

EdX and its Members use cookies and other tracking technologies for performance, analytics, and marketing purposes. By using this website, you accept this use. Learn more about these technologies in the [Privacy Policy](#).



[Course](#) > [Week 6](#) > [Project...](#) > p2_mul...

p2_multiagent_q4_expectimax

Question 4 (5 points): Expectimax

Minimax and alpha-beta are great, but they both assume that you are playing against an adversary who makes optimal decisions. As anyone who has ever won tic-tac-toe can tell you, this is not always the case. In this question you will implement the ExpectimaxAgent, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices.

As with the search and constraint satisfaction problems covered so far in this class, the beauty of these algorithms is their general applicability. To expedite your own development, we've supplied some test cases based on generic trees. You can debug your implementation on small the game trees using the command:

```
python autograder.py -q q4
```

Debugging on these small and manageable test cases is recommended and will help you to find bugs quickly. **Make sure when you compute your averages that you use floats.** Integer division in Python truncates, so that $1/2 = 0$, unlike the case with floats where $1.0/2.0 = 0.5$.

Once your algorithm is working on small trees, you can observe its success in Pacman. Random ghosts are of course not optimal minimax agents, and so modeling them with minimax search may not be appropriate.

ExpectimaxAgent, will no longer take the min over all ghost actions, but the expectation according to your agent's model of how the ghosts act. To simplify your code, assume you will only be running against an adversary which chooses amongst their `getLegalActions` uniformly at random.

To see how the ExpectimaxAgent behaves in Pacman, run:

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

You should now observe a more cavalier approach in close quarters with ghosts. In particular, if Pacman perceives that he could be trapped but might escape to grab a few more pieces of food, he'll at least try. Investigate the results of these two scenarios:

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

You should find that your ExpectimaxAgent wins about half the time, while your AlphaBetaAgent always loses. Make sure you understand why the behavior here differs from the minimax case.

The correct implementation of expectimax will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.