

# Wumpus Game Project Report

Name: Latifa Jbark

Teammate: Oumaima Laghzaoui

## Introduction:

In this project, we aim to develop a logical agent for the Wumpus game in which we will test different queries. Our knowledge base is supposed to decide whether a room is safe or not, decide whether we have a pit or a Wumpus in a specific room...

The Wumpus World is 4 by 4 matrix in which each cell represents a room; in total we have 16 rooms. In our world, the rooms can be safe if they don't have a pit or a Wumpus. They can also contain a pit or a Wumpus. A room can also contain gold.

In this project, we asked many queries concerning the safety of rooms, the existence of a Wumpus or a pit in a specific room, and the existence of a stench or a breeze in a room.

## Explanation of predicates:

- **AdjacentTo(room(X, Y), room(A, B))** will provide us with the adjacent rooms (A,B) to the room (X,Y). The adjacent rooms occur in the cells that are in the left, right, up, and down the room(X,Y).
- **Pit(room(X,Y))**: there is a pit in a room if all of its adjacent rooms contain a breeze.
- **breeze(room(X,Y))**: there is a breeze in a room if one of its adjacent rooms contain a pit.
- **wumpus(room(X,Y))**: there is a Wumpus in a room if all of its adjacent rooms contain a stench.
- **stench(room(X,Y))**: there is a stench in a room if one of its adjacent rooms contain a wumpus
- **safe(room(W,Z))**: this is used to check if a room is safe or not. If it is safe, it will return true; otherwise, it will return false.
- **safe(room(X,Y), room(W,Z))**: this is used to get all the safest rooms (W,Z) that are adjacent to a given room (X,Y).
- **shootWumpus(room(X,Y))**: we can shoot a Wumpus from the room(X,Y) if one of its adjacent rooms contain a Wumpus.

- **gold(room(X,Y))**: This will return true if there is gold in position (X,Y). Otherwise, it will return false.
- **grabGold(room(X,Y))**: it will return true if the agent is in position(X,Y) and also there is gold in position(X,Y).
- The variables entered as a parameter of each function correspond to the room in cell [X , Y]

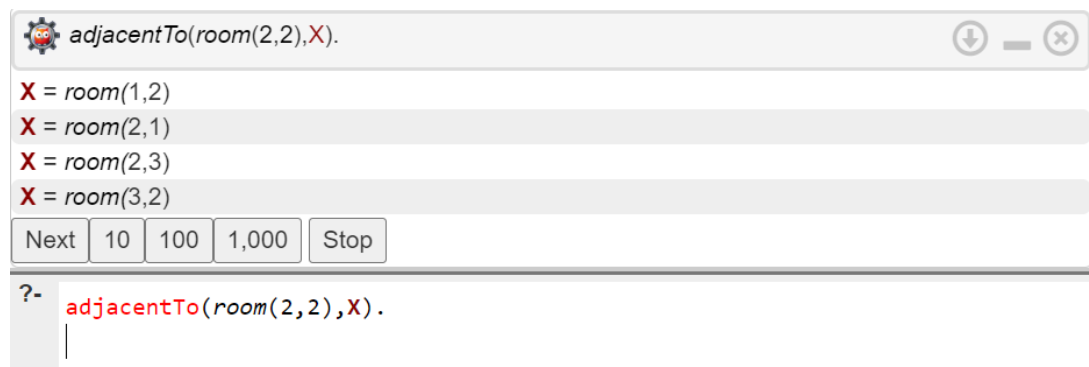
### Snapshots of various Experiments:

#### - **The First configuration:**

The first configuration we tested is the following:

```
%-----initial configurations-----
wumpus_position(room(1,3)).
pit_position(room(3,3)).
pit_position(room(1,2)).
pit_position(room(4,4)).
gold_position(room(2,3)).
agent_position(room(1,1)).
```

Let's check the adjacent function:



