CMSC 671 (Introduction to AI) – Fall 2016

Homework 2: Jumping Frogs and Ricocheting Robots (70 points)
Due: 10/3 at 11:59pm.
Turnin: Blackboard.

Please submit Parts I and III together as a **single PDF file** named *yourlastname*_hw2.pdf, with Parts I and III clearly marked and delineated.

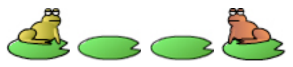
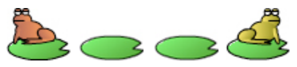
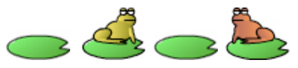
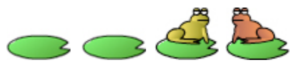
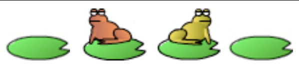
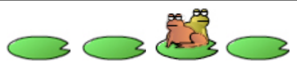
Please submit Part II as two files: a **.py** file, named *yourlastname*_hw2.py, and a file based on our GUI code named *yourlastname*_ricochet.pyde.

All files must start with your last name and have your full name in the file, at/near the top.

Part I. Jumping Frogs (Search Spaces and States) (20 pts)

We are going to formulate the "jumping frog" puzzle as a search problem. (There are a number of flash implementations online, if you want to play with it.[1]) In general, two sets of frogs are trying to get to the opposite ends of a single row of lily pads. Each frog can jump forward either one or two spaces, but cannot jump onto an occupied pad. Frogs cannot jump backwards or turn around.

We are going to discuss a very simple jumping frog puzzle, in which we have two frogs (green and brown), separated by two empty lily pads. Here are some possible states:[2]

Start (*S*):		Goal (*G*):	
Legal state:		Legal state:	
Legal state:		Illegal:	

Assignment: Answer the following questions about searching the space of this puzzle.

1. How many **unique, legal, reachable** states are there in this search space?
   (It can help to draw them out, e.g., [G _ _ B] → [_ G _ B] ...)
2. What are four operations that fully encode this search problem?
   (a) Operation 1:
   (b) Operation 2:
   (c) Operation 3:
   (d) Operation 4:
3. Are there any loops in this search space?
4. Given the order of operations above (e.g., the search space is expanded in a-b-c-d order), how many states are visited in a depth-first search?
5. If you were using heuristic search, what heuristic would you use for evaluating states?
6. Of the uninformed, informed, and local search methods we have looked at:
   (a) What algorithm would you choose for jumping frog problems in general?
   (b) Why is this algorithm a good choice? (3–4 sentences)

---

[1] *Such as: http://britton.disted.camosun.bc.ca/frog_puzzle.htm.*

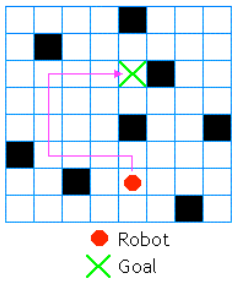
[2] *Frog graphics and some notes from http://britton.disted.camosun.bc.ca/frog_puzzle_sol.htm.*

PART II. RICOCHET ROBOTS[3] (40 PTS.)

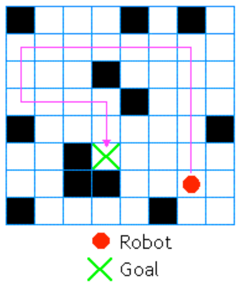
The General Idea

This is a puzzle program, based very loosely on the game Ricochet Robot. The idea is to get your piece (representing a robot) to a particular space on the game board in as few moves as possible.

- The ***puzzle*** consists of:
  - A board with blocks (obstacles) on it;
  - A goal (target) location, and;
  - A starting location for the robot.
- The rules:
  - The robot can move in any of four directions—left, right, up, or down.
  - When the robot moves, it must continue to move in a straight line until it is stopped by a block or by the edge of the playing board, or it reaches the goal.
  - The goal is to get the robot to the target location via the shortest path possible.



Here's a problem with a four-move solution.



Here's a problem with a five-move solution (but you can do better).

The Search Space

- The states are the possible configurations of the playing board.
  - Since the board is fixed for any given puzzle, and only the robot moves, the state is most easily represented by the robot's row, column coordinates.
- The operators cause a transition from one state to another.
  - For this, they are left, right, up, and down.
  - No more is needed since the robot has no choice where it stops!
- The goal state is one in which the robot's position is the same as the target position.
- A ***solution*** consists of a *sequence of states* describing a path from the start state to the goal state.

Since you want the shortest (fewest squares traversed!) solution, you will want to choose an algorithm that guarantees optimality. Since this is a **graph search** rather than a **tree search**, keep track of states you have already visited to avoid getting stuck in a loop.

The GUI

We are providing a GUI for Ricochet Robots; **if you choose not to use it, it is your responsibility** to present the results of your program in a way that can readily be graded. You will need to install **Processing 2.2.1,** a lightweight IDE for Python development. You can download and install Processing (2.2.1!) from processing.org.

---

[3] *Patterned after materials by Dr. David Matuszek at University of Pennsylvania, with thanks.*

Processing adds a couple of hundred methods, two of which are setup() and draw(). The former is where the program starts, the latter is then called 60 times a second. Once installed, you can run the GUI by opening it and clicking the triangle in the top left. (The square stops the program.)

In ricochet.pyde, you will find a block where you should make changes, including a file from which to read in a puzzle from a file (this is what we will use to test your program). A sample test file is included. Please don't change the rest of ricochet.pyde.

Materials can be downloaded from:
www.csee.umbc.edu/courses/graduate/671/fall16/Homework/hw2-ricochet.zip

Assignment: Implement the search described above.

PART III. PROGRAMMING CHOICES (10 PTS)

Assignment: Answer the following questions (1-4 sentences per question).

1. What search algorithm did you implement in Part II?
2. Why did you choose that algorithm? What advantages does it have for this problem? What disadvantages?
3. How fast does your program run on a 50x50 map with 10 blocks? Why?
4. If you had to use a ***heuristic*** search algorithm for Ricochet Robots, what would you use? Why?
5. What would you do differently if you started over?