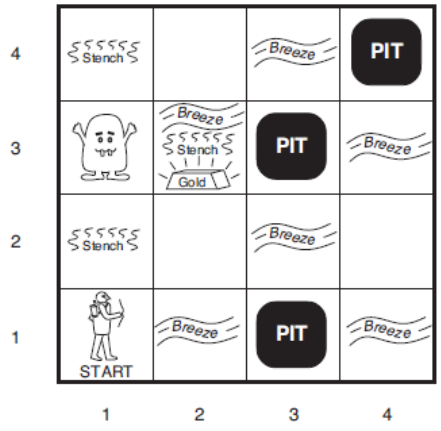
THE COLLEGE OF SAINT ROSE
CSC 535: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

PROJECT THREE

REQUIREMENTS:



For this assignment, you will be implementing the game *Hunt the Wumpus* using a knowledge base with first-order logic. Your game will be completely non-graphical and will use command-line input to interact with the user. You will also be implementing a "helper agent" that will consult a knowledge base, derive new sentences and give hints to the player.

Setup:
The initial "game board" will be loaded from a file specified at runtime (for example "wumpus.txt"). The file will contain *N* rows of *N* comma-separated values detailing the setup of the board. For example, the familiar game board configuration to the right would be represented as:

```
X,X,X,P
W,G,P,X
X,X,X,X
X,X,P,X
```

where P = pit, W = Wumpus, G = gold and X = blank square (note that we are using a simplified version of the game where the Wumpus cannot be in the same square as the gold or a pit).

Your program will read the board and then *insert* the perceptions in each room, as appropriate. For example, based on the above game board, you would insert the percept "Breeze" in [2,1], "Stench" in [1,2], and "Breeze", "Stench" and "Glitter" in [2,3]. You can insert "Bump" perceptions if the user tries to move off of the board.

Game Play:
The game should follow the following pseudo-code:

```
DEAD = false;  WIN = false;
repeat {
  display_details(position(player));
  move = getMove();
  // code here should act based on the value of move
  if (position(player) == position(Wumpus)) then DEAD = true;
  if (position(player) == position(Gold)) then WIN = true;
  add_Percepts_To_KB(percepts);
  output (get_best_move_list(KB));
} until (DEAD or WIN)
```

Your program should solicit the following possible inputs from the user:
R // turns player right 90 degrees
L // turns player left 90 degrees
F // moves player forward
S // for shooting the arrow
For example, using the game board above, here's the console interaction for 6 moves starting from the beginning:

```
You are in room [1,1] of the cave.  Facing EAST.
What would you like to do? Please enter command [R,L,F,S]:
> F
You are in room [2,1] of the cave.  Facing EAST.
There is a BREEZE in here!
HINT:  There may be a PIT in [3,1] or [2,2]
What would you like to do? Please enter command [R,L,F,S]:
> L
You are in room [2,1] of the cave.  Facing NORTH.
There is a BREEZE in here!
HINT:  There may be a PIT in [3,1] or [2,2]
What would you like to do? Please enter command [R,L,F,S]:
> F
You are in room [2,2] of the cave.  Facing NORTH.
HINT:  There is a pit in [3,1]
What would you like to do? Please enter command [R,L,F,S]:
>
… and so on…
```

You do not need to output the "observed board so far". It is assumed that the user is playing with a pencil and graph paper handy. Let them track their own exploration! Also, you should output the BUMP perception if the player attempts to move through a wall. For example, give the game board above, the following should happen.

```
You are in room [4,4] of the cave.  Facing EAST.
What would you like to do? Please enter command [R,L,F,S]:
> F
BUMP!!!  You hit a wall!
You are in room [4,4] of the cave.  Facing EAST.
What would you like to do? Please enter command [R,L,F,S]:
>
```

Recall that shooting your arrow (you have only one) carries the arrow all the way across the cave until it hits either the Wumpus or a wall. If the Wumpus is killed, your program should give off the SCREAM percept and "remove" the Wumpus from the game board by updating your knowledge base. Also, if at any time the player enters the room where the GLITTER is present, the game is automatically won! You do not need to implement the "Grab" actuator.

You do not need to keep score.

THE KNOWLEDGE BASE:
You must store statements formatted in first-order logic (you may choose your own representation symbols). Your knowledge base must start with the fundamental axioms of the Wumpus game. As observations are made, you should simply enter these into the knowledge base. After each entry has been added to the knowledge base, you should entail any new sentences and add them as well.

The knowledge base should be written to a file so that it can be verified. Ideally, you should call this file "kb.dat" or something obvious.

SOME NOTES / HINTS

- ➔ Note that, although the game board above was 4 x 4, the game board can be any of *any* N x N size. Make sure you use the appropriate dynamic data structures to handle the file input
- ➔ You may assume that the board is solvable. When we grade your program, we will use boards that include a possible path to the goal. Sample input boards will be posted
- ➔ Once you find the goal, you may end the game. No need to make the user trace back to the start of the cave
- ➔ The game will always start with the player located in [1,1] facing EAST.
- ➔ You can use whatever data structure you feel is appropriate to keep track of your knowledge base (KB). My BIG HINT would be to use an N x N array of records, where each record contains the *perceived* environment explored at the current time. You can then "update" any entry [i,j] based on inference(s) after examining the entire array.

ADDITIONAL REQUIREMENTS:
Your program must be robust. For example, your program should not crash and burn if the user enters an invalid move or attempts to walk through a wall.

There should be some elegance to the code you write and must contain some documentation. Your may lose a point if your code looks "hacked".

PROGRAMMING LANGUAGES ALLOWED:
You may use any standard programming language to complete this assignment (Java, C, C++, C#, Python).

SUBMISSION:
Please submit all of the source code for your assignment, along with your code walk-though video. You must also include a readme.txt file that explain how to get your program running just in case there are any issues. Also explain any non-standard user-interface issues.