Seamus Holland and Michael Quint

HW 2

For this assignment, which required both a Model-Free and Model-Based reinforcement learning approach, we decided to implement Q-Learning and a version of Richard S. Sutton's "Dyna-Q", respectively.

Q-learning revolves around the following formula:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}^{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

The "Q-value" defined above represents the utility value of each action for a given state, considering the long-term implications of those actions. Because Q-learning is Model-Free, it does not know a set of transition probability values, nor a set of reward values for each state. Rather, it *experiences* those values in the real world, and using its experience and the equation above, generates an optimal policy. This policy takes the form of a "Q-table", which relates a utility value to every possible action at a given state.

As a model-free technique, Q-learning has the benefit of increased access to the real world, which it uses to gain experience. Such a technique is quite effective for our problem, which has a relatively small problem space. In the *actual* real world, problem spaces are often quite massive, and approaches like Q-learning can become quite inefficient.

To tackle more complex problems, therefore, it is often more efficient to use a Model-based approach. Model-Based techniques use a limited amount of real world experiences to generate a model for both transition probabilities and possible rewards. Most of the work then takes place within this model, which ideally excludes many of the unnecessary paths that cloud the real-world problem space.

Our particular Model-Based approach, Dyna-Q, takes the Q-learning algorithm and augments it. For each real-world experience, Dyna-Q interacts with its model several hundreds of times, each time updating its Q-table of utility values. This allowed us to interact with the real-world much less frequently while still approximating well the optimal policy.

Setting the learning rate to a higher value will give more weight to new experiences in the algorithm, while having a lower learning rate value will make give less weight to new experience. This means that a low learning rate will take a lot of new information to change the policy while with a high learning rate the values can be easily over-ridden.

We decided to stop learning when our model values/utility values seemed to be converging at specific values instead of changing. This meant that we were coming to an optimal policy based on Q/ Dyna-Q learning instead of not yet fleshed out numbers. Additional runs after the point of convergence seemed to only change the numbers by small amounts.

Making the discount value smaller meant more weight was given to how good the current action set was instead of giving more weight to the future reward. The discount value directly scaled the worth of future reward, so multiplying it by a smaller value would make the future reward smaller.

The larger the epsilon the more our algorithm explores since if we choose a random value between 0 and 1 and if it is less then epsilon we take a random action. Since not meeting this criteria means that a greedy approach is taken (we choose the best action), the lower epsiolion the less we explore and the more we just take the best path.