Wumpus World – Project Report

Report writer: Mohamed Moumou

Teammate: Ouwais Zlaïgi

Project #2: The Wumpus World

I. Introduction:

This report will describe and explain the code my teammate and I wrote using SWI Prolog to simulate configuration of the Wumpus world game. This report will mainly describe the main predicates and variables used in the code, show snapshots of the game that simulate different configurations, calculate the performance rate. In the end of the report, I will also mention limitations of the code and possible solutions to overcome these limitations.

II. List of Predicates and Functions and their Explanation:

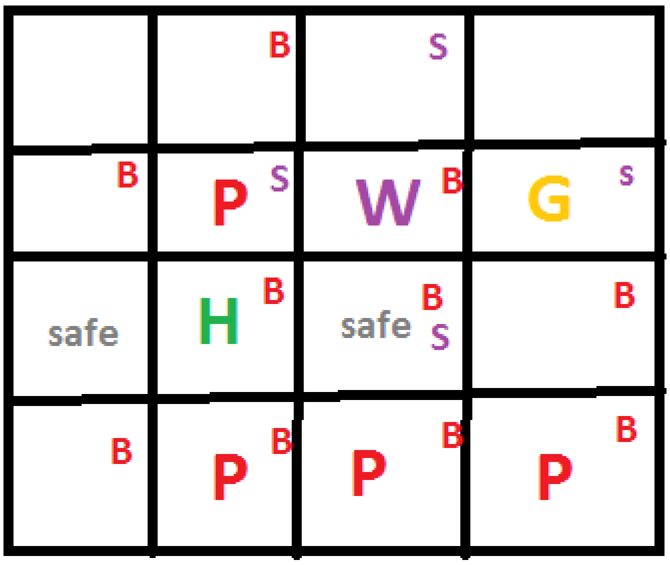
- start: initializes a new configuration of the game
- setHunterAt (): changes the position and the direction of the hunter based on the information that the user enters.
- setPitsPos (): retracts all the existing pits and asserts new pits into the knowledge base.
- pitsProb (X, Y): adds a pit to the position (X, Y) with a probability of 0.2. This predicate is also used in setPitsPos () to assign pits randomly based on the 0.2 probability.
- setWumpusPos (): retracts the previous position of the Wumpus and assigns a new position to it.
- setGoldPos(): retracts the previous position of the gold and assigns a new position to it.

- room (X, Y): returns all the rooms of the configuration, or if specified, returns if a Boolean value to confirm whether there is a room at the position selected.
- getAdjacentRooms (r (X, Y), L): returns a list of all adjacent rooms to the room at position (X, Y).
- adjacentTo (r (X, Y), r (Z, A)): checks whether the room at position (X, Y), and (Z, A) are adjacent or not.
- breeze (r (X, Y)): returns all the positions where a breeze exists in the configuration. If X and Y are specified a Boolean value is returned to confirm whether there is a breeze at the position selected.
- pit (r (X, Y)): returns all the positions where a pit exists in the configuration. If X and Y are specified a Boolean value is returned to confirm whether there is a pit at the position selected.
- gold (r (X, Y)): returns the position where the gold exists in the configuration. If X and Y are specified a Boolean value is returned to confirm whether there is gold at the position selected.
- wumpus (r (X, Y)): returns the position where the wumpus exists in the configuration. If X and Y are specified a Boolean value is returned to confirm whether there is a wumpus at the position selected.
- stench (r (X, Y)): returns all the positions where a stench exists in the configuration. If X and Y are specified a Boolean value is returned to confirm whether there is a stench at the position selected.
- Safe (): returns all the rooms that are both adjacent to the hunter's current room and safe.
- safeWumpus (): checks whether the Wumpus is safe (the hunter is not in an adjacent room).

- wumpusAlive (): checks whether the Wumpus is still alive.
- hasArrow (): checks whether the hunter still has an arrow.
- wallCheck (r (X, Y)): checks whether the position entered is not a valid room.
- shootWumpus (): shoots the Wumpus if the Wumpus is in an adjacent position.
- shootWumpusWithDirection (): shoots the Wumpus if and only if the hunter is adjacent to the Wumpus and it is facing it.
- grabGold (): grabs the gold if and only if the hunter is at the same position as the gold.
- turnLeft (): changes the direction of the hunter 90 degrees to the left.
- TurnRight (): changes the direction of the hunter 90 degrees to the right.

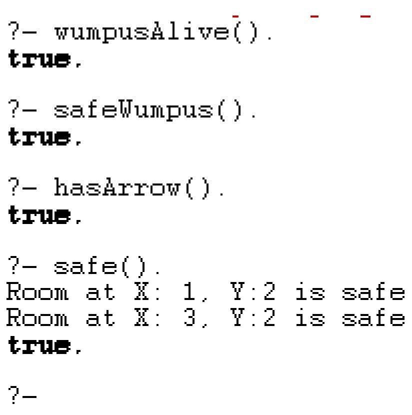
III. Experiments with Different Configurations:

1. Configuration 1:



Screenshots of different functions being executed in the SWI-Prolog CLI:

