Programming Assignment 04: Logic Programming (50 points)

In this assignment in addition to Python you will work with Answer Set Programming tool called clingo that was discussed in class.

The assignment is to implement a simple version of the Wumpus world game using a Knowledge Based Agent

**The software:**

ASP

Go to <https://potassco.org/>

Under Systems, click on clingo and then download. You should get here: <https://github.com/potassco/clingo/releases>

Scroll down and find what you need for your operating system. Download the archive and extract somewhere on your computer. The clingo executable should be there.

There are quite a few options. The main options you should know about are

--help to list all the options

--models 0 to return all the answers

The main web site has links to documentation/user guide.

I recommend that you play with the Map Coloring program and the other programs that I went over in class to review the ASP approach. Especially the choice statements: MIN { predicate : condition} MAX. These are very important as they let you generate sets of all possible solutions. Try changing the Map Coloring program slightly and observe the effects. Then move on to solving the problem in this assignment.

Python wrapper

I use this one: <https://github.com/aluriak/clyngor>

I was able to install it using pip install clyngor command.

The only feature that is necessary to run ASP from within python is the solve function:

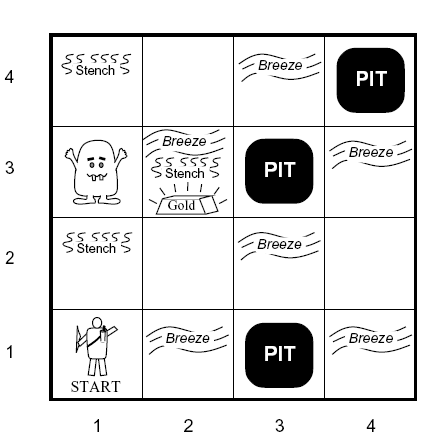
*import clyngor*

*answers = clyngor.solve(ASPFile)*

answers contains a list of tuples, each tuple corresponding to an answer set. You need to iterate through each answer set to get to individual predicates.

**The problem:**

Recall the Wumpus world game that we discussed in class and that is also described in the textbook. In this assignment you are asked to implement a simple version of the game, using an ASP knowledge base to store facts and rules that govern the Wumpus world.



**Rules of the Game**

1. The game happens on an n by n field, n can be arbitrary
2. Use the cell indexing (x,y), where x goes left to right and y bottom up. In the figure above, the wumpus is in cell (1,3)
3. Each cell can contain: nothing, pit, wumpus, gold
4. The agent starts at arbitrary cell, In the figure above the starting position is (1,1)
5. The agent is allowed the following actions: move right, left, up, down, grab gold.
6. Of course the agent can’t move off the field
7. If the agent steps in the cell with wumpus or a pit it dies
8. If the agent steps in the cell with gold, it grabs the gold and wins the game
9. The four neighboring cells around wumpus smell
10. The four neighboring cells around pit have breeze
11. The cell with gold has glitter
12. The agent can perceive smell, breeze, and glitter of the cell it is in
13. The agent remembers its previous percepts and the cells it already visited

**Implementation**

The game will be implemented as a Knowledge Based agent. Define a class for the agent, name it KBAgent, that encapsulates all the functionality. The two main functions of the agent are 1) receive and save percepts from the environment, and 2) decide on and execute the next action. The main program will contain a loop alternating these two function calls until the game is over.

Use ASP to store all the requires knowledge. You may want to use several .gr files, for better organization and to make the KB updates easy.

1. Game configuration: positions of pits, gold, wumpus
2. Current position of the agent
3. History of positions and percepts of the agent
4. Percept rules: what percepts exist in the current position of the agent
5. Termination rules: is the game over due to pit, wumpus or gold?
6. Action rules: what is the best action to take from the current position

The action rules is where you need to generate possible moves. In general there are 4 possible moves however some of them may be bad because of the possible wumpus of pit. The current and historical percepts should help decide which moves are better. Your ASP script may return more than one good move in which case you may decide which one to take by random choice inside the python code. In my implementation I decided to categorize the moves into safe and risky. You may or may not do the same, but you will need to explain how the agent decided on the next move.

The game configuration must me specified in a separate file that looks like this:

% World configuration

wumpus(1,3).

pit(3,3).

pit(3,1).

