checks if there is a Wumpus in a given room

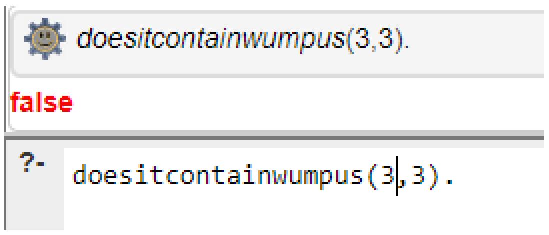


```
doesitcontainwumpus(1,3).
true
?- doesitcontainwumpus(1,3).
```

So any room besides [1,3] should give us false

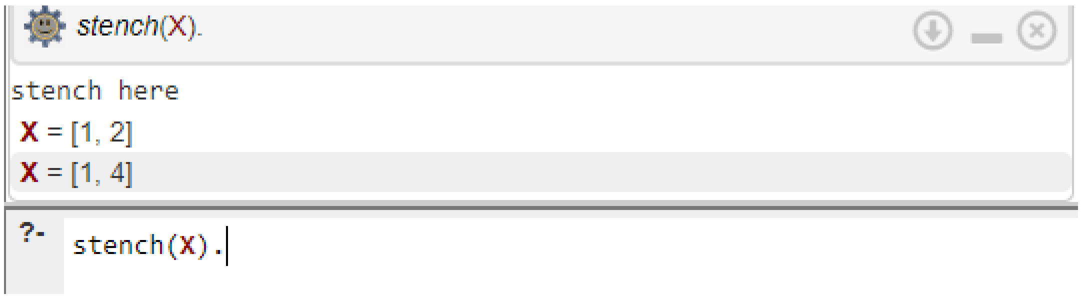


```
doesitcontainwumpus(2,3).
false
?- doesitcontainwumpus(2,3).
```



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

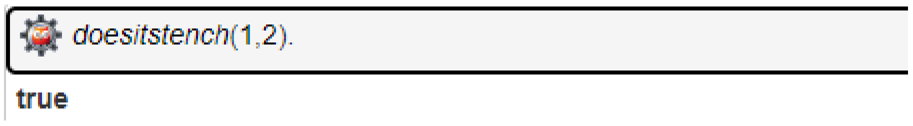
- List of rooms where there is stench



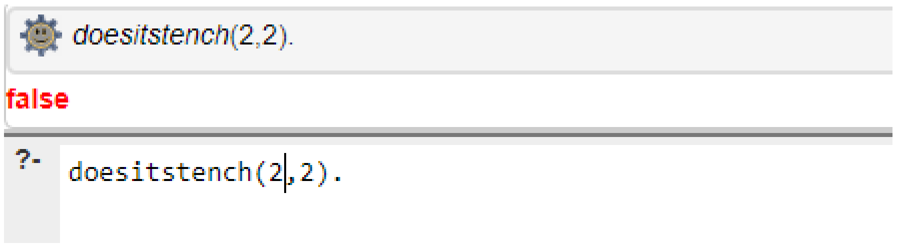
```
stench(X).
stench here
X = [1, 2]
X = [1, 4]
?- stench(X).
```

- Then the predicate doesitstench(A,B) checks if in a given room we have stench

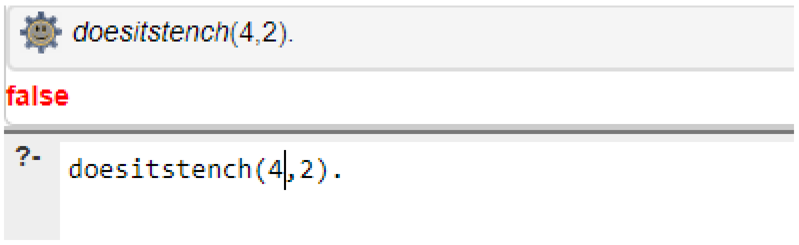
(any room besides [1,2] and [1,4] should return false)



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

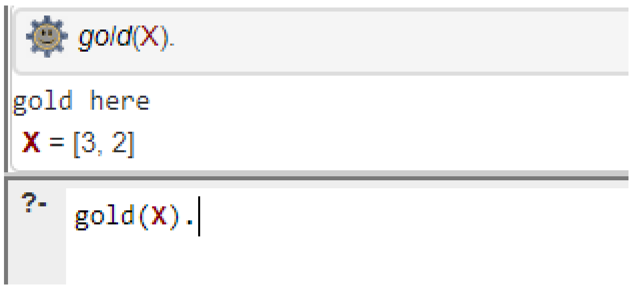


```
doesitstench(2,2).
false
?- doesitstench(2,2).
```



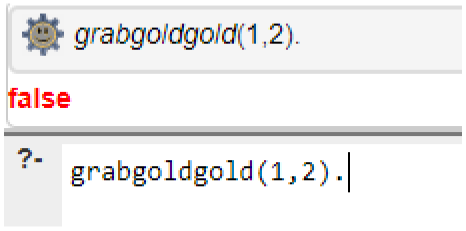
```
doesitstench(4,2).
false
?- doesitstench(4,2).
```

- Room location of the gold

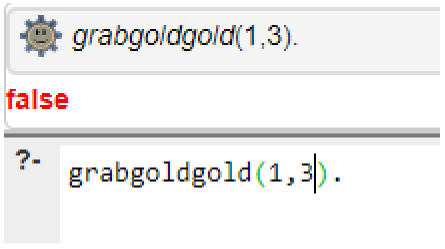


```
gold(X).
gold here
X = [3, 2]
?- gold(X).
```

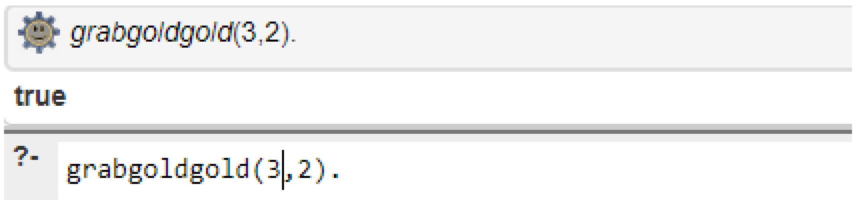
- Then grabgoldgold(A,B) checks if we can grab gold from a position



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



```
grabgoldgold(1,3).
false
?- grabgoldgold(1,3).
```



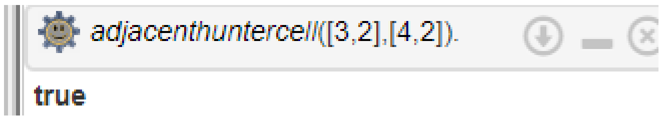
```
grabgoldgold(3,2).
true
?- grabgoldgold(3,2).
```

- AdjacentTo (R(X,Y), R(ZT)) room R(T,Z) is adjacent to room R(X,Y)

adjacenthuntercell([X, Y], [A, B]) checks if 2 rooms given as inputs are adjacent

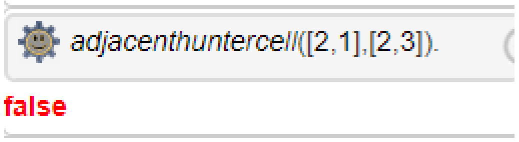
let's try some examples two of them are adjacents and the other is not

[3,2] and [4,2] will return true



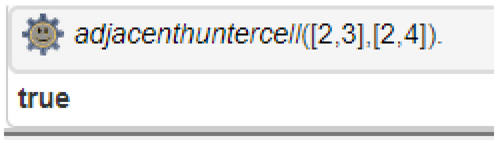
```
adjacenthuntercell([3,2],[4,2]).
true
```

[2,1] and [2,3] will return false



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

[2,3] and [2,4] will return true

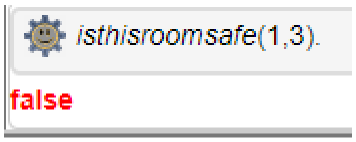


```
adjacenthuntercell([2,3],[2,4]).
true
```

- The predicate isthisroomsafe(X,Y) checks if a room with some given inputs is safe or not

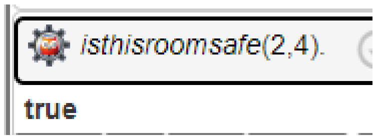
Let's try it with examples when one room is safe and the other is not.

