Homework 1
Extra Credit
Computer Science
B351 Spring 2017
Prof. M.M. Dalkilic

James Gregory

January 20, 2017

All the work herein is mine.

Introduction

The aim of this homework is to get you acquianted with problem solving and the steps (Real World → Concept → Logic → Implementation). You will turn-in three files

- A *pdf with the written answers called `h1.pdf`
- A Python script called `p1.py` for the drone scanning the floor plan
- A Python script called `p2.py` for rock-paper-scissors

. If you've attempted extra credit, add the comment #ExtraCredit to the programs and ExtraCredit to the homework near the top so it's visible and obvious. I am providing this LaTeX document for you to freely use as well. Please enjoy this homework and ask yourself what interests you and then how can you add that interest to it! Finally, each homework question is worth 100 points.

Homework Questions

1. Problem 3.2 in the text.

   (a) Being able to move in any distance, in any of four directions, from any given point leaves the state space for this problem infinite.

   (b) Taking into acount that the robot only needs to change direction at intersections of two or more corridors, but still allowing totally unrestricted movement, still leaves us with an infinite state space.

   (c) Only including meaningful movement, we get a state space of n, where n equals the number of intersections of two or more corridors. This includes transitions that return to their origin state space, but not facing different directions from the same position.

   (d) Three simplifications from the real world we made were:

      i. Only move North, East, South, or West.

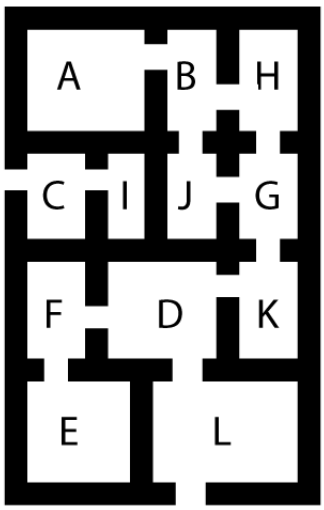
      ii. Always move the distance to the next intersection of two or more corridors.

      iii. Only change directions at intersections of two or more corridors.

2. Define *Artificial Intelligence* as rational behavior. What is a problem that AI is *not* suited for?

   AI, defined at rational behavior would be ill-suited for comprehending natural human speach, where one word can hold various meanings.

3. Assume you're programming an office security drone who's mission is to move through a floor, checking for any movement. You've been given the floor plan shown below. There are two entrances to the floor: L, C.



   (a) Using a graph $G = (V, E)$, model this plan so that the drone can navigate the floor. (*hint*) You'll need to have an additional vertex that represents the outside–call this vertex U. G is an *undirected* graph (edges have no direction). And edge is now either $\{u, v\}$ or $\{(u, v), (v, u)\}$. This impacts the implementation. For an adjancency list, you'll have to decide whether to have one edge or both.

   $G = (V, E)$
   $V = \{A, B, C, D, E, F, G, H, I, J, K, L, U\}$
   E = {(A,B),(B,H),(B,J),(C,I),(C,U),(D,F),(D,K),(D,L),(E,F),(G,H),(G,J),(G,K),(L,U)}

(b) Using an adjacency list, model this floor plan.

((A,B), (B,A,H,J), (C,I,U), (D,F,K,L), (E,F), (F,D,E), (G,H,J,K), (H,B,G), (I,C), (J,B,G), (K,D,G), (L,D,U), (U,C,L))

(c) Trace a DFS on G starting at U.

U, U-C, U-C-I
U, U-L, U-L-D, U-L-D-F, U-L-D-F-E, U-L-D-K, U-L-D-K-G

(d) The drone takes 4 minutes to scan a room and 2 minutes to move from one room to another. The drone needs 7 minutes to move from L to C (or *vice versa*. What is the approximate total time it takes for the drone to check the floor? How would you annotate the graph to reflect this cost? (*You're not asked to redo the graph–just explain what you'd do*)

103 minutes
With the weights of all but two of the edges being equal, I would probably just include the simple weight function $W(u, v) = 6 + 0.5x$, where $x =$ [number of U's in path], and not formally annotate the graph at all.

(e) A single battery for the drone holds about 45 minutes worth of charge. How many batteries does the drone need to scan the floor?

2

(f) Using Python, create an adjacency list data structure (I used a dictionary which seems reasonable); then using either a Python package or your own implementation, do a DFS from the *outside* of the floor, *i.e.,* DFS(G, U). The DFS should yield a sequence of rooms that the drone would scan.

see p1.py

(g) For extra credit, add the cost to the drone and battery life to the search; when the drone has scanned all the rooms, give the percent battery life left.

see p1.py

(h) For extra extra credit, presume the drone starts and ends at U.

see p1.py

4. Rock/Paper/Scissors is a simple game: numbers 0,1,2 denotes each of these items with an outcome that rock *beats* scissors, scissors *cut* paper, and paper *covers* rock; otherwise it's a tie. Included below (and as file) is a Python script that allows a human to play with a computer.

(a) Play 5 times of 10 games (it will go quickly) and record the statistics for your wins, $H_1, H_2, \ldots, H_5$. *Expectation* gives us a measure of central tendency (sometimes called the mean). We can calculate the expectation as

$$\mathrm{E}[H] \quad = \quad \sum_{i=1}^{5} H_i/5$$

What is the expecation of a human winning? Are you surpised? Of the computer winning?

$[H_1, H_2, \ldots, H_5] = [0, 0, 1, -1, 0]; 0 = tie, 1 = win, -1 = loss$
$E[H] = 0$
I'm not surprised about an average of a tie. The simple AI used for the computer player merely chooses a psuedo-random - with independent, uniform distribution - play each game. This means each player always has 1/3 chance to win, and 1/3 chance to tie.

(b) From `python.org` look-up the random function. How is it picking the numbers 0,1,2? Does the computer's choice of numbers depend on its previous choice? How about a yours?

Python's random function generates a random float uniformly in the range [0.0, 1.0), using the Mersenne Twister as the primary generator. It then uses that float to determine which element of the provided sequence of random.randrange to select. These numbers are independent of one another, but depend on some other constant. This causes multiple instances of random running simultaneously to show less evenly distributed uniformity.
My choices were dependent on my first choice, as I simply chose the same play each game.

```
import random as rn

#values of rock, paper, scissors
r,p,s = 0,1,2
#dictionary e.g., rock beats scissors
ws = {r:s, p:r, s:p}

nogames = int(input("Number of games? "))

totgames = 0
compwins = 0
humwins = 0
ties = 0

gamehistory = []

while totgames < nogames:
     human = int(input("r=0,p=1,s=2 "))
     comp = rn.randrange(0,3,1)
     gamehistory.append([human, comp])

     print("Human: {0}, Comp: {1}".format(human, comp))

     if ws[comp] == human:
        compwins += 1
     elif ws[human] == comp:
        humwins += 1
     else:
        ties += 1
     totgames += 1

v = list(map(lambda x: 100*x/totgames, [compwins, humwins, ties]))
print("Stats\ncw% = {0}, hm% = {1}, ties% = {2}".format(*v))
```

(c) Remove the human player and add another computer player called Robby. See if you can find a strategy that has Robby's expected winning greater than the computer. You are free to employ whatever you'd like so long as it's yours.

It's kind of cheap, but given the parameters, it's the best play strategy. *see p2.py*