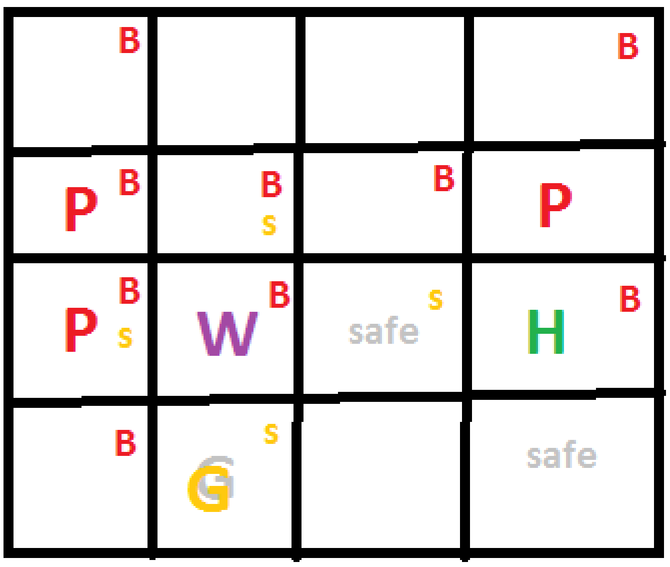
```
?- start.
Please enter the X coordinate of the hunter:
|: 2.
Please enter the Y coordinate of the hunter:
|: 2.
Please enter the initial direction of the wumpus
|: e.

true.

?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = e.

?- wampus(r(X,Y)).
Correct to: "wumpus(r(X,Y))"?
Please answer 'y' or 'n'? yes
X = Y, Y = 3.

?-
|    pit(r(X,Y)).
X = 2,
Y = 1 ;
X = 2,
Y = 3 ;
X = 3,
Y = 1 ;
X = 4,
Y = 1.

?- gold(r(X,Y)).
X = 4,
Y = 3.

?- breeze(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 3 ;
X = 2,
Y = 1 ;
X = Y, Y = 2 ;
X = Y, Y = 2 ;
X = 2,
Y = 4 ;
X = 3,
Y = 1 ;
X = 3,
Y = 1 ;
X = 3,
Y = 2 ;
X = Y, Y = 3 ;
X = 4,
Y = 1 ;
X = 4,
Y = 2 ;
false.

?- stench(r(X,Y)).
X = 2,
Y = 3 ;
X = 3,
Y = 2 ;
X = 3,
Y = 4 ;
X = 4,
Y = 3 ;
false.
```



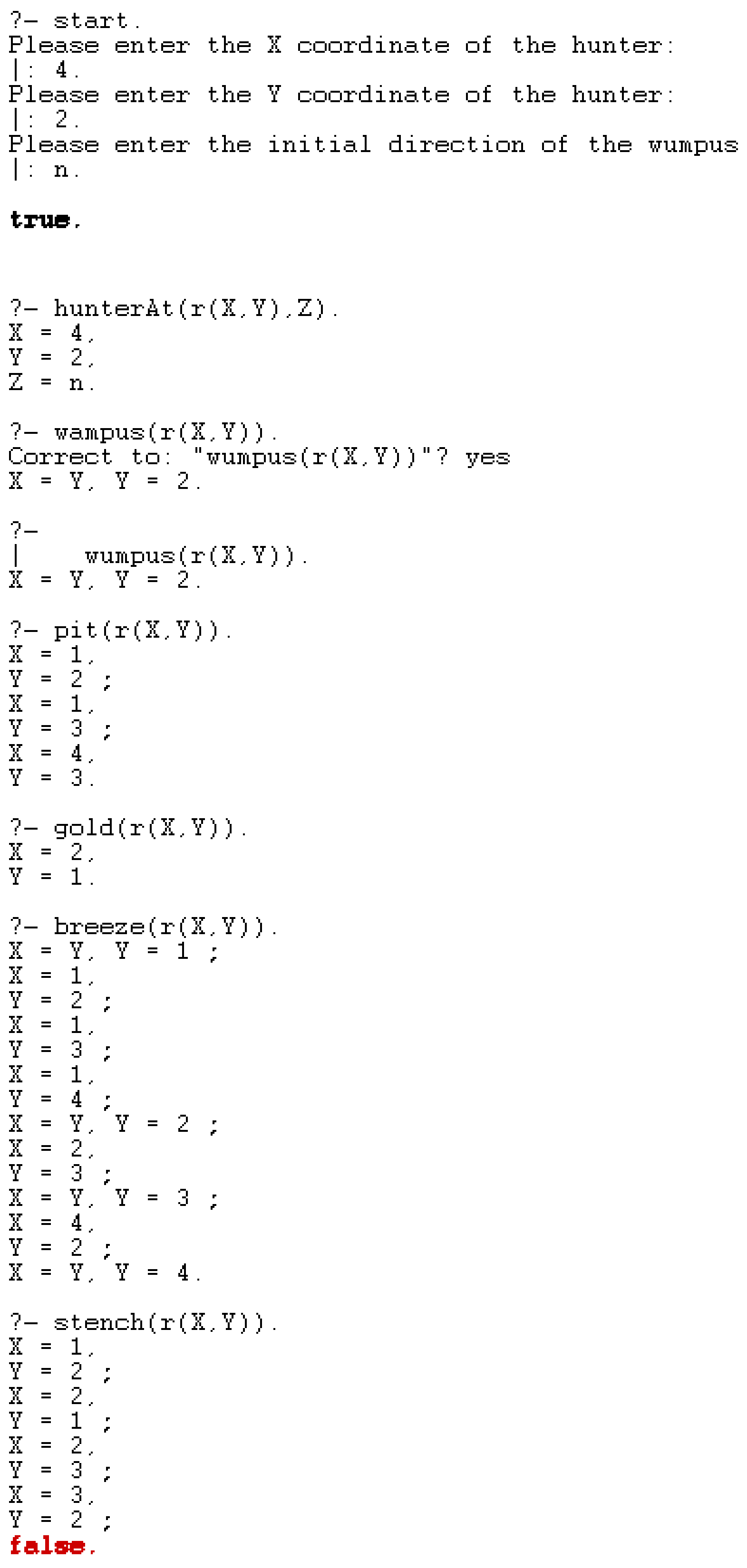
```
?- wumpusAlive().
true.

?- safeWumpus().
true.

?- hasArrow().
true.

?- safe().
Room at X: 1, Y:2 is safe
Room at X: 3, Y:2 is safe
true.

?-
```