In room [1,3] there is the Wumpus so it is not safe



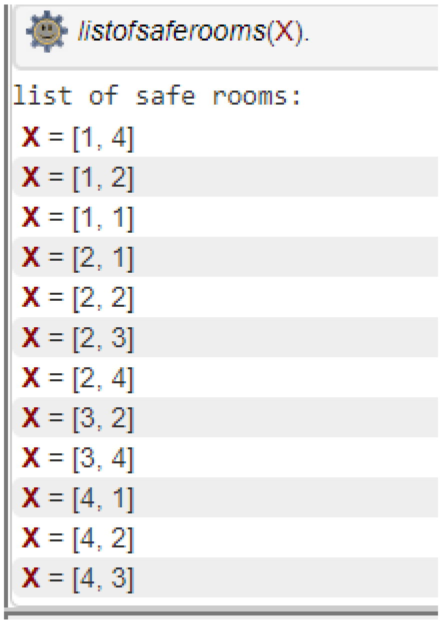
```
isthisroomsafe(1,3).
false
```

In the room [2,4] there is nothing so it should return true



```
isthisroomsafe(2,4).
true
```

- The predicate listofsaferooms(X) displays the rooms that are safe

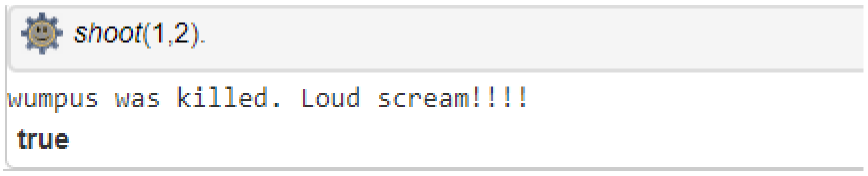


```
listofsaferooms(X).

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]
```

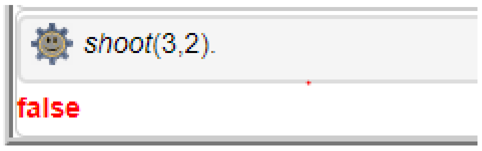
- the predicate shoot(X,Y) checks if we can shoot and kill the Wumpus from a room with coordinates [X,Y]

if the entered room is adjacent to the room where the wumpus is, we will hear a scream and it will return true. Otherwise, it will return false.

Let's do it with 2 different examples. In [1,2], it will return true, and in [3,2] it will return false since this room is not adjacent to the room where the Wumpus is.



```
shoot(1,2).

wumpus was killed. Loud scream!!!!
true
```



```
shoot(3,2).

false
```

Second Configuration:

Now let's try our code with another configuration

Let's try it with this one

Position of the starting agent: [1,1]

Gold room: [1,4]

Wumpus room: [4,4]

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

From this configuration we deduce that the stenches are on the following positions:

[3,4] && [4,3]

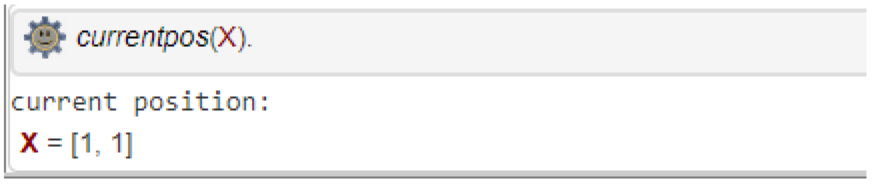
For the breeze rooms:

[1,2] && [1,4] && [2,2] && [2,4] && [3,1] && [3,3] && [4,1] && [4,3]

BREEZE+GOLD	BREEZE	STENCH	WUMPUS
PIT	PIT	BREEZE	STENCH+ BREEZE
BREEZE	BREEZE	PIT	PIT
AGENT		BREEZE	BREEZE

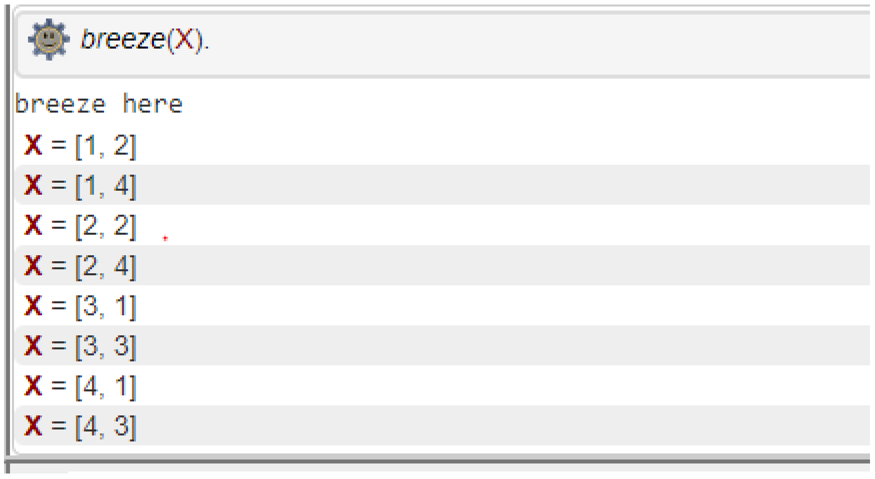
Let's use our code to answer questions related to this configuration

- Current position location:



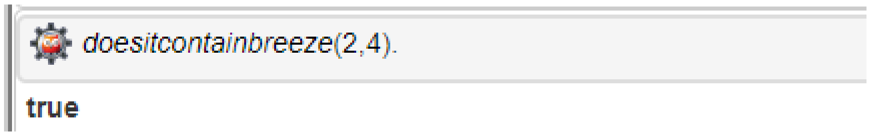
```
currentpos(X).
current position:
X = [1, 1]
```

- List of rooms where there is breeze:



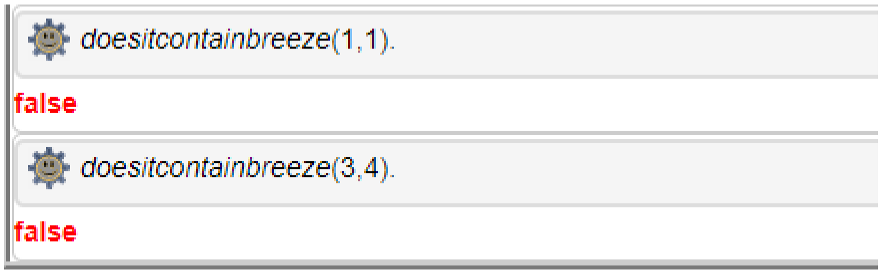
```
breeze(X).
breeze here
X = [1, 2]
X = [1, 4]
X = [2, 2]
X = [2, 4]
X = [3, 1]
X = [3, 3]
X = [4, 1]
X = [4, 3]
```

- There is also another predicate that takes as an input the position of a room and checks if there is a breeze there (returns true) or not (returns false)



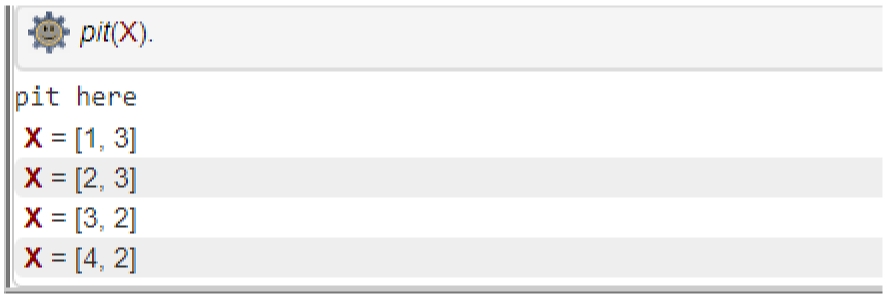
```
doesitcontainbreeze(2,4).
true
```

Now let's take 2 rooms where there is no breeze (it will return false)