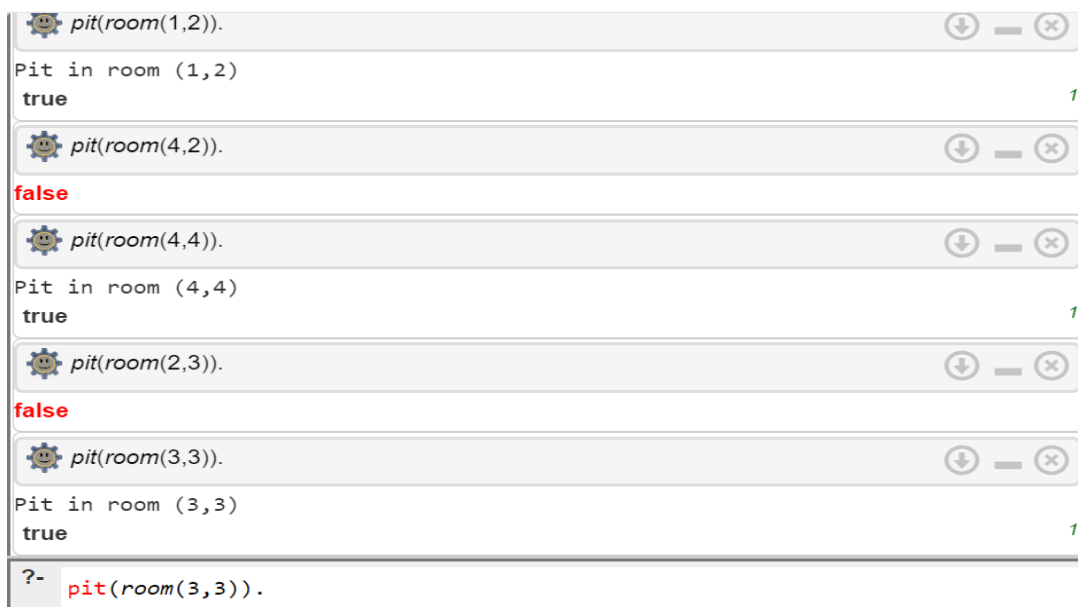
```
adjacentTo(room(2,2),X).
```

X = room(1,2)  
X = room(2,1)  
X = room(2,3)  
X = room(3,2)

Next 10 100 1,000 Stop

?- adjacentTo(room(2,2),X).

Now, we will check the pit positions:



```
pit(room(1,2)).
```

Pit in room (1,2)  
true 1

```
pit(room(4,2)).
```

false

```
pit(room(4,4)).
```

Pit in room (4,4)  
true 1

```
pit(room(2,3)).
```

false

```
pit(room(3,3)).
```

Pit in room (3,3)  
true 1

?- pit(room(3,3)).

We will check the breeze function:

The screenshot shows a Prolog interpreter window with the following content:






















- Query: `breeze(room(1,4)).` Result: `false`
- Query: `breeze(room(3,2)).` Result: `true` (1)
- Query: `breeze(room(3,4)).` Result: `true` (1)
- Query: `breeze(room(4,4)).` Result: `false`
- Query: `breeze(room(4,2)).` Result: `false`
- Query: `breeze(room(2,3)).` Result: `true` (1)
- Query: `?- breeze(room(2,3)).`

The stench function works as follow:























The screenshot shows a Prolog interpreter window with the following content:

- Query: `stench(room(1,4)).` Result: `true` (1)
- Query: `stench(room(4,4)).` Result: `false`
- Query: `stench(room(3,4)).` Result: `false`
- Query: `stench(room(3,3)).` Result: `false`
- Query: `stench(room(2,3)).` Result: `true` (1)
- Query: `stench(room(1,2)).` Result: `true` (1)
- Query: `?- stench(room(1,2)).`

















To check whether we have a Wumpus in a certain position, we use:

 <code>wumpus(room(1,2)).</code>	  
<b>false</b>	
 <code>wumpus(room(1,1)).</code>	  
<b>false</b>	
 <code>wumpus(room(4,2)).</code>	  
<b>false</b>	
 <code>wumpus(room(1,3)).</code>	  
Wumpus in room (1,3) <b>true</b>	 1
 <code>wumpus(room(3,3)).</code>	  
<b>false</b>	
<b>?- wumpus(room(3,3)).</b>	

