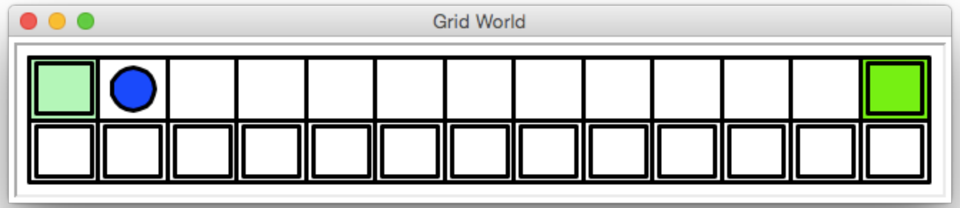
Project 3: Reinforcement Learning

(All parts are to be completed as a team)
Due [1-1.5 weeks after release]

0. The Story So Far…

Congratulations! You've just been hired at a robotics start-up. They want to use reinforcement learning to create industrial robots that adapt their behavior based on their specific work conditions, rather than coming with a preset, unchangeable control policy. Already on the first day, they need your expertise.

The engineering team was applying basic RL algorithms to a simplified version of their robot's problem, which is shown below in the form of a gridworld. The blue circle is the robot, which can move in any of the four cardinal directions (trying to move off the grid causes the agent to stay in place). The double squares are terminal states. The green shows two sources of positive reward. Entering the terminal state on the left yields 1 reward. The one on the right yields 3 reward. In order to get the bigger reward, the robot has to perform a delicate maneuver – if it deviates toward the bottom of the grid (representing an area outside the robot's workspace), the episode is terminated to prevent the robot from damaging anything (or itself!).



The engineers tried both SARSA and Q-learning with $\alpha = 0.1$ and $\gamma = 0.9$. The results were promising: both algorithms eventually learned to get the big reward. Then things got strange. It was discovered that the implementation had a couple of bugs. First, ε was always set to 0 (no matter what value the user actually specified). When they fixed this, and used $\varepsilon = 0.1$ as intended, SARSA no longer learned to get the big reward. Then it was discovered that the Q-values were all initialized to 3 instead of 0. Woops! The problem is that when they fixed that bug too, *neither* algorithm learned to get the big reward!

While most folks at this company are familiar with the basics of RL, they don't have a lot of experience applying the algorithms. That's why they hired you! Your boss wants a report on what is going on (see Parts 1 and 2).

In addition to help with this mystery, the engineering team would like some guidance about how to choose step sizes when linear value function approximation is used (see Parts 3 and 4).

1. SARSA and Q-learning (10 pts)

First, you'll need to reproduce the engineering team's results. In *rl.py* I have provided some unfinished classes for you to complete. The `QTableLearner` is the superclass, representing a reinforcement learning agent that stores its Q-values in a lookup table. The constructor sets up the table (indexed first by state, then by action) and also stores the other learning parameters. You should implement the following methods:

- `greedy(state)` – Returns a greedy action for the given state. If multiple actions have the maximum Q-value, it should uniformly randomly return one.
- `epsilonGreedy(state)` – Returns an ε-greedy action for the given state. With probability ε it should choose an action uniformly randomly. Otherwise, it should return a greedy action.
- `terminalStep(curState, action reward)` – Performs the last learning update of an episode. Both SARSA and Q-learning do the same update in this case since the Q-value of the *next* state-action pair is 0. This method should update the Q-value for `curState` and `action` using the TD update rule.

Now, in both `SarsaLearner` and `QLearner`, implement

- `learningStep(curState, action, reward, nextState)` – Performs a learning update based on the given state transition. This method should return the next action the agent wants to take (i.e. the action to take in `nextState`). The two algorithms have different update rules!

You can test out your algorithms with *gridworld.py*. Use the -h option to see the various options available to you.

It takes an input file representing a grid world: *grid.txt* contains the example above. The first row has the dimensions of the grid. In the grid itself, each entry has three comma-separated entries. The first is either "." or "#", indicating whether that space is a wall or not. The second is the reward for entering that square. The third is either "T" or "F", indicating whether that square is terminal or not.

It also takes an output file, in which it will write results from the learning process. In particular, you can specify the number of trials to run and the number of episodes per trial. The output file will contain the average results over the trials. In particular, the columns of the output file are:

- Episode number
- Average total reward
- Average discounted sum of rewards
- Average number of steps in the episode

In each trial, after training for the specified number of episodes, the program will also run the greedy policy (no learning, no ε) and report the results for that policy.

You can set parameters of the algorithm on the command line as well. For instance,
```
python3 gridworld.py -l sarsa -e 0 -i 3
```
uses SARSA with the parameters the engineering team started with (ε = 0, Q-values initialized at 3). You can even watch the agent do its thing and visualize the Q-function. For example, using the option `-d 10` will display every 10th episode.

For testing purposes I have also included *sarsavq.txt*, as seen in class. You could compare SARSA and Q-learning to see if they behave as expected. You should also consider printing out learning updates to see if they match your understanding of the update rules.

2. Exploration Report (10 pts)

Now it's time to do your real job – the team is counting on you! In this section you should produce a report in *rl.pdf* that will answer the engineering team's questions and offer some recommendations.

- Your report will be disseminated to the engineering team, so they are your main audience. After reading your report they should be thoroughly convinced by your explanation of the phenomenon.
- Another part of your audience is your boss, who you want to impress. Make sure your report is polished and professional.
- The last audience you should consider are readers in the future. Maybe a new team-member is hired, or maybe a year from now a similar question comes up and the team wants to refer back to this issue. What you are writing isn't an email, meant to be immediately discarded. This report should be self-contained enough to have value to a reader who is not fully immersed in the current problem. Don't just launch into it – set the stage!