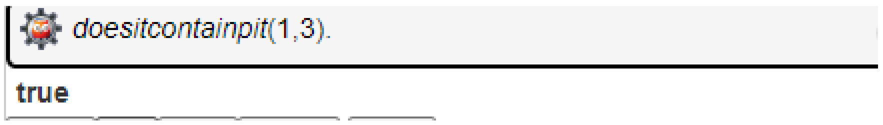
```
doesitcontainbreeze(1,1).
false
doesitcontainbreeze(3,4).
false
```

- List of rooms where there is a pit:

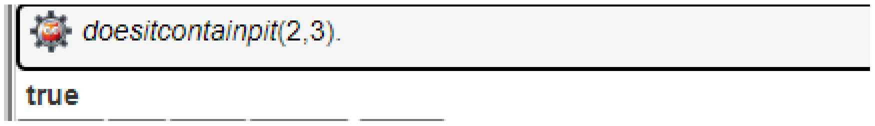


```
pit(X).
pit here
X = [1, 3]
X = [2, 3]
X = [3, 2]
X = [4, 2]
```

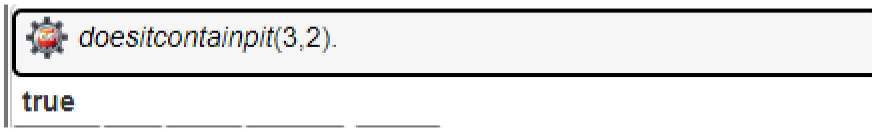
- Here is another predicate to see if there is a pit in a room with some entered inputs:



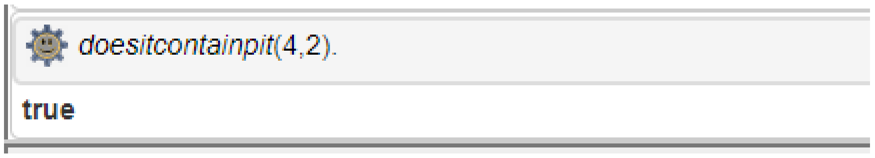
```
doesitcontainpit(1,3).
true
```



```
doesitcontainpit(2,3).
true
```



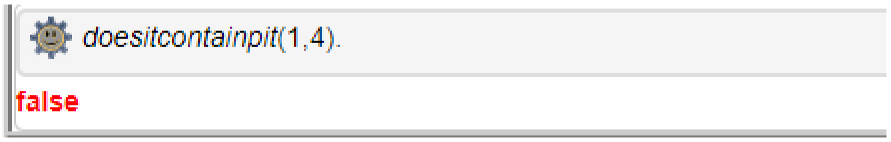
```
doesitcontainpit(3,2).
true
```



```
doesitcontainpit(4,2).
true
```

The results concord with our chosen set of pits

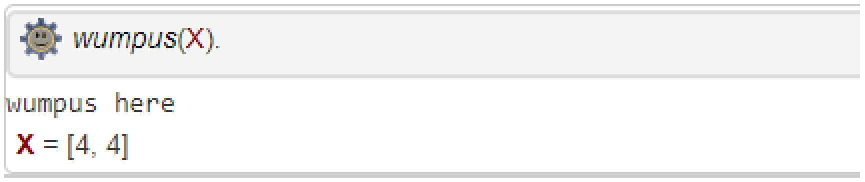
Now let’s try it with a room where there is not a pit



```
doesitcontainpit(1,4).
false
```

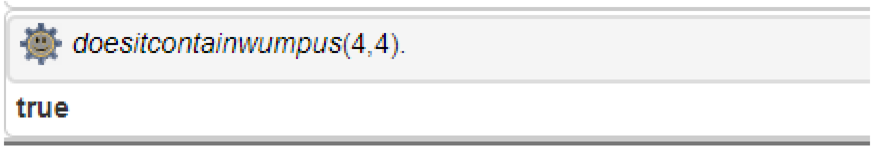
The predicate works well

- Wumpus position:



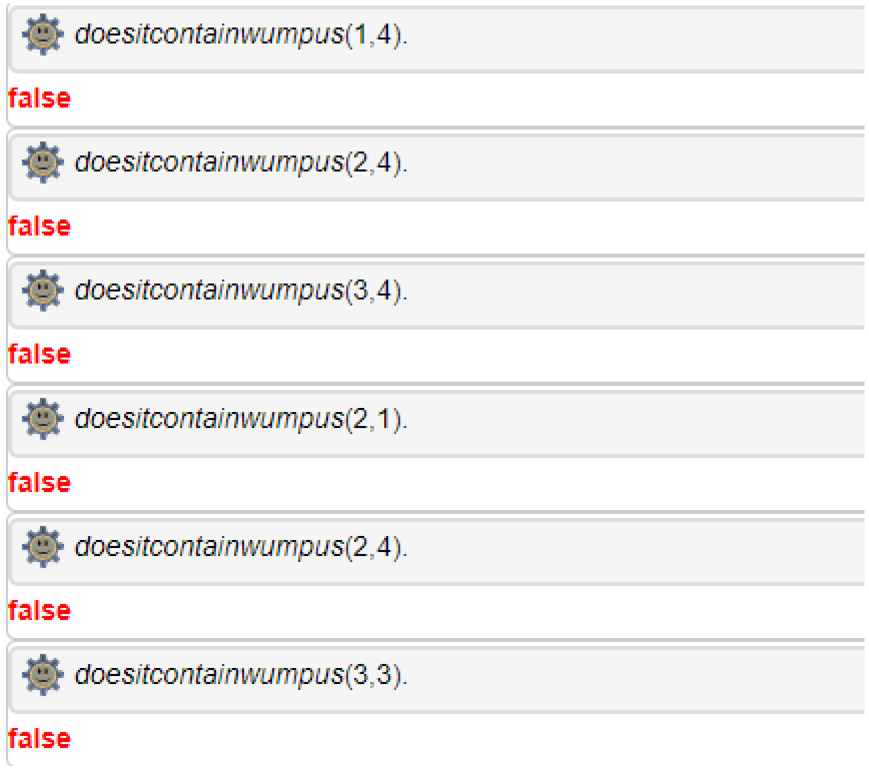
```
wumpus(X).
wumpus here
X = [4, 4]
```

- The other predicate checks if there is a Wumpus in the room entered as input



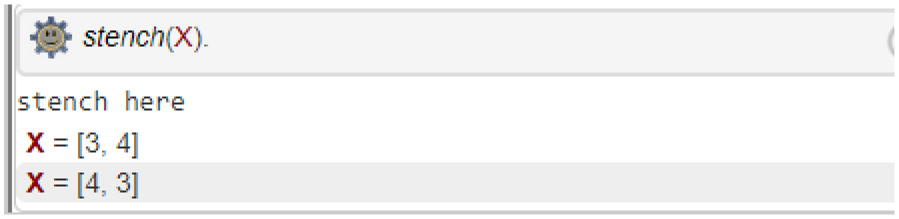
```
doesitcontainwumpus(4,4).
true
```

Now let's try it with some positions where there is no Wumpus



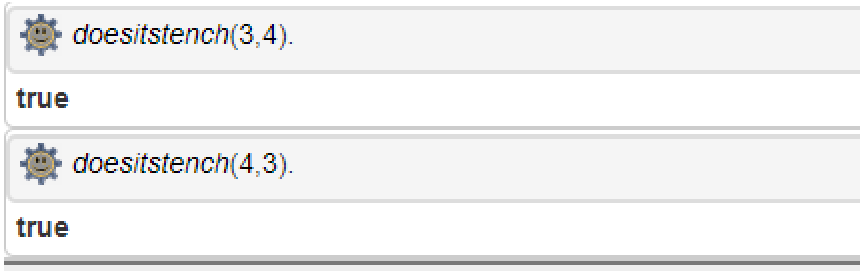
```
doesitcontainwumpus(1,4).
false
doesitcontainwumpus(2,4).
false
doesitcontainwumpus(3,4).
false
doesitcontainwumpus(2,1).
false
doesitcontainwumpus(2,4).
false
doesitcontainwumpus(3,3).
false
```

- List of rooms where there is stench:



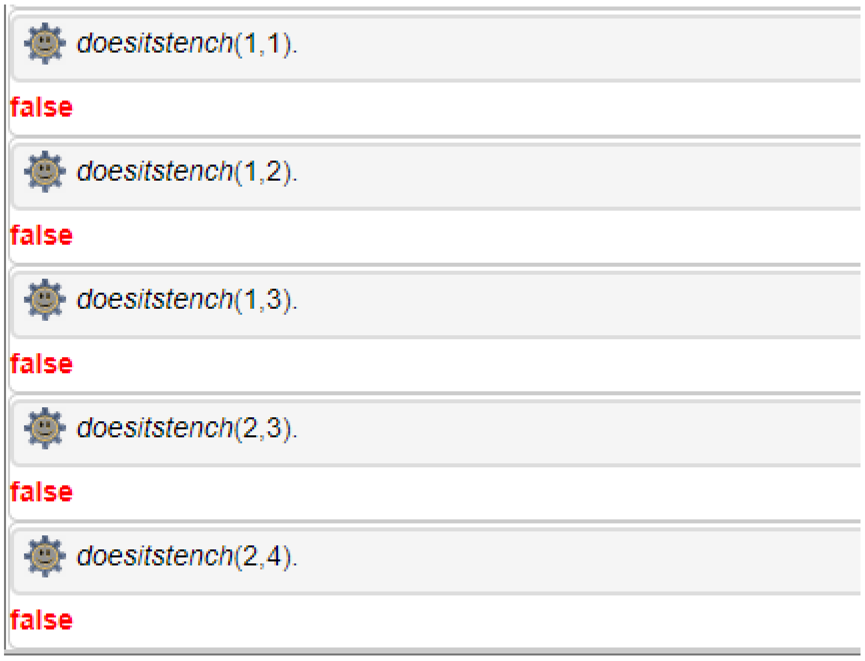
```
stench(X).
stench here
X = [3, 4]
X = [4, 3]
```

- Then the predicate doesitstench(A,B) checks if in a given room we have stench



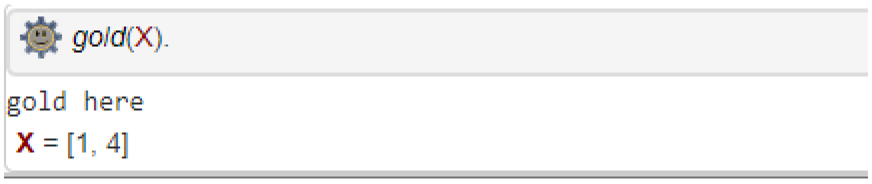
```
doesitstench(3,4).
true
doesitstench(4,3).
true
```

Then we try with some rooms where there is no stench

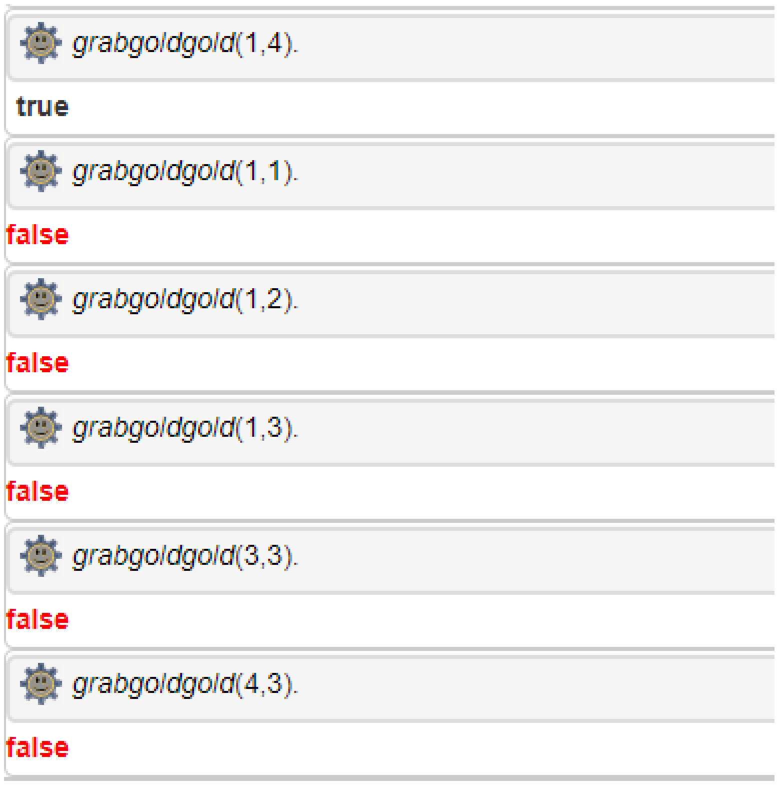


```
doesitstench(1,1).
false
doesitstench(1,2).
false
doesitstench(1,3).
false
doesitstench(2,3).
false
doesitstench(2,4).
false
```

- Gold location:


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

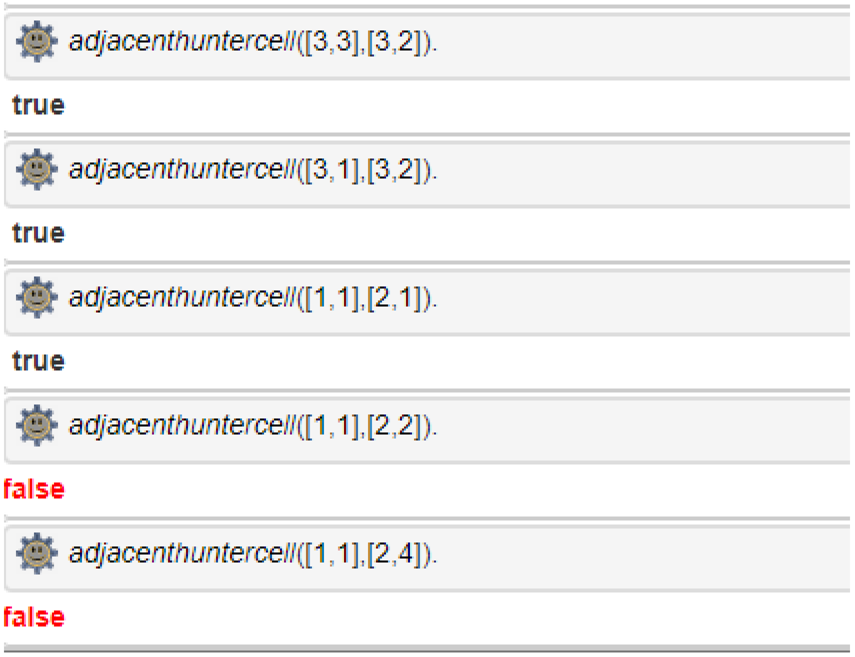
- Then grabgoldgold(A,B) checks if we can grab gold from the input position. It should return true if and only if the entered position is (1,4)


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

The results are correct

- AdjacentTo (R(X,Y), R(ZT)) room R(T,Z) is adjacent to room R(X,Y)

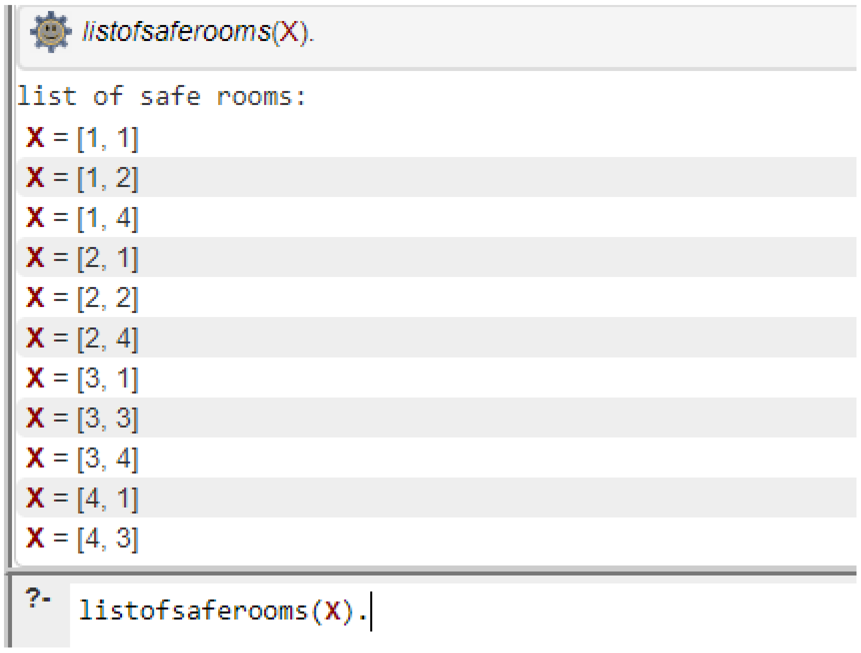
adjacenthuntercell([X, Y], [A, B]) checks if 2 rooms given as inputs are adjacent



```
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
```