We initiate a configuration by calling start, and then we set the position of the hunter to (2,2) looking at the east. We then check the position of the wumpus, pits, gold, and breezes. We can see that the positions of the breezes are accurate (as represented in the configuration above). However, we can see that there are breezes in positions where there are pits, which is unnecessary. We, then, check whether the Wumpus is alive, for which the answer is true. Also, the wumpus is safe because the hunter is not adjacent to it, and since the wumpus is still alive, the hunter still has their arrow. The function safe printed all the safe rooms (1, 2) and (3, 2), rooms that have no pit or wumpus.

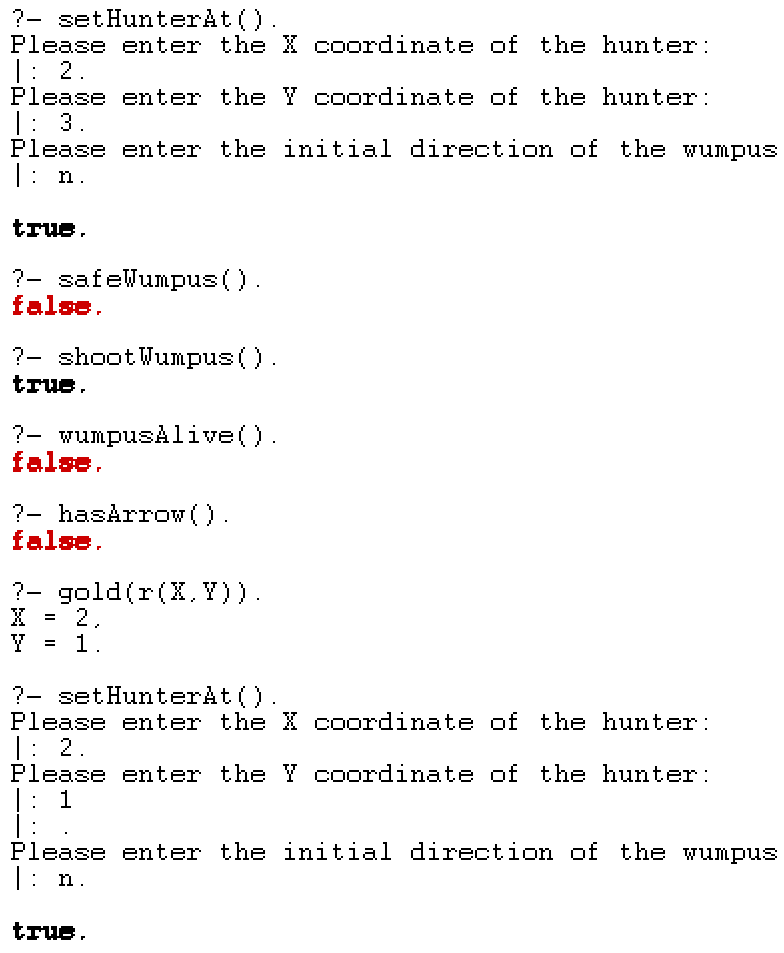
2. Configuration 2:



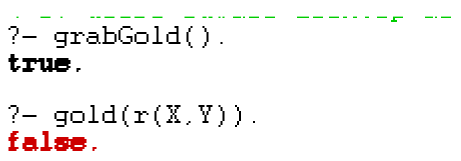
Screenshots of different functions being executed in the SWI-Prolog CLI for the configuration above:



```
?- start.
Please enter the X coordinate of the hunter:
|: 4.
Please enter the Y coordinate of the hunter:
|: 2.
Please enter the initial direction of the wumpus
|: n.

true.


?- hunterAt(r(X,Y),Z).
X = 4,
Y = 2,
Z = n.

?- wampus(r(X,Y)).
Correct to: "wumpus(r(X,Y))"? yes
X = Y, Y = 2.

?-
|    wumpus(r(X,Y)).
X = Y, Y = 2.

?- pit(r(X,Y)).
X = 1,
Y = 2 ;
X = 1,
Y = 3 ;
X = 4,
Y = 3.

?- gold(r(X,Y)).
X = 2,
Y = 1.

?- breeze(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 2 ;
X = 1,
Y = 3 ;
X = 1,
Y = 4 ;
X = Y, Y = 2 ;
X = 2,
Y = 3 ;
X = Y, Y = 3 ;
X = 4,
Y = 2 ;
X = Y, Y = 4.

?- stench(r(X,Y)).
X = 1,
Y = 2 ;
X = 2,
Y = 1 ;
X = 2,
Y = 3 ;
X = 3,
Y = 2 ;
false.
```



```
?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 2.
Please enter the Y coordinate of the hunter:
|: 3.
Please enter the initial direction of the wumpus
|: n.

true.

?- safeWumpus().
false.

?- shootWumpus().
true.

?- wumpusAlive().
false.

?- hasArrow().
false.

?- gold(r(X,Y)).
X = 2,
Y = 1.

?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 2.
Please enter the Y coordinate of the hunter:
|: 1
|: .
Please enter the initial direction of the wumpus
|: n.

true.
```