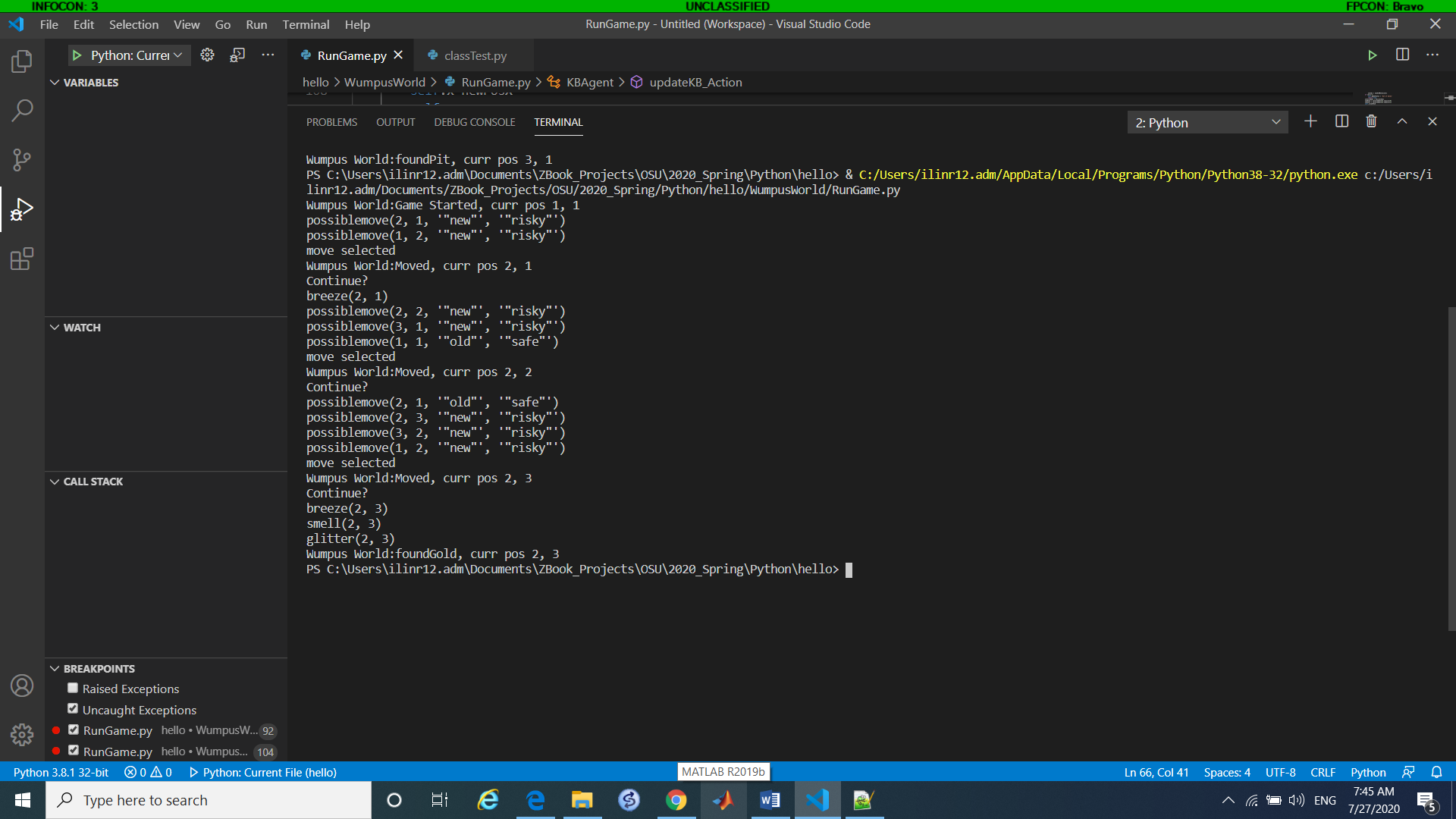
pit(4,4).

gold(2,3).

You must use these predicates as for grading we will substitute your configuration with a different one and test your code.

**Execution**

Below is a screen shot of my game run. Print the percepts, current possible moves, then selected move and then pause the game for the user to understand what’s going on before allowing the agent to make the next move.



**Graphical Display (Extra 10 points)**

If you enjoy coding and want to earn extra credit, you may implement a graphical display for the game. If you do have a GUI, make sure it has a button to advance the game one move at a time.

**Code compactness**

Please strive to make your code as short and simple as possible. My implementation had about 130 lines of python code. The ASP scripts should also be pretty short.

Submission

Submit your solution with a readme file with clean instructions how to run your program and the game strategy. The strategy must explain how your agent decided on the next action.

The grader will test the program with your game configuration file first, and then run the solution against several other configuration files provided by the instructor. If the game behaves reasonably, you get the credit! The grader will also review the code to make sure it is organized correctly. Messy/unreadable code will result in reduced points.