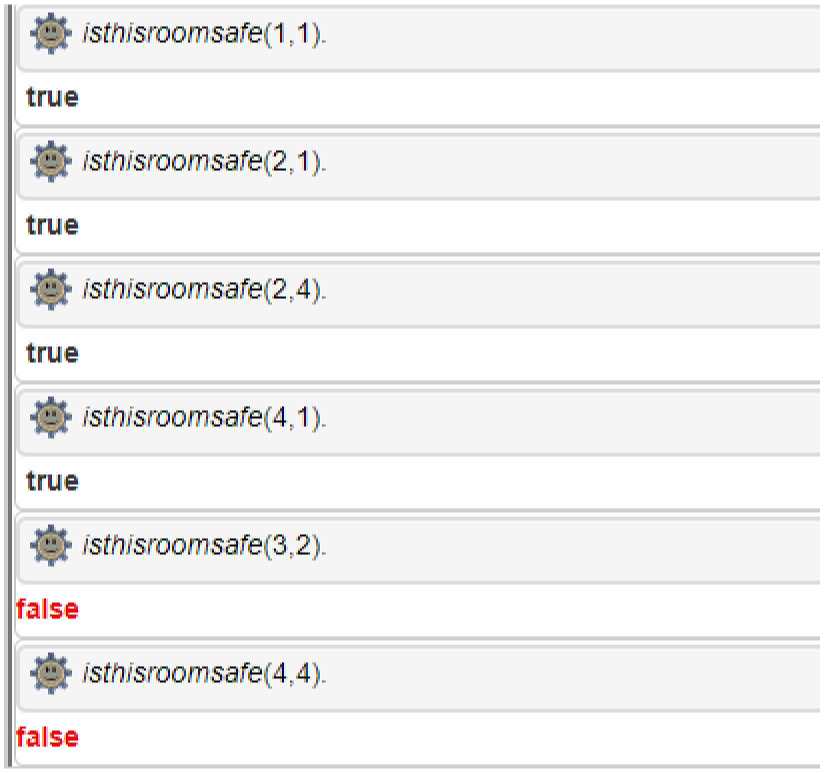
WORKS WELL

- List of safe rooms:



```
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 predicate isthisroomsafe(X,Y) checks if a room with some inputs is safe or not



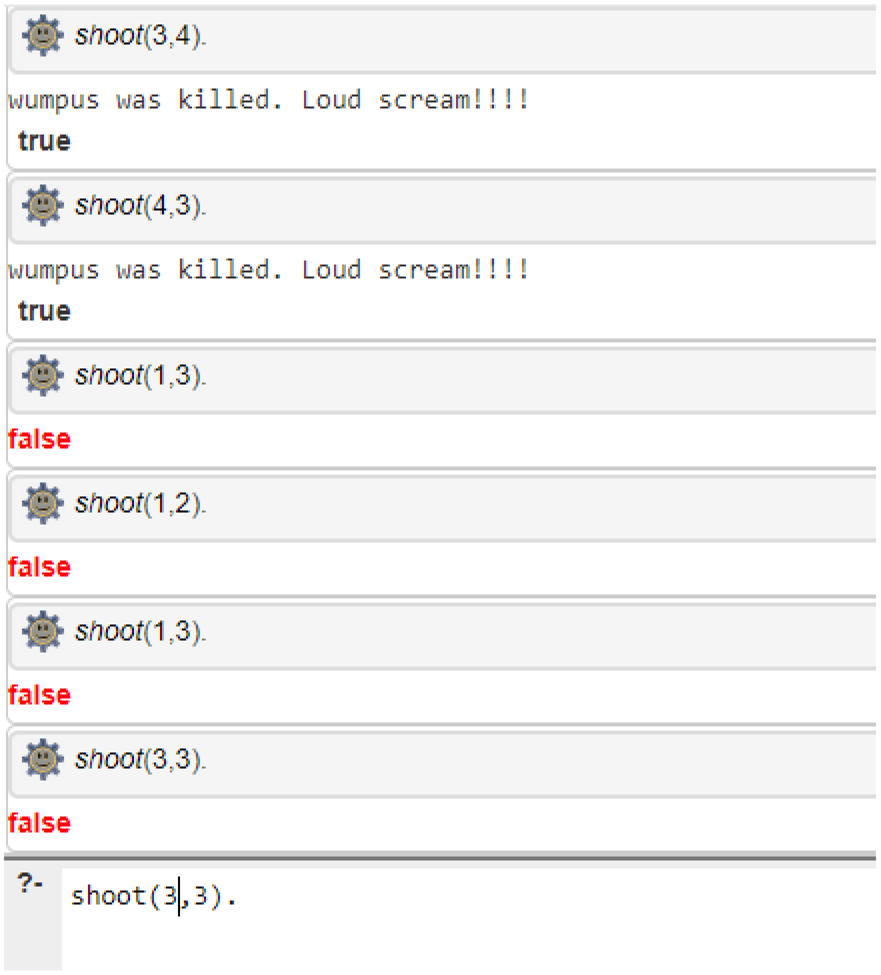
```
isthisroomsafe(1,1).
true
isthisroomsafe(2,1).
true
isthisroomsafe(2,4).
true
isthisroomsafe(4,1).
true
isthisroomsafe(3,2).
false
isthisroomsafe(4,4).
false
```

- the predicate shoot(X,Y) checks if we can shoot and kill the Wumpus from a room with coordinates [X,Y]

if the entered room is adjacent to the room where the wumpus is, we will hear a scream and it will return true. Otherwise, it will return false.

In our configuration the room from where the arrow will be launched should be in a room adjacent to [4,4] to hear the scream.

The rooms that satisfy this condition are the following: [3,4] and [4,3]



```
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).
```

The predicate works well.

Third configuration:

Now let’s try our code with another configuration

Let’s try it with this one

Position of the starting agent: [1,1]

Gold room: [4,1]

Wumpus room: [3,2]

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

From this configuration we deduce that the stenches are on the following positions:

[2,2] && [3,1] && [3,3] && [4,2]

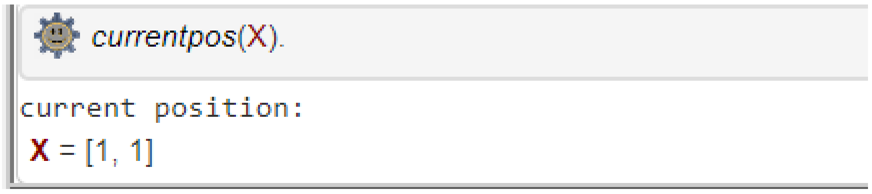
For the breeze rooms:

[1,2] && [1,4] && [2,3] && [3,3] && [4,2] && [4,4]

breeze			breeze
pit	breeze	Stench+breeze	pit
breeze	stench	wumpus	Stench+breeze
agent		stench	gold

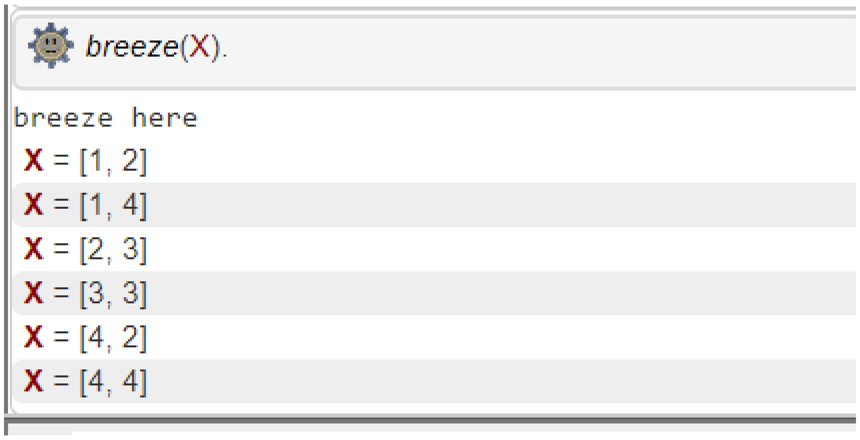
Let’s use our code to answer questions related to this configuration

- Current position location:



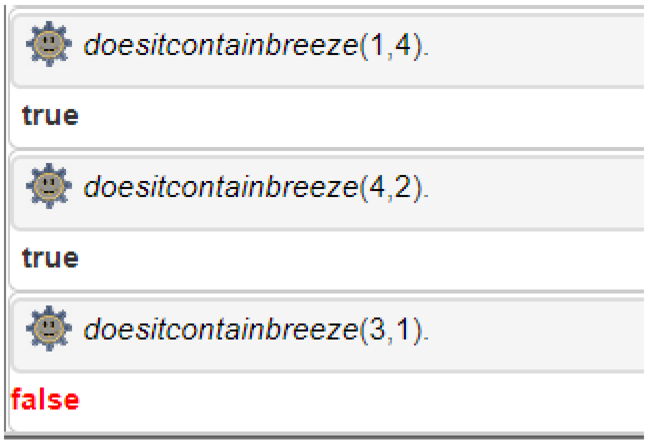
```
currentpos(X).

