


Homework 5

Due Sunday by 11:59pm **Points** 10 **Submitting** a file upload **File Types** py
Available Mar 7 at 2:45pm - Mar 25 at 11:59pm 18 days

Introduction

In this project, you will design agents for the classic version of Pacman that includes ghosts.

As in the previous projects, an autograder is provided for you to grade your answers on your machine.

The code for this project consists of several Python files. You can download all the code and supporting files as a zip archive file: [homework5.zip](#) 

File you'll edit and submit: multiAgents.py

You will fill in portions of multiAgents.py. You should submit this file with your code and comments. Please do not change the other files in this distribution or submit any of the original files other than this file.

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation -- not the autograder's judgements -- will be the final judge of your score.

Academic Dishonesty: Your code will be checked for plagiarism. If you copy someone else's code and submit it with minor changes, it will be easy to detect and it will lead to a 0 on the assignment. I trust you all to submit your own work only; please don't let me down.

Getting Help: You are not alone! If you find yourself stuck on something, please ask for help. Office hours and the discussion forum are there for your support; please use them. I want these projects to be rewarding and instructional, not frustrating and demoralizing. But, I don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

Question 1 (5 points): Minimax

Your task is to implement an adversarial search agent in the MinimaxAgent class stub provided in multiAgents.py. Your minimax agent should work with any number of ghosts, so you'll have to write an algorithm that is slightly more general than what we've previously seen in the lecture. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.

Your code should also expand the game tree to an arbitrary depth. Score the leaves of your minimax tree with the supplied self.evaluationFunction, which defaults to scoreEvaluationFunction. MinimaxAgent extends MultiAgentSearchAgent, which gives access to self.depth and self.evaluationFunction. Make sure your minimax code makes reference to these two variables where appropriate as these variables are populated in response to command line options.

Important: A single search ply is considered to be one Pacman move and all the ghosts' responses, so depth 2 search will involve Pacman and each ghost moving two times.

Grading: Your code will be checked to determine whether it explores the correct number of game states. This is the only reliable way to detect some very subtle bugs in implementations of minimax. As a result, the autograder will be very picky about how many times you call GameState.generateSuccessor. If you call it any more or less than necessary, the autograder will complain. To test and debug your code for this question, run

```
python autograder.py -q q1
```

This will show what your algorithm does on a number of small trees, as well as a pacman game.

Hints:

- Pacman is always agent 0 (self.index value), and the agents move in order of increasing agent index.
- All states in minimax are GameStates, either passed in to getAction or generated via GameState.generateSuccessor.
- The correct implementation of minimax will lead to Pacman losing the game in some tests. This is not a problem: as it is correct behavior, it will pass the tests.
- The evaluation function is already written for you.

Layouts to try:

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
python pacman.py -p MinimaxAgent -l smallClassic -a depth=2
python pacman.py -p MinimaxAgent -l smallClassic -a depth=3
python pacman.py -p MinimaxAgent -l mediumClassic -a depth=2
python pacman.py -p MinimaxAgent -l openClassic -a depth=2
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

Observations:

- The minimax values of the initial state in the minimaxClassic layout are 9, 8, 7, -492 for depths 1, 2, 3 and 4 respectively. Note that your minimax agent will often win despite the dire prediction of depth 4 minimax.

- Note that going beyond a depth of 2 is impractical for all but the smallest mazes.
- On larger boards such as openClassic and mediumClassic, you'll find Pacman to be good at not dying, but quite bad at winning. He'll often thrash around without making progress. He might even thrash around right next to a dot without eating it because he doesn't know where he'd go after eating that dot. A better evaluation function would get better results here. Can you think of a better evaluation function?
- When Pacman believes that his death is unavoidable, (as in the trappedClassic maze) he will try to end the game as soon as possible because of the constant penalty for living. Sometimes, this is the wrong thing to do with random ghosts, but minimax agents always assume the worst.

Question 2 (5 points): Alpha-Beta Pruning

Your task is to implement a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree, in AlphaBetaAgent. Again, your algorithm will be slightly more general than the pseudocode from the lecture, so part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.

Layouts to try:

```
python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
python pacman.py -p AlphaBetaAgent -l smallClassic -a depth=3
python pacman.py -p AlphaBetaAgent -l mediumClassic -a depth=3
python pacman.py -p AlphaBetaAgent -l openClassic -a depth=3
```

Observations:

- The AlphaBetaAgent minimax values should be identical to the MinimaxAgent minimax values, although the actions it selects can vary because of different tie-breaking behavior. Again, the minimax values of the initial state in the minimaxClassic layout are 9, 8, 7 and -492 for depths 1, 2, 3 and 4 respectively.
- You should see a speed-up (perhaps depth 3 alpha-beta will run as fast as depth 2 minimax).

Grading: Your code will be checked to determine whether it explores the correct number of game states. As a result, it is important that you perform alpha-beta pruning without reordering children. In other words, successor states should always be processed in the order returned by GameState.getLegalActions. Again, do not call GameState.generateSuccessor more than necessary.

You must not prune on equality in order to match the set of states explored by the autograder.

The pseudo-code below represents the algorithm you should implement for this question.

α : MAX's best option on path to root
 β : MIN's best option on path to root

```
def max-value(state,  $\alpha$ ,  $\beta$ ):
    initialize v =  $-\infty$ 
    for each successor of state:
        v = max(v, value(successor,  $\alpha$ ,  $\beta$ ))
        if v >  $\beta$  return v
         $\alpha$  = max( $\alpha$ , v)
    return v
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):
    initialize v =  $+\infty$ 
    for each successor of state:
        v = min(v, value(successor,  $\alpha$ ,  $\beta$ ))
        if v <  $\alpha$  return v
         $\beta$  = min( $\beta$ , v)
    return v
```

To test and debug your code for this question, run:

```
python autograder.py -q q2
```

This will show what your algorithm does on a number of small trees, as well as a pacman game.

Note that the correct implementation of alpha-beta pruning will lead to Pacman losing the game in some of the tests. This is not a problem: as it is correct behavior, it will pass the tests.

File Upload

[Google Doc](#)

Upload a file, or choose a file you've already uploaded.

3/16/2016

File:

Choose File

No file chosen

Homework 5

[Add Another File](#)

[Click here to find a file you've already uploaded](#)

Comments...

Cancel

Submit Assignment