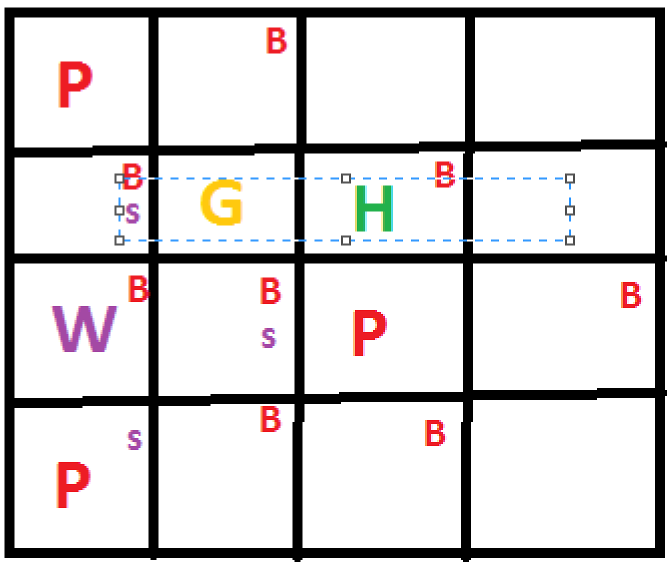


```
?- grabGold().
true.

?- gold(r(X,Y)).
false.
```

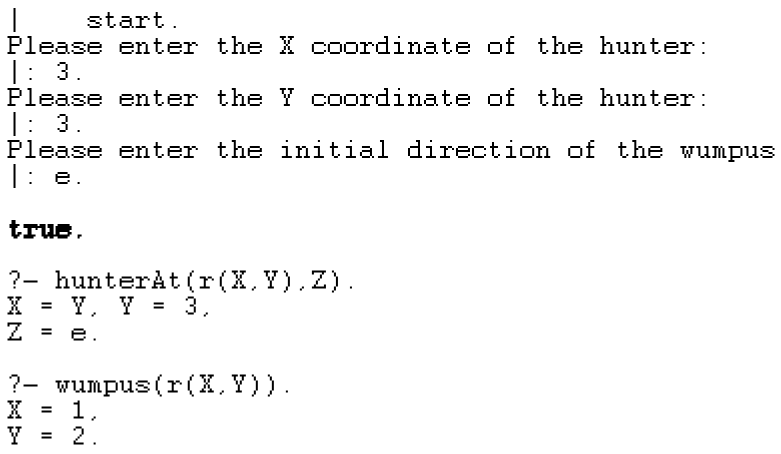
The new functions used in testing this configuration are shootWumpus (), and grabGold ().

To shoot the wumpus, we had to set the position of the hunter to a position adjacent to the wumpus. As such, we called the function safeWumpus and it returned false. We, then, shoot the wumpus and check whether it is alive or not, and whether the hunter still has their arrow. We, then, changed the position of the hunter again to the position of the gold to grab it.

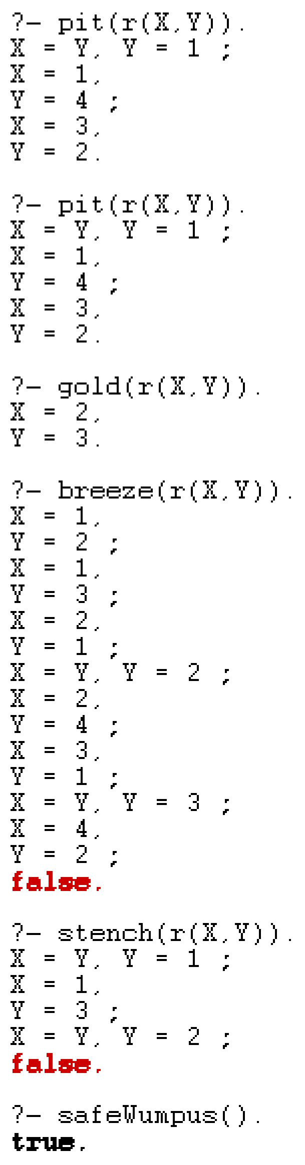
3. Configuration 3:



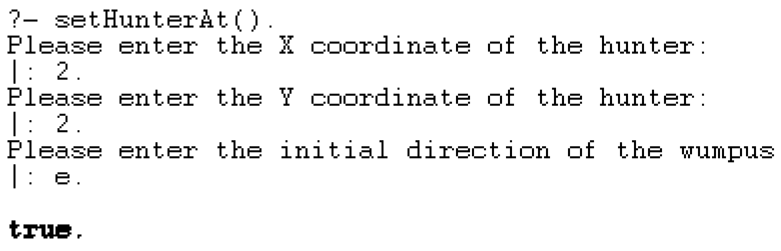
Screenshots of different functions being executed in the SWI-Prolog CLI for the configuration above:



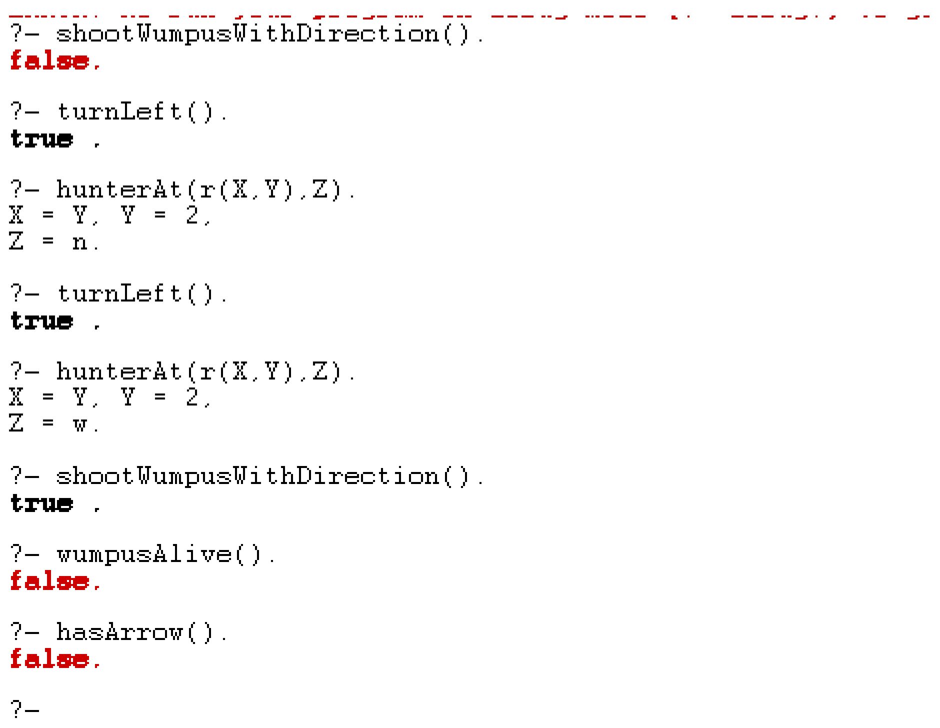
```
|    start.
Please enter the X coordinate of the hunter:
|: 3.
Please enter the Y coordinate of the hunter:
|: 3.
Please enter the initial direction of the wumpus
|: e.

true.

?- hunterAt(r(X,Y),Z).
X = Y, Y = 3,
Z = e.

?- wumpus(r(X,Y)).
X = 1,
Y = 2.
```



```
?- pit(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 4 ;
X = 3,
Y = 2.

?- pit(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 4 ;
X = 3,
Y = 2.

?- gold(r(X,Y)).
X = 2,
Y = 3.

?- breeze(r(X,Y)).
X = 1,
Y = 2 ;
X = 1,
Y = 3 ;
X = 2,
Y = 1 ;
X = Y, Y = 2 ;
X = 2,
Y = 4 ;
X = 3,
Y = 1 ;
X = Y, Y = 3 ;
X = 4,
Y = 2 ;
false.

?- stench(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 3 ;
X = Y, Y = 2 ;
false.

?- safeWumpus().
true.
```



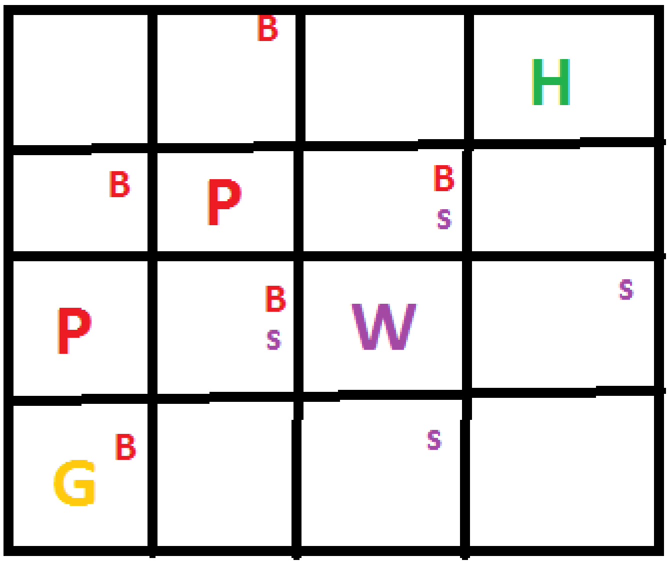
```
?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 2.
Please enter the Y coordinate of the hunter:
|: 2.
Please enter the initial direction of the wumpus
|: e.

true.
```