current position:
X = [1, 1]
```

- List of rooms where there is breeze:



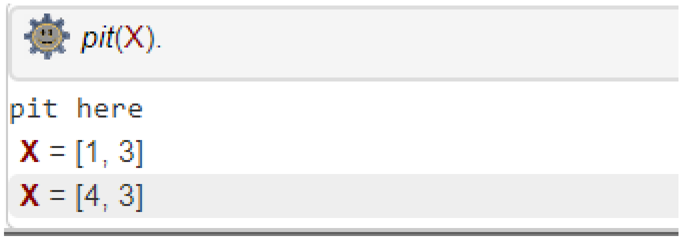
```
breeze(X).

breeze here
X = [1, 2]
X = [1, 4]
X = [2, 3]
X = [3, 3]
X = [4, 2]
X = [4, 4]
```

- There is also another predicate that takes as an input the position of a room and checks if there is a breeze there (returns true) or not (returns false)



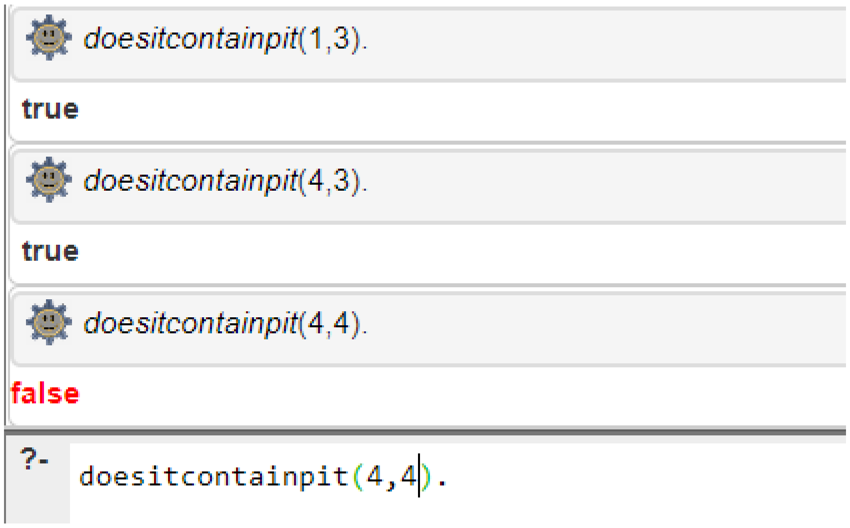
```
doesitcontainbreeze(1,4).

true

doesitcontainbreeze(4,2).

true

doesitcontainbreeze(3,1).

false
```

- List of rooms where there is a pit:



```
pit(X).

