ROB311 – TP4 – Q-Learning and Pac-Man

Implement Q-Learning in Python.

You will train an AI player of the famous Pac-Man arcade game.

You will work on a set of Python files and libraries provided to you by the **UC Berkeley university**, as part of their ***CS188 – Intro to AI*** course, and you will only have to write the core algorithm of the methods dealing with the Q-Learning, based on the equations you have seen in the course.

Files

The **pacman.zip** archive you have downloaded from the ROB311 course page provides you with the files you need in order to implement the Q-Learning for the Pac-Man game.

File and classes you have to modify:

- ***qlearningAgents.py:*** *QLearningAgent*

AI/Machine Learning related files and classes (you will have to look through):

- ***learningAgents.py:*** *ReinforcementAgent, ValueEstimationAgent*
- ***util.py:*** *Counter*
- ***game.py:*** *Agent*
- ***featureExtractors.py***

Objectives

1. Implement Q-Learning by modifying the following *QLearningAgent* methods in the ***qLearningAgents.py*** file:

- *__init__()*
- *getQValue()*
- *computeValueFromQValues()*
- *computeActionFromQValues()*
- *getAction()*
- *update()*

How to test your code

Navigate using a terminal to your project folder and run the following commands:

1. Q-Learning:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

This command will attempt learning from 2000 training episodes and then test the resulting AI agent (player) on 10 games. You will be able to see the AI playing the 10 test games. The learning is done on the *smallGrid* map and the results of the 10 test games will be displayed in the terminal. The win rate on the 10 test game is supposed to be very high, ideally 100%.

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

This command will show you what happens during the training process for 10 games.

If you want to test your code in other scenarios, use the following command to understand what each of the command line parameters does:

```
python pacman.py --help
```

Tips & tricks

- spend some time reading the comments wrote inside the given Python files and trying to understand the architecture of the given code, especially what classes inherit what and from whom
- most of the constants and variables you need in order to implement the algorithms (*epsilon, alpha, gamma, etc.*) are already defined or described inside the classes you have to modify or inside the classes they inherit from
- the *Counter* class, which is an extension of a Python *dictionary*, is very useful for the easy implementation of this algorithm and contains a series of methods you might need