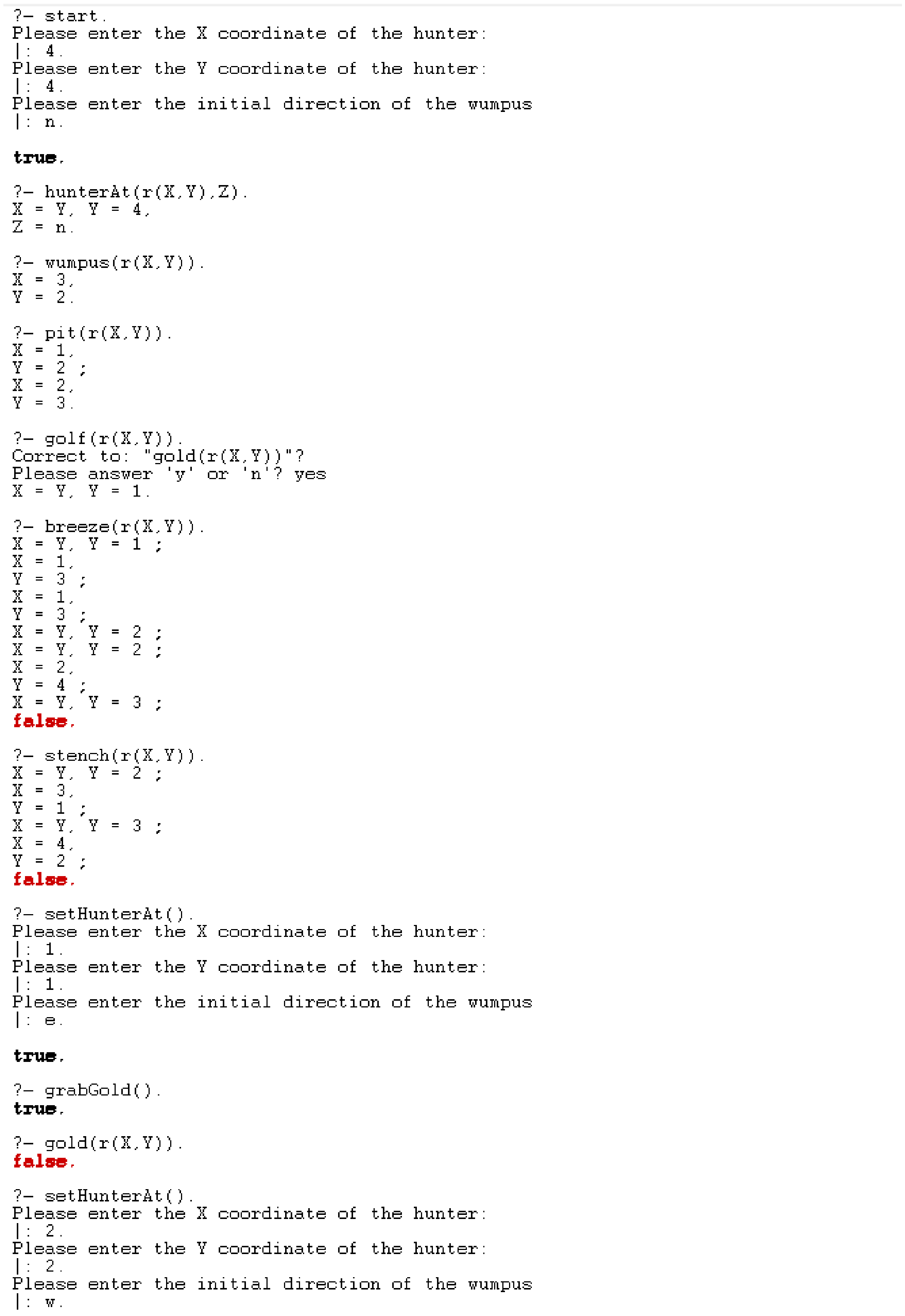
```
?- shootWumpusWithDirection().
false.

?- turnLeft().
true .

?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = n.

?- turnLeft().
true .

?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = w.

?- shootWumpusWithDirection().
true .

?- wumpusAlive().
false.

?- hasArrow().
false.

?-
```

For this configuration, we tried the function shootWumpusWithDirection (). This function does not only require the hunter to be adjacent to the wumpus, it also requires that the hunter faces the latter. The hunter was at position (2, 2) and could not kill the Wumpus at position (1,2) because the hunter was facing the east, and it needs to face the west to kill the Wumpus. Therefore, we called turnLeft() two time to direct the hunter toward the west. Indeed, when we tried killing the Wumpus using shootWumpusWithDirection (), we were able to kill the Wumpus. However, we can see here that our code does not perform well because, we already shot an arrow, i.e., we do not have an arrow left to shoot the Wumpus.

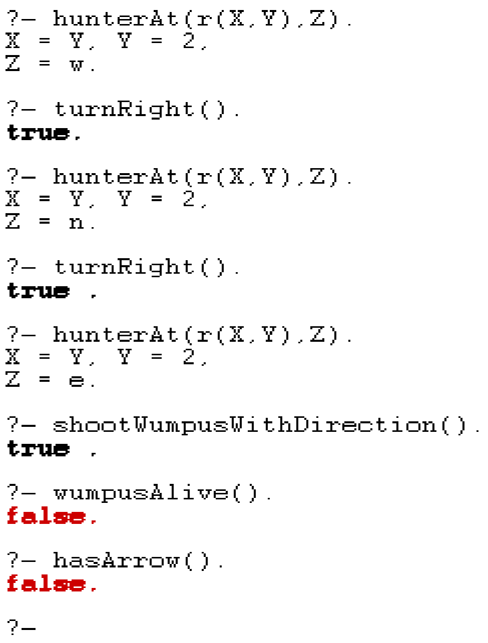
4. Configuration 4:



Screenshots of different functions being executed in the SWI-Prolog CLI for the configuration above:



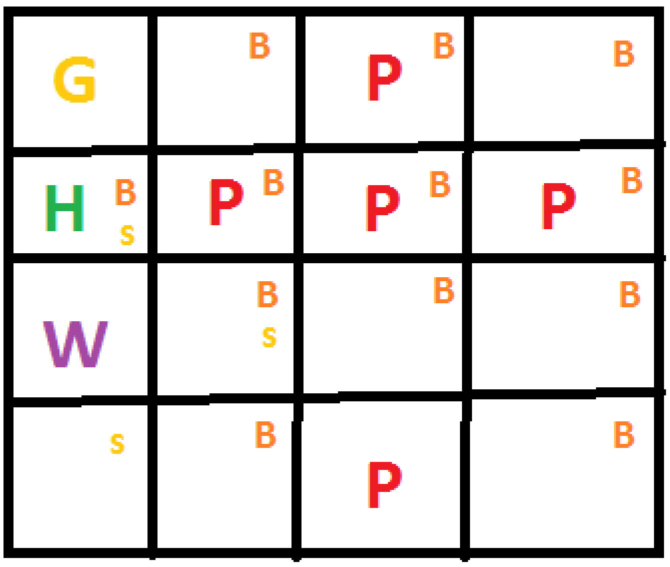
```
?- start.
Please enter the X coordinate of the hunter:
|: 4.
Please enter the Y coordinate of the hunter:
|: 4.
Please enter the initial direction of the wumpus
|: n.

true.

?- hunterAt(r(X,Y),Z).
X = Y, Y = 4,
Z = n.

?- wumpus(r(X,Y)).
X = 3,
Y = 2.

?- pit(r(X,Y)).
X = 1,
Y = 2 ;
X = 2,
Y = 3.

?- golf(r(X,Y)).
Correct to: "gold(r(X,Y))"?
Please answer 'y' or 'n'? yes
X = Y, Y = 1.

?- breeze(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 3 ;
X = 1,
Y = 3 ;
X = Y, Y = 2 ;
X = Y, Y = 2 ;
X = 2,
Y = 4 ;
X = Y, Y = 3 ;
false.

?- stench(r(X,Y)).
X = Y, Y = 2 ;
X = 3,
Y = 1 ;
X = Y, Y = 3 ;
X = 4,
Y = 2 ;
false.

?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 1.
Please enter the Y coordinate of the hunter:
|: 1.
Please enter the initial direction of the wumpus
|: e.

true.

?- grabGold().
true.

?- gold(r(X,Y)).
false.

?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 2.
Please enter the Y coordinate of the hunter:
|: 2.
Please enter the initial direction of the wumpus
|: w.
```