pit here
X = [1, 3]
X = [4, 3]
```

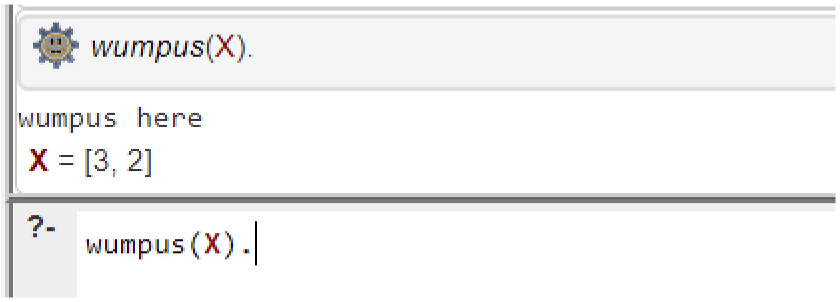
- Here is another predicate to see if there is a pit in a room with some entered inputs:



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

The predicate gives logical outputs.

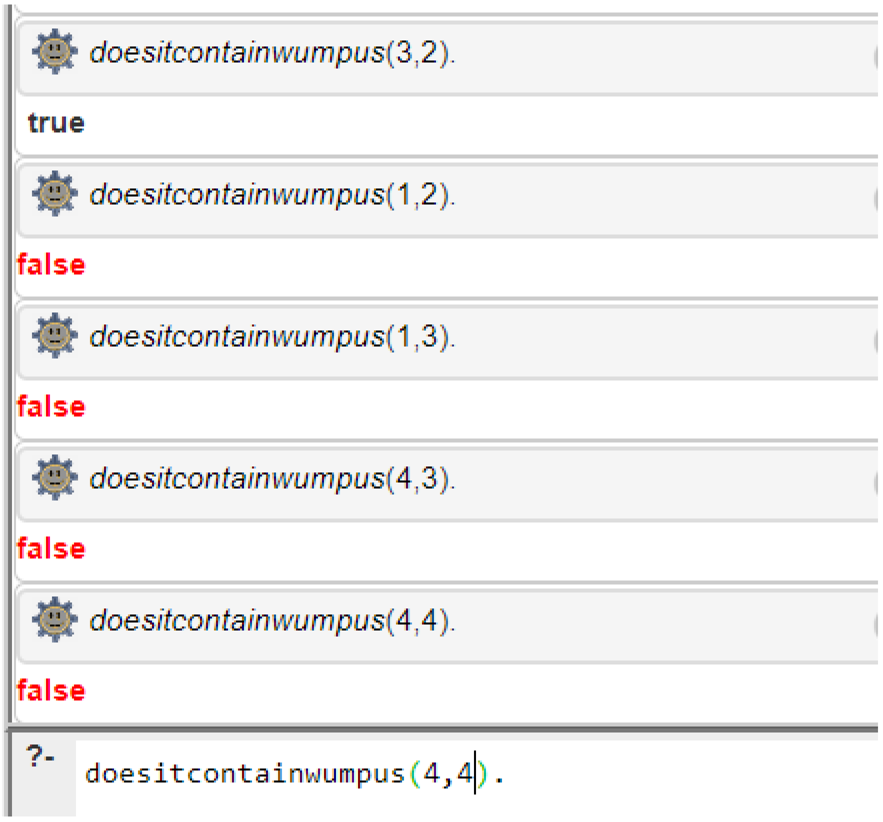
- Wumpus position:



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

- The other predicate(doesitcontainwumpus(A,B)) checks if there is a Wumpus in the room entered as input

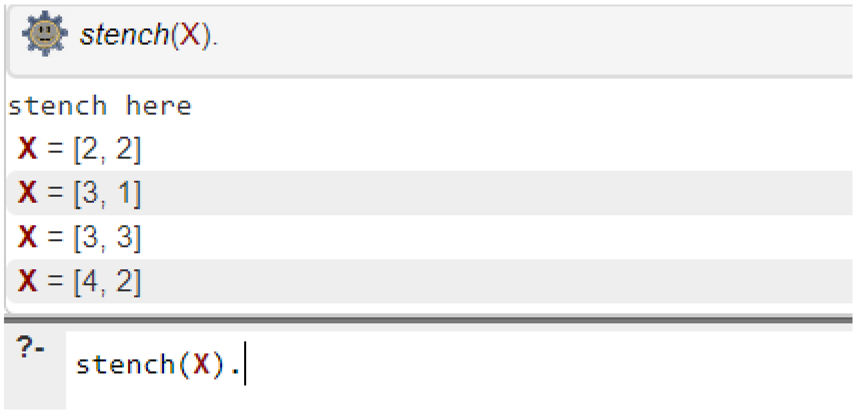
This predicate should return true if and only if the input is (3,2)



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

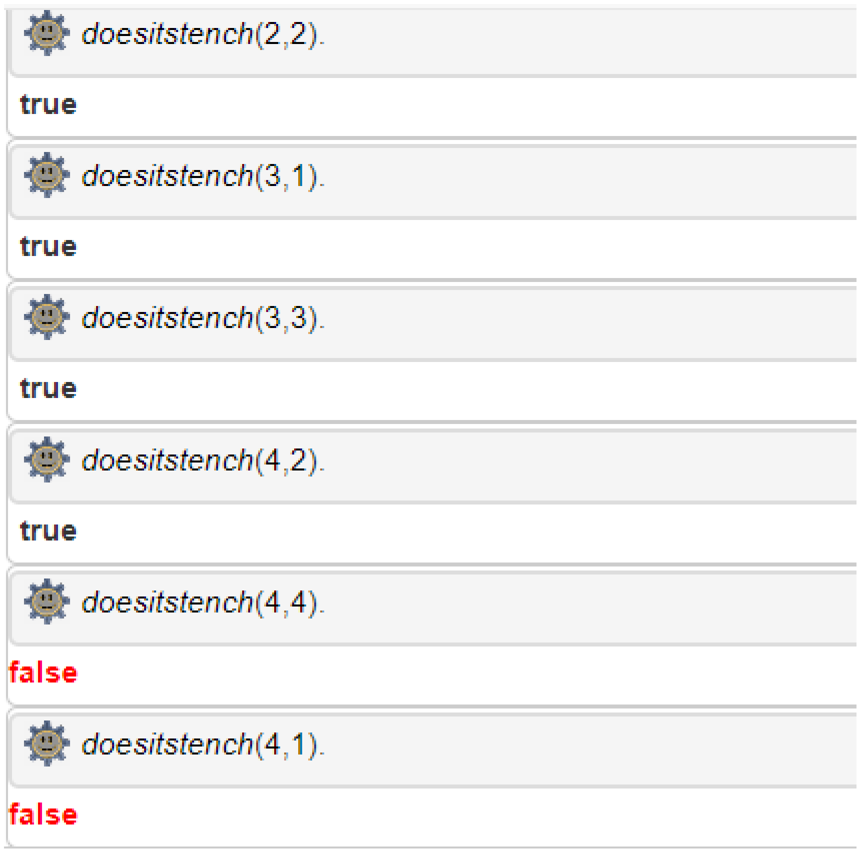
The result is logical

- List of rooms where there is stench:



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

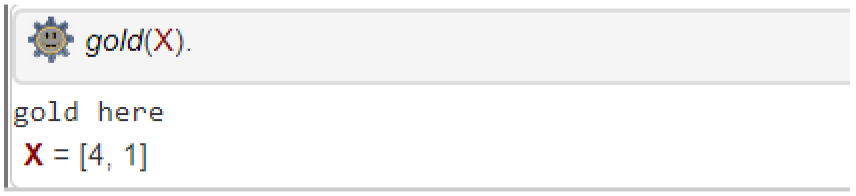
- Then the predicate doesitstench(A,B) checks if in a given room we have stench



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

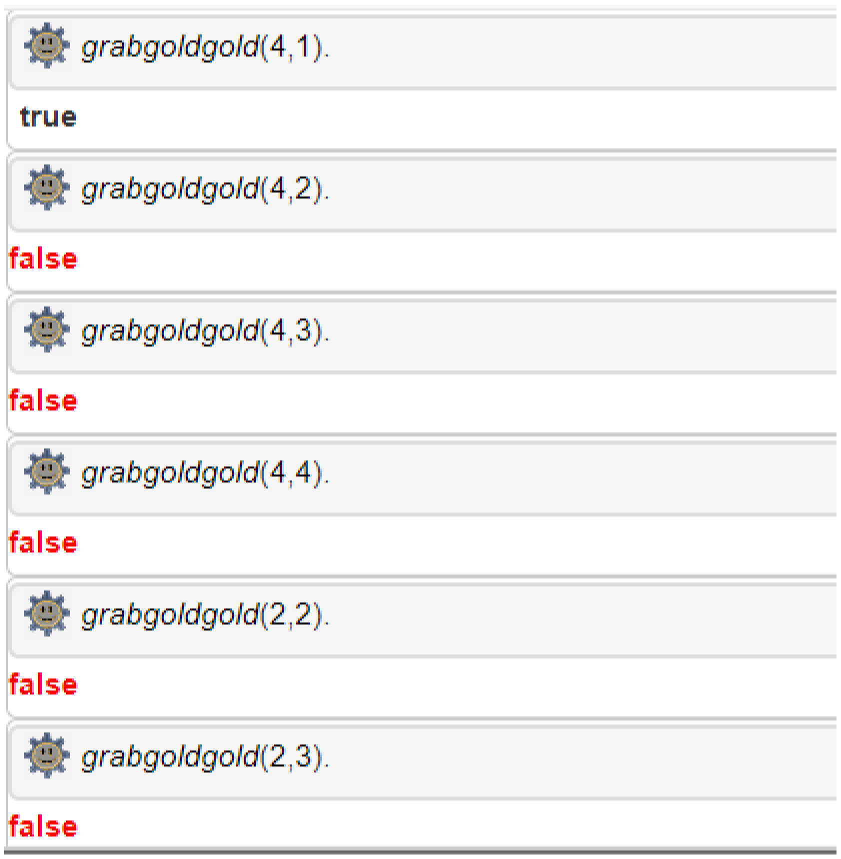
The results concord with the rooms where there is stench

- Gold location:



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

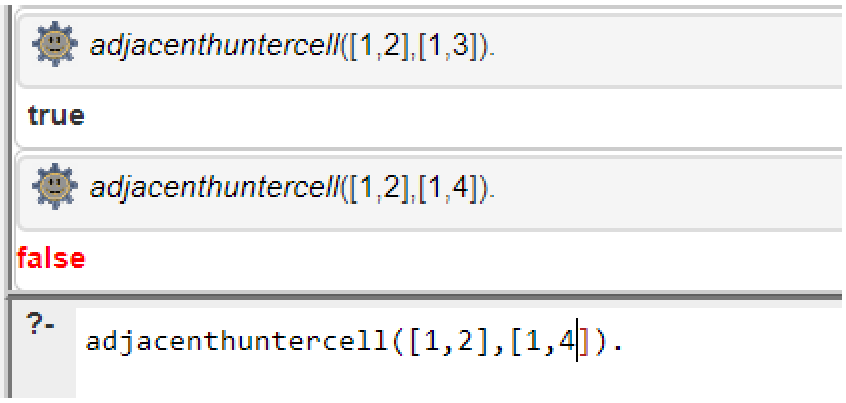
- Then grabgoldgold(A,B) checks if we can grab gold from the input position. It should return true if and only if the entered position is (4,1)



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

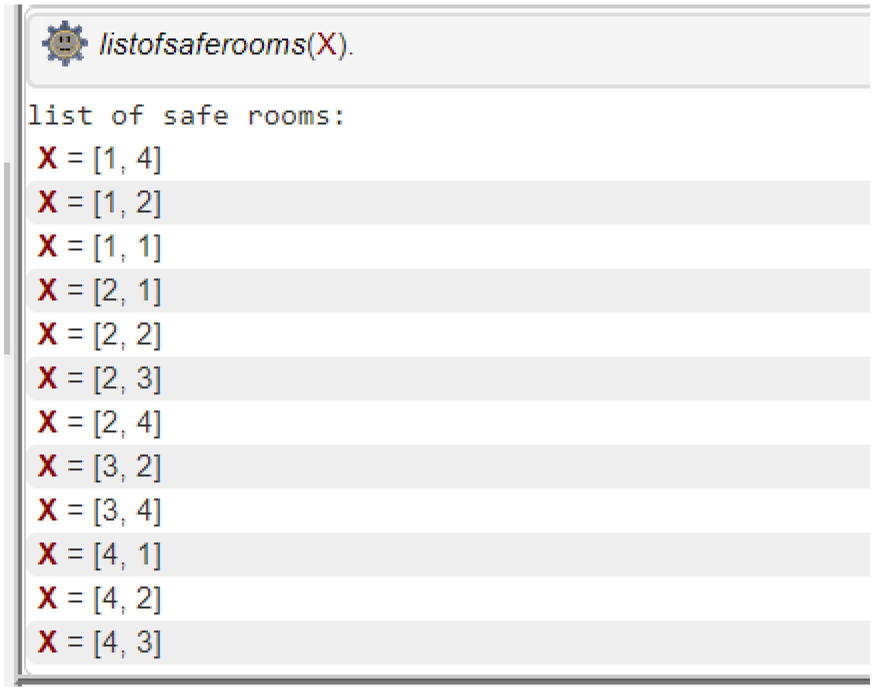
- AdjacentTo (R(X,Y), R(ZT)) room R(T,Z) is adjacent to room R(X,Y)

adjacenthuntercell([X, Y], [A, B]) checks if 2 rooms given as inputs are adjacent



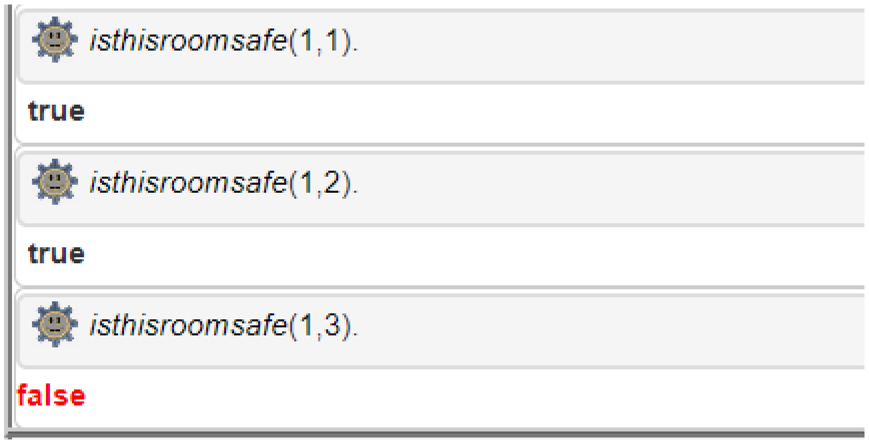
```
adjacenthuntercell([1,2],[1,3]).
true
adjacenthuntercell([1,2],[1,4]).
false
?- adjacenthuntercell([1,2],[1,4]).
```

- List of safe rooms:



```
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]
```

- The predicate isthisroomsafe(X,Y) checks if a room with some inputs is safe or not



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

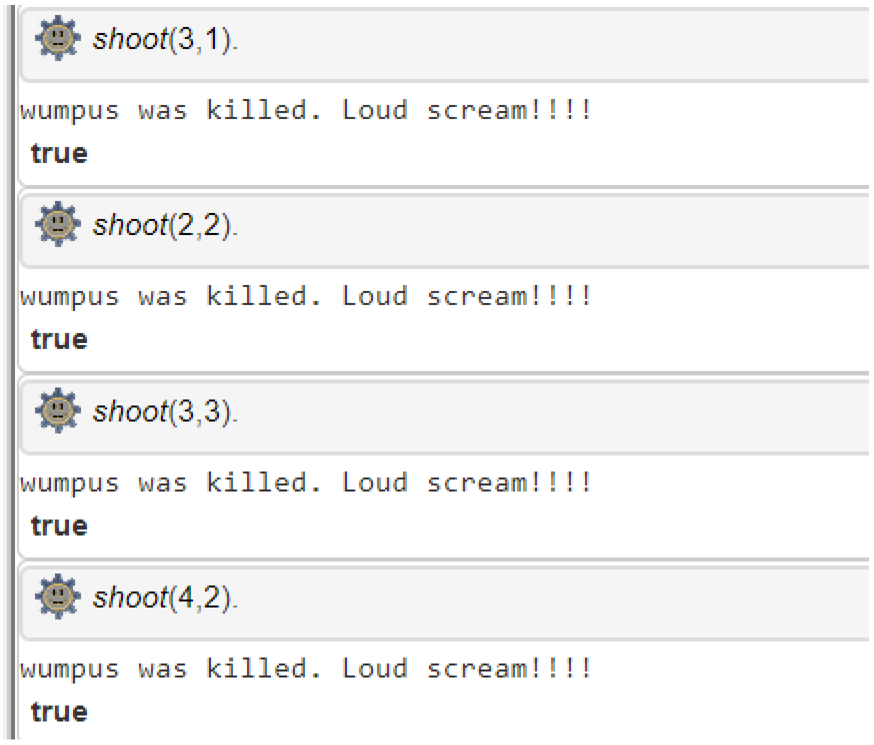
It works well

- the predicate shoot(X,Y) checks if we can shoot and kill the Wumpus from a room with coordinates [X,Y]

if the entered room is adjacent to the room where the wumpus is, we will hear a scream and it will return true. Otherwise, it will return false.

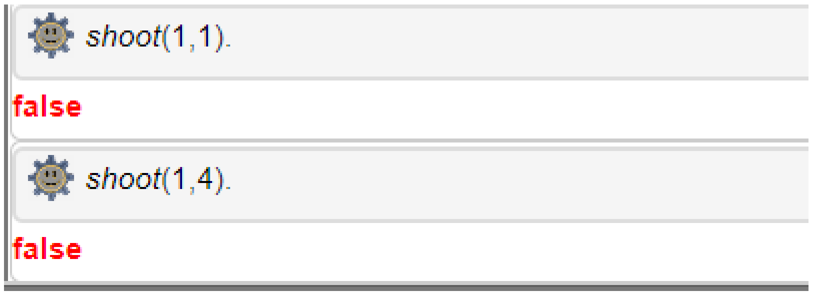
The Wumpus is in room [3,2] so the input should be one of its adjacent rooms from the following:

(3,1) and (2,2) and (4,2) and (3,3)



```
shoot(3,1).
wumpus was killed. Loud scream!!!!
true
shoot(2,2).
wumpus was killed. Loud scream!!!!
true
shoot(3,3).
wumpus was killed. Loud scream!!!!
true
shoot(4,2).
wumpus was killed. Loud scream!!!!
true
```

Now let's try other rooms. We must have false as an answer.



```
shoot(1,1).
false
shoot(1,4).
false
```

The results concord with the chosen configuration.

<u>C.Limitations of our solution:</u>

The system of points in the game is not set. For example, we can launch an arrow more than once to shoot the Wumpus and we do not lose -10 after throwing the arrow. If the agent and the Wumpus are on the same room, we do not gain lose 1000 points because of the Wumpus eating the agent since we did not include points in our code. Also, if the agent fell into a pit, it will not again lose 1000 points. Moreover, the agent does not gain 1000 points if it leaves the game after grabbing the gold. We did not handle those limitations because our program deals with the ability to give answers related to possibilities of actions. In other words, a human asks questions and the human should get answers about eventualities. Because of those limitations, the game does not reach an end automatically. Indeed, the agent will not die even if it loses 1000 points.

<u>Conclusion:</u>

From those 3 configurations, we can conclude that the agent performed well in all the questions that we asked. In other words, the agent was efficient in terms of representing its knowledge base. Also, we feel that we have learnt a lot during this project as it was the first time we implement a knowledge base agent that uses logic to answer questions(from a human) related to the components of the knowledge base(is it possible to kill the Wumpus from a certain position, is this room safe, can we grab gold from here…)