Now, we will check whether a specific position is safe or not:

 <code>safe(room(1,3)).</code>	  
<b>false</b>	
 <code>safe(room(2,3)).</code>	  
Room (2,3) is safe <b>true</b>	 1
 <code>safe(room(1,3)).</code>	  
<b>false</b>	
 <code>safe(room(3,3)).</code>	  
<b>false</b>	
 <code>safe(room(3,4)).</code>	  
Room (3,4) is safe <b>true</b>	 1
<b>?- safe(room(3,4)).</b>	

Let's check all the safest rooms that are adjacent to a specific room

 <code>safe(room(3,4),X).</code>	  
<b>X = room(2,4)</b> <b>false</b>	
 <code>safe(room(1,1),X).</code>	  
<b>X = room(2,1)</b> <b>false</b>	
 <code>safe(room(3,1),X).</code>	  
<b>X = room(2,1)</b> <b>X = room(3,2)</b> <b>X = room(4,1)</b> <b>false</b>	
 <code>safe(room(4,4),X).</code>	  
<b>X = room(3,4)</b> <b>X = room(4,3)</b>	
<b>?- safe(room(4,4),X).</b>	

## - The Second configuration:

Now, we will change the starting configurations and test the remaining functions:

```
%-----initial configurations-----  
wumpus_position(room(3,1)).  
pit_position(room(1,2)).  
pit_position(room(3,1)).  
pit_position(room(3,4)).  
pit_position(room(3,3)).  
gold_position(room(3,1)).  
agent_position(room(1,1)).
```

We will check whether we can shoot the Wumpus from a certain room

The screenshot shows a Prolog environment with the following content:

- Query: `shootWumpus(room(3,2)).` Result: `true` (green), `false` (red). A green '1' is on the right.
- Query: `shootWumpus(room(1,2)).` Result: `false` (red).
- Query: `shootWumpus(room(2,2)).` Result: `false` (red).
- Query: `shootWumpus(room(4,1)).` Result: `true` (green), `false` (red). A green '1' is on the right.

At the bottom, there is a control bar with buttons: Next, 10, 100, 1,000, Stop. Below that, a prompt `?- shootWumpus(room(4,1)).` is shown.

Now, let's check the gold function

The screenshot shows a Prolog environment with the following content:

- Query: `gold(X).` Result: `There is gold in room (3,1)`, `Grab it to gain more scores...!`, `X = room(3,1)`.
- Query: `gold(room(1,1)).` Result: `false` (red).
- Query: `gold(room(3,1)).` Result: `There is gold in room (3,1)`, `Grab it to gain more scores...!`, `true` (green). A green '1' is on the right.
- Query: `gold(room(2,2)).` Result: `false` (red).

At the bottom, a prompt `?- gold(room(2,2)).` is shown.

Here, we changed the position of the agent so that we can see if the agent can grab the gold or they will have any problems with that

```
⚙️ grabGold(room(2,2)).
false

⚙️ grabGold(room(1,2)).
false

⚙️ grabGold(room(1,3)).
false

⚙️ grabGold(room(3,1)).
You have grabed the gold in room (3,1)
Points have been added to your score...!
true
1

?- grabGold(room(3,1)).
```

