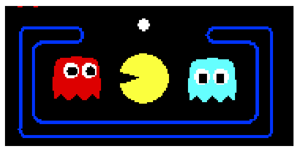
CS188 Fall 2017 Section 6: RL

1 Pacman with Feature-Based Q-Learning

We would like to use a Q-learning agent for Pacman, but the state size for a large grid is too massive to hold in memory. To solve this, we will switch to feature-based representation of Pacman's state.

1. Say our two minimal features are the number of ghosts within 1 step of Pacman (F_g) and the number of food pellets within 1 step of Pacman (F_p). You'll notice that these features depend only on the state, not the actions you take. Keep that in mind as you answer the next couple of questions. For this pacman board:



   Extract the two features (calculate their values).

2. With Q Learning, we train off of a few episodes, so our weights begin to take on values. Right now $w_g = 100$ and $w_p = -10$. Calculate the Q value for the state above.

3. We receive an episode, so now we need to update our values. An episode consists of a start state s, an action a, an end state s', and a reward r. The start state of the episode is the state above (where you already calculated the feature values and the expected Q value). The next state has feature values $F_g = 0$ and $F_p = 2$ and the reward is 50. Assuming a discount of $\gamma = 0.5$, calculate the new estimate of the Q value for s based on this episode.

4. With this new estimate and a learning rate (α) of 0.5, update the weights for each feature.

2 Odds and Ends

1. Can all MDPs be solved using expectimax search? Justify your answer.

2. When using features to represent the Q-function is it guaranteed that the feature-based Q-learning finds the same optimal $Q*$ as would be found when using a tabular representation for the Q-function?

3. Why might Q-learning be superior to TD learning of values?

4. When performing Q-learning with ϵ-greedy action selection, is it a good idea to decrease ϵ to 0 with time? Why or why not? Remember that ϵ is the (small) probability that you choose a *random* action, and $1 - \epsilon$ is the (large) probability you act on your current policy.