Project 3 Bonus: MDPs and Reinforcement Learning

CSCI 360 Fall 2022

Released: March 11, 2022
Due: April 8, 2022

Contents

Bonus Questions

In question 8, you learned about approximate Q-learning and saw how it enabled Pacman to tackle larger environments. In particular, if you ran approximate Q-learning with the `SimpleExtractor`, Pacman learned to consistently win on the `mediumClassic` grid.

However, if you visualize Pacman's games with the following command, you would see that Pacman is overly cautious of ghosts. Even if Pacman consumes the capsules (the larger white dots) and makes the ghosts scared (i.e. the ghosts turn white and can be eaten), he still avoids eating the ghosts.

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor \
  -x 50 -n 60 -l mediumClassic
```

This becomes even more of a problem if we make the ghosts smarter, i.e. instead of random motions we can set the ghosts to pursue Pacman by using the `-g DirectionalGhost` option. If Pacman would realize that he can eat the ghosts when they are scared, he may be able to handle the smarter ghosts, and he would score more points since there is a 200 point reward for eating each ghost.

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor \
  -x 50 -n 60 -l mediumClassic -g DirectionalGhost
```

In these bonus questions, you will have the opportunity to improve Pacman's performance in the face of these smarter ghosts.

Setting Up

1. Download the starter code from Vocareum, as before.
2. Replace the `valueIterationAgents.py`, `qlearningAgents.py`, and `analysis.py` with the versions you have worked on thus far.
3. Open `bonusQuestionsAgents.py` — you will write all your code in this file.

Bonus Question 1 (2 points)

In this first question, let us teach Pacman to eat ghosts and score higher on the `MediumClassic` grid. For now, we will stick to using the default ghosts, which move in random directions. In `bonusQuestionsAgents.py`, you will find two classes to modify for this question:

- `CustomExtractor1` extracts features from the game state. Currently, this is copied from `SimpleExtractor` in `featureExtractors.py`. You will likely need to add or remove features to improve Pacman's performance. When implementing your feature extractor, you may find it helpful to refer to the `GameState` class in `pacman.py`.
- `CustomQAgent1` inherits from `ApproximateQAgent`. Here, you can modify the learning rate (alpha), discount factor, and epsilon in the constructor. You can also modify the update function. Currently, this class behaves exactly like `ApproximateQAgent`.

To visualize your agent, execute the following command, which runs 110 episodes — 100 for training and 10 for testing.

```
python pacman.py -p CustomQAgent1 -a extractor=CustomExtractor1 \
  -x 100 -n 110 -l mediumClassic -f
```

Hint

For this question, consider how you might leverage information about whether the ghosts are scared. If you have the agent index of a ghost, you can check whether it is scared with `state.getGhostScared(agentIndex)`. For instance, the following would iterate over the positions of all ghosts that are scared:

```
ghosts = state.getGhostPositions()
for i, position in enumerate(ghosts, 1):
    if state.getGhostScared(i):
        # The ghost at this 'position' is scared.
```

Grading

Your agent will train on 100 episodes and test on 100 episodes. **You will receive 1 point if you score greater than 1450 on average, and 2 points if you score greater than 1550 on average.** To grade your question, run:

```
python autograder.py -q bonus1
```

Bonus Question 2 (2 points)

In this question, let us see if we can get Pacman to handle smarter ghosts on the `MediumClassic` grid. Similar to the previous question, you will modify the `CustomExtractor2` and `CustomQAgent2` classes in `bonusQuestionsAgents.py`. To visualize your agent, execute the following command, which runs 110 episodes — 100 for training and 10 for testing.

```
python pacman.py -p CustomQAgent2 -a extractor=CustomExtractor2 \
  -x 100 -n 110 -l mediumClassic -g DirectionalGhost -f
```

Grading

As in the previous question, your agent will train on 100 episodes and test on 100 episodes. You will receive 1 point if your agent wins on at least 60 of the 100 episodes.

Furthermore, a second point will be based on your performance on a class leaderboard. After the due date, we will look at all submissions for this bonus question and let $x =$ maximum win rate on this question. Let $y = 56/100$, which is the baseline win rate for our approximate Q-learning agent. We will then normalize your score to the range $[0, 1]$, based on where it lies in the range $[y, x]$. Agents which score 56 or less will not receive an extra point from the leaderboard.

Note that the autograder only assigns up to one point for this bonus question, since the second point will be determined later.

You can run the autograder on this question with:

```
python autograder.py -q bonus2
```

Grading Clarification

Project 3 is scored out of **21 points**. You have the opportunity to earn up to **25 points** since there are two bonus questions worth 2 points each. Since the last point of Bonus Question 2 is based on a leaderboard, you can only earn up to **24 points** before the deadline. The final point will be determined after the deadline based on the leaderboard. **As long as you attain 21 points or more on this assignment, you have earned full credit.**

Submission

In addition to your files from the main assignment (`valueIterationAgents.py`, `qlearningAgents.py`, and `analysis.py`), upload `bonusQuestionsAgents.py` to Vocareum and submit. You are welcome to submit as many times as you'd like before the due date!

Note: Make sure you submit on Vocareum, not on Blackboard.