### - The Third configuration:

Now, we change the configuration to test again our functions:

```
%-----initial configurations-----
wumpus_position(room(1,2)).
pit_position(room(1,2)).
pit_position(room(2,2)).
pit_position(room(3,4)).
gold_position(room(3,3)).
agent_position(room(1,1)).
```

```
⚙️ breeze(room(1,1)).
true
1

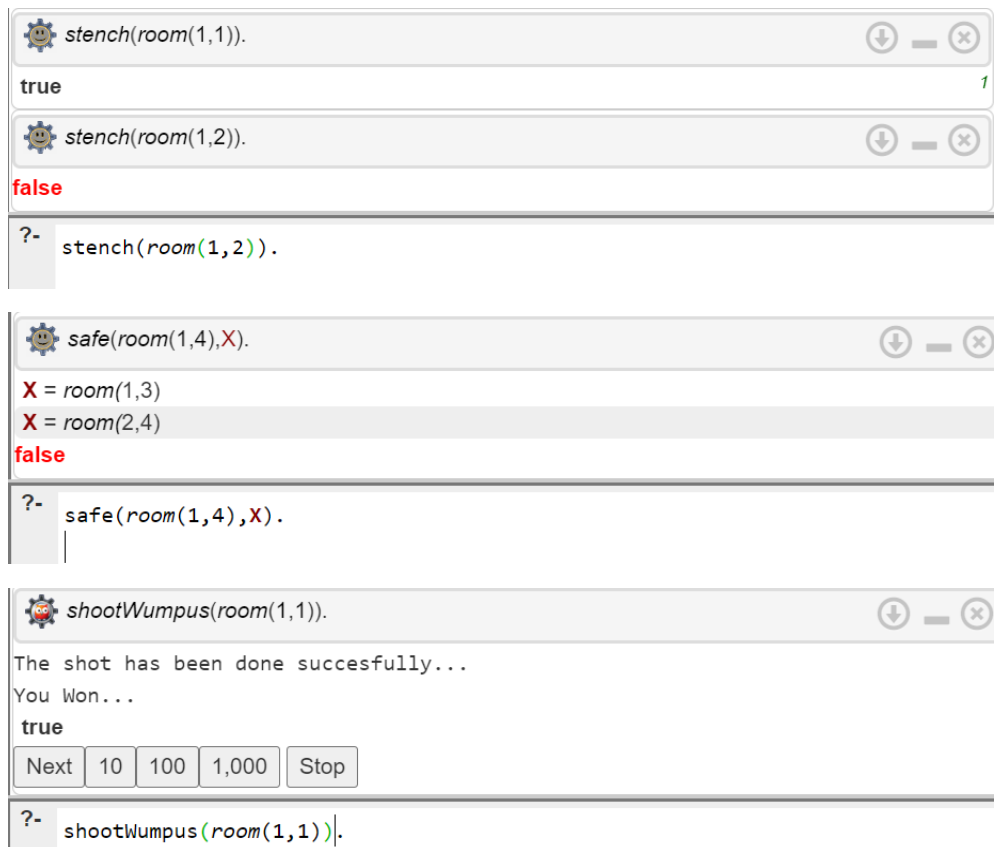
⚙️ breeze(room(4,4)).
true
1

⚙️ breeze(room(3,4)).
false

⚙️ breeze(room(1,2)).
true
1

⚙️ breeze(room(1,4)).
false

?- breeze(room(1,4)).
```



### *The performance rate:*

Our program works perfectly for the configurations given to it. However, there are some situations that influence the performance of the program. For example, the situation where we check if there is a breeze in the pit position and it returns false so the user might infer that this place could be safe for them. Therefore, it should output that there no breeze but there is a pit at that cell. Also, in the situation where we had gold and pit in the same room. When we run the function grab gold it gave us true meaning that the gold has been grabbed successfully. This normally should be false since we have a pit at that position.

We have done 54 queries; 2 of them gave us a wrong answer meaning that 52 are the correct ones.

Therefore, the performance rate is:

$$2/54 = 0.037$$

### *Limitations of our solution and future remedies:*

The limitation of our solution lies in the fact that our program does not handle the case where there is gold in the same cell as pit or Wumpus. For example, in the second configuration, when we query to our program to take a gold in the position when there is a pit and the Wumpus it said that the gold has been collected successfully. However, when the agent enters this cell, they will fall in the pit, or they will be killed by the Wumpus. Therefore, we should get as an answer, when asking about gold, that even though there is gold in that position the agent should not enter to it. To solve this issue, if we are in one of the adjacent rooms to the gold, we need to also check whether this position contains a pit or a Wumpus, so that we don't lose the game while trying to get the gold in this room.