Alessio Mazzone
CS 1571 Artificial Intelligence
Project 3 Frogger

HOW TO LOAD AND RUN TRAINED AGENT:
python Qagent.py COMMAND

Values for COMMAND:
'start' --> begins training agent from scratch
'continue' --> loads FROG.config file which contains Q-Table from training

Example:
To start training agent from scratch, type:
python Qagent.py start

To continue training agent from existing configuration file, type:
python Qagent.py continue

FORMAT OF DATA FILE:
My Q-Table is located in the file called FROG.config. I use Python's Pickle object serialization library to save this table. The format of the table is the following:

state	K_w	K_s	K_a	K_d	K_F15
s1	0.0	0.0	0.0	0.0	0.0
s2	0.0	0.0	0.0	0.0	0.0
s3	0.0	0.0	0.0	0.0	0.0

etc..

My states act as keys to my Q-Table. I made my state object hashable by invoking their def __hash__(self) property. This allowed me to use my states directly as keys to the Q-Table dictionary. The values of the Q-Table are 1x5 arrays. The array is comprised of the calculated Q values for each action (up, down, left, right, no-operation).

DESCRIBE HOW YOUR FINAL AGENT IS PERFORMING:
On most games, 4 frogs make it home, and sometimes all 5 can make it home. Rarely is there ever less than 3 frogs home. On average, the total reward the agent receives is at least 5 or greater, based on testing by printing out the total reward at each game's end.

DESCRIBE IN SUFFICIENT DETAILS THE CHOICE YOU'VE MADE TO GET THE LEARNING FRAMEWORK GOING:

Throughout all of training, I left my alpha at 0.2 and my discount at 0.8. The only thing I changed was my exploration rate. I chose my alpha values and discount values based on class discussion/notes.

The exploration rate varied between 0 and 0.8. When I began training my model based on State3, I set my exploration rate to 0.8, which is very high and means that 80% of the moves the frog made were random, with the other 20% being chosen from the Q-Table. I chose a high exploration rate at the beginning so that the frog would learn about as many states as possible. After A the first overnight training session, I changed my exploration rate to 0.4. I cut the exploration rate in half so that the frog could start learning about the states already in the Q-Table a little more heavily. But, I left a good amount of exploration at 40% so that the frog could still explore adequately. I changed the exploration rate to 0.1 before I started my second overnight session. This really helped the frog learn more about the states it had already seen while still letting it explore a little bit and solidify the Q values. By the end of the second night, most games could get 3, 4, or all 5 frogs home.

I tried three different state representations with varying levels of success. The first state representation I tried (State1 in my code) looks like this:

```
  X
X F X
  X
```

The frog can only see directly in front, behind, left, and right. I tried this first because I thought it was easiest to implement. I set my exploration rate to very high, and let it train over night. After one night of training, my frog basically learned to never leave the starting area. It didn't quite learn that cars were bad, and I though that perhaps it needed more time to train. After another night of training, my frog still just would not leave the starting area. It learned to mildly doge cars if my random move made it go forward, but that was it. I started to think about a second state representation now. My second state representation I tried (State2 in my code) looks like this:

```
    X
    X
X X F X X
    X
    X
```

This is very similar to the first state, but I extended the visibility

of the frog to include two squares in front, two behind, two left, and two right. I let this model train over night with much better results. The frog learned to go home over night. It would always go into either the middle home, or immediately to the right or left of the middle. I trained it for a second night in a row and found that one frog could get home pretty consistently. I felt like the frog had tunnel vision, and that's why only one frog could make it home. To fix this, I expanded my state representation. My third state representation I tried and settled on (State3 in my code) looks like this:

```
X X X X X
X X X X X
X X F X X
X X X X X
X X X X X
```

In this State3, the frog is able to see a 5x5 square grid around itself. Since only one of my frogs was making it home, I thought that it was not seeing any of the other homes. By expanding what the frog could see, especially in front of it, the frog could learn to see more homes in one state. That is precisely the problem with my first two state representations. State1 and State2 frogs could only see at most a single home at any given time. But, by using State3, a frog could potentially see 3 different homes in a single state. With State3, the frog could see which homes are empty, and go to those home. By traveling to an empty home, the 5x5 grid allows the frog to see the rest of the homes if it travels left or right, which is why the frog learned to get at least 3 frogs home in my model.

OVERALL, HOW LONG DID IT TAKE TO TRAIN THE FROG:
The first time I trained my State3 model, I trained it overnight (about 8 hours). At the end of this first training session, 3 frogs regularly got home. I changed my exploration parameter from 0.8 to 0.4 and let my model train for another 4 hours. After this session, 3 to 4 frogs made it home relatively easily. My third and final training session was another overnight (8 hours). By the end of this third session, 3, 4, or 5 frogs got home almost every time. The frog learned to dodge cars very well, and it learned to traverse the river in all directions in order to get to empty homes.

To make training more efficient, perhaps I could think of an altogether new state representation that would make my state space smaller. By having a smaller state space, the training time could be reduced.

OUTSIDE RESOURCES:
I used Pickle (https://docs.python.org/2/library/pickle.html) in order to serialize and easily store my Q-Table and Numpy (http://

www.numpy.org) for some math related functions.