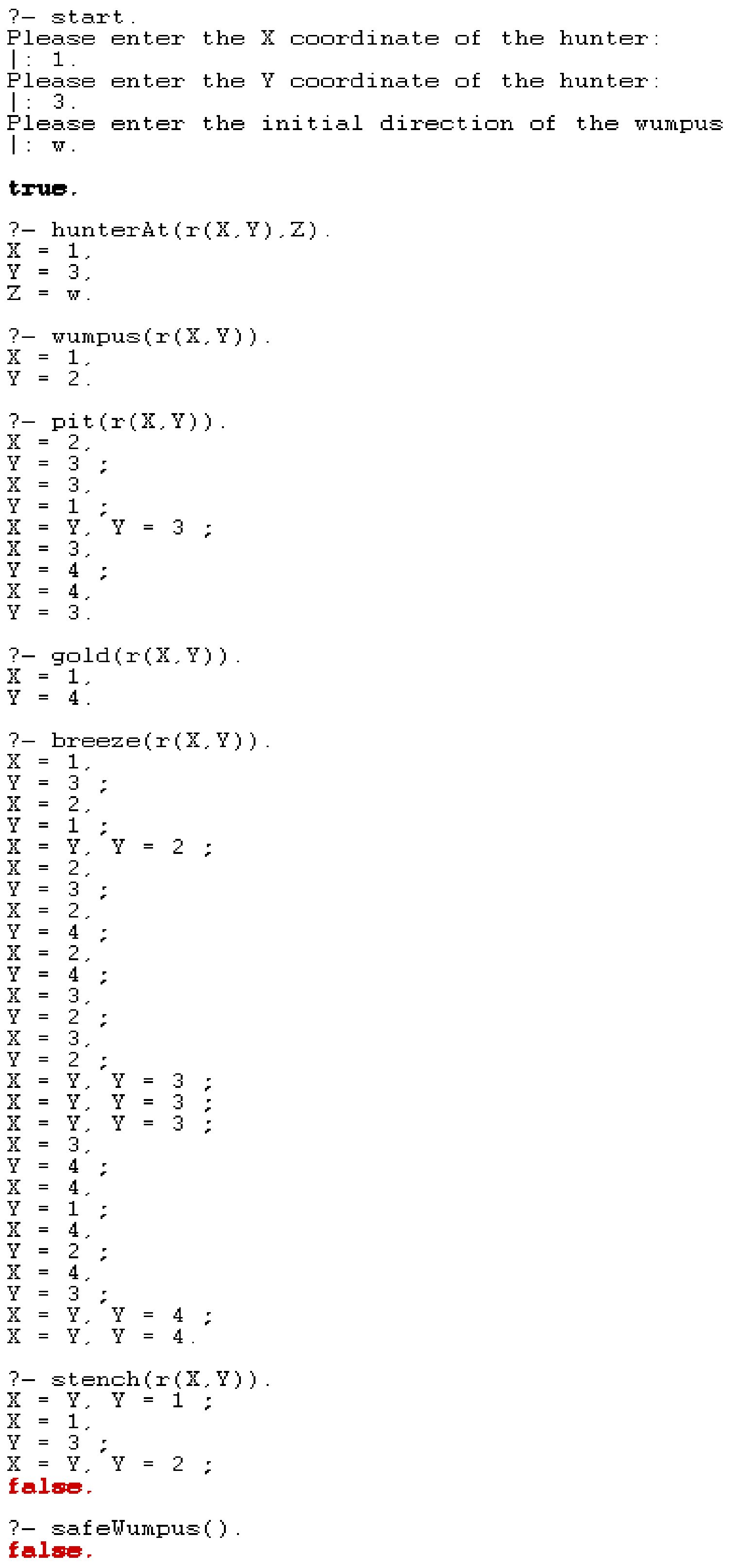
```
?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = w.

?- turnRight().
true.

?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = n.

?- turnRight().
true .

?- hunterAt(r(X,Y),Z).
X = Y, Y = 2,
Z = e.

?- shootWumpusWithDirection().
true .

?- wumpusAlive().
false.

?- hasArrow().
false.

?-
```

In this configuration, I tried to try almost all functions. The new function here was turnRight (). This time the wumpus was facing the west, so we turned them right two times to face the east (the Wumpus).

