CMSC 170 MACHINE PROBLEM 4
Multi-Agent Pacman

Total: 60 points (15% of grade)
Deadline: May 29, 2018 (Tuesday)

INTRODUCTION

The fourth Machine Problem is another one of the *Pacman Projects* from UC Berkeley's CS188 Artificial Intelligence class (http://ai.berkeley.edu/multiagent.html), which will test your abilities and understanding of *adversarial search*, *search in uncertainty*, and *evaluation functions*. You will be working in groups with 2 or 3 members, and you are allowed to choose your group members.

In this project, you will be implementing **evaluation functions** for a reflex agent (Q1) and a planning agent (Q5), which will take a state and assign it a score that estimates how much you want your agent to be in that state (higher score = better state). You will also be implementing search algorithms for multi-agent settings - **minimax** search (Q2), minimax search with **alpha-beta pruning** (Q3), and **expectimax** search (Q4).

FILES AND SOFTWARE

All the codes and supporting files needed for this project are included in the zip file (credits to UCB CS188 for making this available to the public), including the original instructions from UC Berkeley CS 188. You will need **Python 2** installed to run the code. As usual, an **autograder** is included to automatically test the correctness of your codes.

You only need to edit **multiAgents.py** from the given codes. For extra function definitions, please write them in a new file named **extra.py**, and import as needed.

Please read the original instructions (**pacman_multiagent.pdf**) for a detailed discussion of each problem. The multiAgents.py file has been *modified* to include tips and suggestions for each problem.

SCORING

The scores from the autograder will be multiplied by 2. Code comments (5pts) and punctuality of submission (5pts) will be checked too.

Q1	4	x2	8	
Q2	5	x2	10	
Q3	5	x2	10	
Q4	5	x2	10	
Q5	6	x2	12	
Comments			5	Explain the major steps in your solution on your code
Punctuality			5	Late submissions will get zero points
Total			**60**	

Academic Dishonesty. Work on this project as a group. Hopefully the tips given will be sufficient to help you solve the problems. Please do not submit code that you copied from another group or from a source you found online (and you slightly modified). Copied codes will automatically be give a score of zero.

SUBMISSION

1. Create a **zip file** containing *multiAgents.py*, and *extra.py* (if applicable).
2. Please follow this filename format for the zip file: *lastname1_lastname2_lastname3.zip*
3. Email your output to jrdaradal@up.edu.ph on or before **May 29, 2018 (Tuesday), 11:59 PM.**
4. Please use this format as the subject: **[CMSC 170 - MP4 - Lastname1 / Lastname2 / Lastname3]**
5. Late submissions will not receive points for punctuality.
6. **Hard deadline**: Submissions after **May 30, 2018 (Wednesday)** will no longer be accepted. Failure to submit will result in a grade of INC.
7. Rate yourself and your groupmates (1-10) and submit your ratings to the instructor.

Maximize your expected utility.

DEPTH-LIMITED MINIMAX SEARCH

```
function minimax-value(state, depth):
        if cutoff(state, depth):    evaluation_function(state)
        if state is terminal state: value(state)
        if state is max-node:       max-value(state, depth)
        If state is min-node:       min-value(state,depth)

function max-value(state, depth):
        v = -∞
        for action,next_state in successors(state):
                next_v = minimax-value(next_state, depth+1)
                v = max(v, next_v)
        return v

function min-value(state, depth):
        v = ∞
        for action,next_state in successors(state):
                next_v = minimax-value(next_state, depth+1)
                v = min(v, next_v)
        return v
```

ALPHA-BETA PRUNING

```
function minimax-value(state, depth, alpha, beta):
        if cutoff(state, depth):    evaluation_function(state)
        if state is terminal state: value(state)
        if state is max-node:       max-value(state, depth, alpha, beta)
        If state is min-node:       min-value(state,depth, alpha, beta)

function max-value(state, depth, alpha, beta):
        v = -∞
        for action,next_state in successors(state):
                next_v = minimax-value(next_state, depth+1, alpha, beta)
                v = max(v, next_v)
                if v > beta: return v
                alpha = max(alpha, v)
        return v

function min-value(state, depth, alpha, beta):
        v = ∞
        for action,next_state in successors(state):
                next_v = minimax-value(next_state, depth+1, alpha, beta)
                v = min(v, next_v)
                if v < alpha: return v
                beta = min(beta, v)
        return v
```

EXPECTIMAX SEARCH

```
function exp-value(state, depth):
        v = 0
        for action,next_state in successors(state):
                next_v = expectimax-value(next_state, depth+1)
                prob = probability(action)
                v += prob * next_v
        return v
```