Explaining the Results
Start by re-generating the engineering team's results. Run SARSA and Q-learning for 30 trials of 500 episodes each with each of the three parameter settings:

- ε = 0, initial Q = 3
- ε = 0.1, initial Q = 3
- ε = 0.1, initial Q = 0

Decide what of the output you want to present and produce clear, well-labeled plots of the average performance while learning. The plots should be organized to facilitate comparison of the two algorithms and of the three parameter settings. You should also report the average performance of the final greedy policies.

In your report present your results and thoroughly explain why the algorithms behave this way. There is a lot to unpack here. Make sure you address

- Why Q = 3 seems to yield better results than Q = 0
- Why it is possible to set ε = 0 when Q is initialized to 3 and still learn a good policy
- Why setting ε = 0.1 hurts SARSA's performance more than Q-learning's

- Why neither algorithm learns to get the big reward when Q is initialized to 0

It may help to watch the agent and visualize the value function under these conditions. Also consider doing a few steps of the algorithms by hand to see what is going on!

As you are writing keep the following in mind:

- You may assume that your boss and the engineers are familiar with the problem and the basic algorithms, so you don't need to spend a lot of time describing them in detail. However, remember that your report should be self-contained enough to still be useful later on if a team-member is hired or someone wants to refer back to the issue. Also, you may want to refer to aspects of the problem or the algorithms in order to explain the observations.
- Make sure you describe both how you generated the results *and* the results themselves, as shown in your graphs. Explain what the graphs show and point out the key features to pay attention to.
- Your boss is a busy lady! Be professional, clear, and to the point. Make sure she knows early on where you are going with this and why it matters or she might just stop reading.
- Remember, you are writing for engineers – they need you to be precise, thorough, and logical to trust that your explanation is correct. Vague language or murky arguments will not be received well.
- Don't be afraid to draw diagrams and work through examples if it helps get your point across.

Make Recommendations

The exploration strategy the engineers stumbled upon of setting the initial Q-values to a high number has a name: *optimistic initialization.* It is often coupled with setting $\varepsilon = 0$. In your report you should also give some practical guidance based on your findings regarding this strategy. In particular you should produce an example MDP that demonstrates that optimistic initialization with $\varepsilon = 0$ is not always better than ε-greedy exploration with pessimistic initialization. Design an MDP in which optimistic initialization with $\varepsilon = 0$ causes the agent to take *far* longer to learn a near-optimal policy than pessimistic initialization with $\varepsilon > 0$.

Note: the goal of this example is to convince the engineers that optimistic initialization is not always a good idea. For instance, it won't help that goal if you do something absurd that no one would ever do, like set the initial value to 1,000,000 times the true largest value. To that end:

- The optimistic value *must be attainable*, so there must be some reachable state-action pair that actually has the initial value.
- Your pessimistic initialization should truly be no greater than the lowest possible value (otherwise it's still optimistic initialization!).

- Your example should be as simple as possible so it clearly illustrates the principle and doesn't leave your reader scratching their head about a lot of unnecessary details.

You should describe your MDP and your experimental methodology and report the results that illustrate and explain the difference between these two strategies in your environment. Based on your findings, comment on when optimistic initialization is likely to be a benefit or an impediment.

3. Linear Value Function Approximation (5 pts)

Of course robots usually have to deal with more continuous values – position, velocity, angle, torque, that kind of thing. And that means value function approximation. The engineering team is a little bit intimidated by the prospect, so it's up to you to get the ball rolling.

In *rl.py*, complete the implementation of the `LinearSarsaLearner` class. It has a similar structure to `SarsaLearner`. The main difference is that it should use linear value function approximation with binary features. Rather than a state, the methods take a list of feature indices, indicating which binary features are "on" (have a value of 1). They use these indices to determine which weights to add together.

You can test your agent in *mountaincar.py*, which applies SARSA with value function approximation to the Mountain Car problem, described in Sutton and Barto (Chapter 8.4). It uses a tile coding as described by Sutton and Barto, implemented in *tilefeatures.py*. By default it uses 5 9x9 tilings, but these parameters can be set on the command line.

4. Step Size Report (5 pts)

One thing the engineers have trouble with is picking α, the step size. They feel like they don't know where to begin. In fact, it is pretty common practice to just try several different values for α and see which one works best. This is usually called a "parameter sweep." You should do this in Mountain Car with the default tiling and $\varepsilon = 0$ to find a good value of α. Some notes:

- Make sure to run multiple trials of each parameter setting to reduce noise.
- I recommend starting with a few candidate values over a wide range, seeing what happens, and then narrowing in on a more promising region (kind of a binary search strategy).
- You should gather results with enough values of α to make a compelling case for the value you ultimately settle on. It is not enough to just say "We picked *x*, and it seemed to work!"

Now what happens when you increase the number of tilings to 10 (using the -n 10 option)? Similarly find a good setting of α in this case.

In *rl.pdf* write a brief report that presents your results (plots or tables are probably a good idea). It should use them to explain to the confused engineers how to pick a good step size. It should use the Mountain Car example to…

- Recommend and illustrate a methodology for picking a good step size, making sure to address any ambiguity in what the meaning "good" is,
- Explain and illustrate the symptoms of having a step size that is too big or too small (so the engineers can notice when they have that problem),
- Explain and illustrate how the number of active features per step relates to the best step size, and
- Actually recommend a step size in the Mountain Car example for both 5 tilings and 10 tilings using these principles.

All of the above guidelines apply: make sure your report is polished and professional, clear and to the point, and self-contained enough to be useful later on.