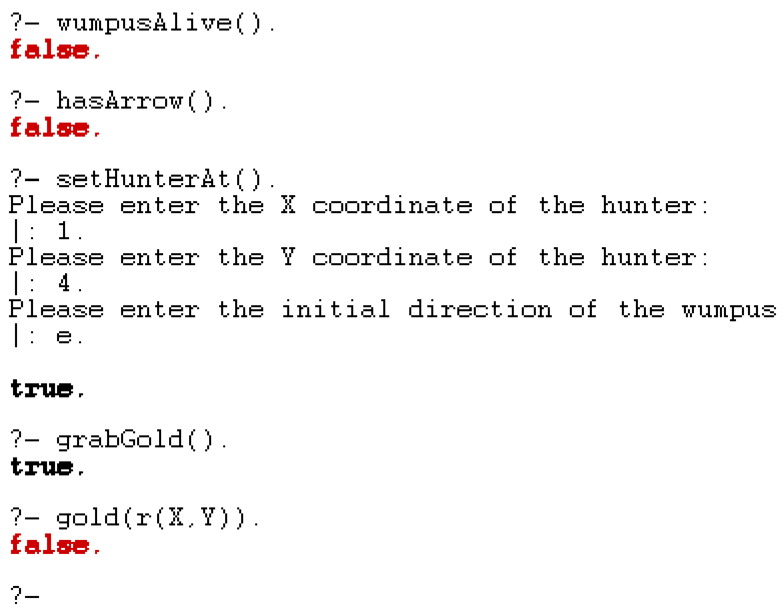
5. Configuration 5:



Screenshots of different functions being executed in the SWI-Prolog CLI for the configuration above:



```
?- start.
Please enter the X coordinate of the hunter:
|: 1.
Please enter the Y coordinate of the hunter:
|: 3.
Please enter the initial direction of the wumpus
|: w.

true.

?- hunterAt(r(X,Y),Z).
X = 1,
Y = 3,
Z = w.

?- wumpus(r(X,Y)).
X = 1,
Y = 2.

?- pit(r(X,Y)).
X = 2,
Y = 3 ;
X = 3,
Y = 1 ;
X = Y, Y = 3 ;
X = 3,
Y = 4 ;
X = 4,
Y = 3.

?- gold(r(X,Y)).
X = 1,
Y = 4.

?- breeze(r(X,Y)).
X = 1,
Y = 3 ;
X = 2,
Y = 1 ;
X = Y, Y = 2 ;
X = 2,
Y = 3 ;
X = 2,
Y = 4 ;
X = 2,
Y = 4 ;
X = 3,
Y = 2 ;
X = 3,
Y = 2 ;
X = Y, Y = 3 ;
X = Y, Y = 3 ;
X = Y, Y = 3 ;
X = 3,
Y = 4 ;
X = 4,
Y = 1 ;
X = 4,
Y = 2 ;
X = 4,
Y = 3 ;
X = Y, Y = 4 ;
X = Y, Y = 4.

?- stench(r(X,Y)).
X = Y, Y = 1 ;
X = 1,
Y = 3 ;
X = Y, Y = 2 ;
false.

?- safeWumpus().
false.
```



```
?- wumpusAlive().
false.

?- hasArrow().
false.

?- setHunterAt().
Please enter the X coordinate of the hunter:
|: 1.
Please enter the Y coordinate of the hunter:
|: 4.
Please enter the initial direction of the wumpus
|: e.

true.

?- grabGold().
true.

?- gold(r(X,Y)).
false.

?-
```

IV. The Performance Rate:

To calculate the performance of the game, I will mainly measure the performance of 4 functions, safe, grabGold, shootWumpus, and shootWumpusWithDirection, to asses the performance of the game. The following table provides a summary of the calculations:

	Config 1.	Config 2.	Config 3.	Config 4.	Config 4.	Average
safe	1	1	1	1	1	1
grabGold	_	1	1	1	1	1
shootWumpus	_	1	_	_	0	0.5
shootWumpusWithDirection	_	1	0	1	_	0.67
Average	1	1	0.67	1	0.67	0.87

We can see that bot functions shootWumpus and shootWumusWithDirection provide wrong answers in some configurations in which the hunter shot before and did not kill the wumpus. Our code assumes that the hunter has infinite arrows which is the wrong assumption.

V. Limitations of the code

Problem or limitation: The first limitation that we can identify in the code is the fact that it assumes that the Wumpus has infinite arrows, which is the wrong assumption.

Solution: in the initialization of the configuration, we add the hasArrow() predicate, once the first shoot is executed this predicate must be retracted from the knowledge base.

Problem or limitation: In the perception of a breeze or a stench, we still perceive a breeze or a stench at a room although a pit is there.

Solution: Adding a condition of no pit, \+pit(r (X, Y)), if a breeze is detected.

Problem or limitation: The user infers the best action by asking all the possible actions, and in some cases, they will lose the game due to wrong judgment or inability to predict what can happen.

Solution: adding heuristics to the game to make sure the user has also visibility over reaching their goal.

Problem or limitation: The code does not assume the safety of the adjacent rooms, but rather checks the existence of the wumpus and the pits.

Solution:

The following functions check pit and wumpus certainty:

```
checkPitsCertainty(RP,[H|T],S) :-
   breeze(H,S),
   (
        (

        getAdjacentRooms(H,LA),

        trimVisited(LA,S,LT),

        trimWall(LT,S,LT2),

        LT2 = [RP]

        )

        ; checkPitsCertainty(RP,T,S)

      ).

checkWumpusCertainty(RW,do(A,S)) :-

      checkWumpusCertainty (RW,S);

      ( setof(R,stench(R,do(A,S)),[H|T]),

      getAdjacentRooms(H,LA), trimVisited(LA,S,LAT), trimNotAdjacent(LAT,T,LT),

      LT = [RW]).
```