Assignment 3

Piergiuseppe Mallozzi
Alessia Knauss
(Group 27)

2. What happens?
The learning takes too long and the agent seems to not be really learning (the average reward is getting worse). We do not feel the agent is learning. It take exponentially more time for every 100 episodes.

Why?
There are too many states to update our q-value table. Also, our implementation of the qlearningAgents at this point is not optimal, We use a basic dictionary and for each (state,action) pair we check every entry of the dictionary (that we will get bigger and bigger as we keep adding new (state, action) keys).

Can you fix this?
Not with exact Q-learning. We need to use approximation and sampling.

What does it tell you about exact Q-learning?
Exact Q-learning cannot be used for every kind of problem.

3. For command (3), we get a 100% winning rate

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

we get a 90% winning rate.

For
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 150 -l mediumClassic -q

we get a 94% winning rate.

The autograder passes 3 out of 3.

4.
Winning rate is 73% first time, 71 % second time, 74% third time, 73% fourth time, 66%, 79%, 74% …

Compared to (3), it performs worse. The algorithm performs worse on the small grid, than on the mediumgrid. Approximation seems not to perform well on the small grid. Why??
By watching some games it seems that the pacman agent always prefer the immediate reward and loose by getting stuck in the corner there. We have tried increasing the training episodes to 200 and it performs better (84%), then we increased them again to 400 and it got worse again! (77%)

So we think that is not about the small or the medium grid but about the structure of the grid. By observing how pacman behaves it seems it only cares about the food and the distance to the ghost. So it goes for the immediate food and gets stuck in the corner because he thinks the ghost is far.

Questions:
List of features:
- whether food will be eaten
- how far away the next food is
- whether a ghost collision is imminent
- whether a ghost is one step away

We've seen how the feature vector values are modified in different situations of the game and we've noticed that
#-of-ghosts-1-step-away -> 0.1
Only when pacman is eaten by the ghost! So when the collision is imminent (as described in the function).

We propose a better feature vector that takes into consideration the distance in steps to the ghost at all the times. Furthermore we would consider also when the ghosts are eatable (when pacman eats the big food).
We would add features to detect the number of the walls around pacmans, so we can avoid pacman to be trapped into dead ends.

We've tried to implement some "dead-end" features. While it seems to improve in the beginning we believe it doesn't change much the winning rate…. :(

```
# conf is only so that getPossibleActions(conf, walls) will get a conf.pos object
conf = Configuration((x + dx, y + dy), Directions.STOP)
possible_actions = Actions.getPossibleActions(conf, walls)
if (len(possible_actions) == 1):
  features["dead-end"] = 1.0
else:
  features["dead-end"] = 0.0
```

Piergiuseppe Mallozzi
Alessia Knauss

Group 27