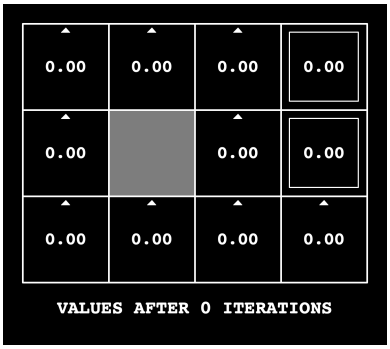
Assignment #2 Report

Problem 1

1. **Without changing the discount, noise, or livingReward (i.e., don't specify those parameters in the pattern above), screenshot the values and q-values of value iteration with -i = 0, 1, 2.**

i = 0	i = 1	i = 2
		

2. **Explain why it took until -i=1 for the terminal rewards to appear.**
   There were no iterations so it would have never found out there was a terminal state at that tile.

3. **Explain why, for the i=2 case, the q-values $Q(s,a)$ for the space directly left of the fiery pit (the -1 tile) are non-zero, but the value of this tile $V2(s)$=0. You may want to draw part of the expectimax tree as part of your answer.**
   Going based on the value iteration status update equation, we are multiplying the learning rate by the previous state reward which we saw was 0 so that part of the equation is 0. We are then adding that to the reward we got for entering the state which is also 0 since there is no living reward. Finally, we are multiplying that sum by the probability, and that product would be 0 (probability * [0+0]).

4. **Run the simulation with parameters python gridworld.py -a value -i 100 --livingReward 0.01 --discount 1 and take a screenshot of the result for your report.**

5. **Run the simulation with parameters python gridworld.py -a value -i 100 --livingReward 0.01 --discount 0.9 and take a screenshot of the result for your report.**

6. **Explain why the policies are different between the simulations in items 4 and 5 above.**
   Since the discount factor is simply 1 in question 4, the future reward does not get discounted and thus stays at the same value at each iteration, as opposed to in question 5, where there is a discount factor of 0.9, it will impact the future rewards accordingly, resulting in lower values at tiles.

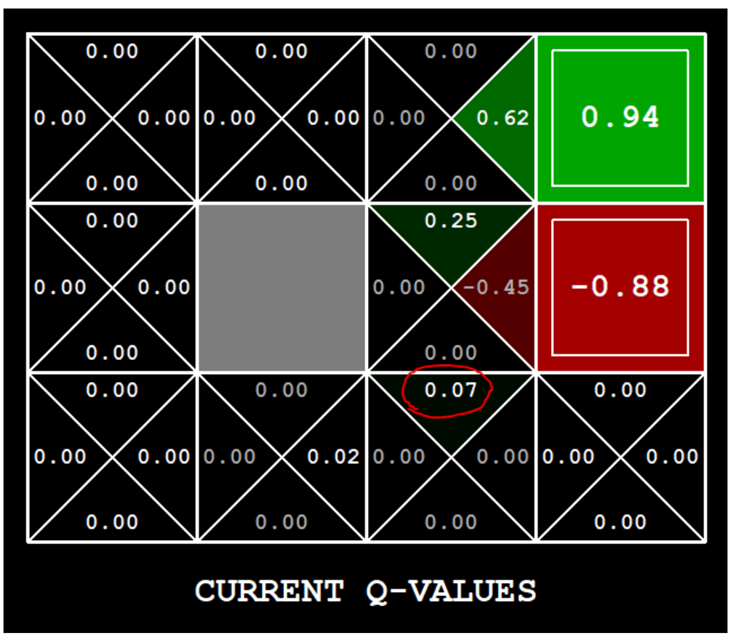
Problem 2

1. **In a sentence each, define the purpose of each of the following and what effects each of the following have on the agent's policy:**
   a. **Terminal Rewards**
      These rewards provide feedback when an agent reaches a state where no further action can be taken. This is to help encourage the agent to reach terminal states that help them maximize their cumulative reward and avoid those that would result in a catastrophic loss.
   b. **Living Rewards**
      The purpose of this type of reward is to encourage the agent to make progress in its environment (by reaching a terminal state). The effect it has on the agent's policy depends on the value of the reward. If the reward exceeds the "good" positive terminal reward, the agent will never terminate an episode because life is good. If the reward is lower than the "bad" negative terminal reward, then the agent will seek to end its existence as fast as possible since its life is pain.
   c. **Transition Noise**
      The purpose of this is to encourage exploration since it can lead the agent to explore new areas it hasn't seen anything and also to mimic real-world situations where the agent may encounter unfamiliar situations (i.e. a self-driving car needing to detour from the default route due to construction). The effect this has on our agent's policy is that it can get lucky and find a state that helps max its cumulative reward by accident.
   d. **Discount Factor: This aims** to devalue future rewards to help solve the explore vs. exploit problem. The effect this has on the agent's policy depends on the value of the discount factor. In general, a small discount factor will lead to a greedier agent who highly values any type of close reward since the future rewards are being discounted heavily, and a larger discount reward will lead to a more patient agent who will take a plan for future rewards.

2. **If the livingReward = -1, noise = 0.8, would it be possible to find a value of the discount factor such that the agent's policy (starting at the yellow square above) prefers ONLY the distant exit (+10)? If so, what value and why? If not, why?**
   This isn't possible since the noise is too high. Even when the future reward is not discounted at all, the agent will just prefer going to the closer, less valuable terminal reward (we even set the discount factor to be 20 and the agent still said no).

Problem 3

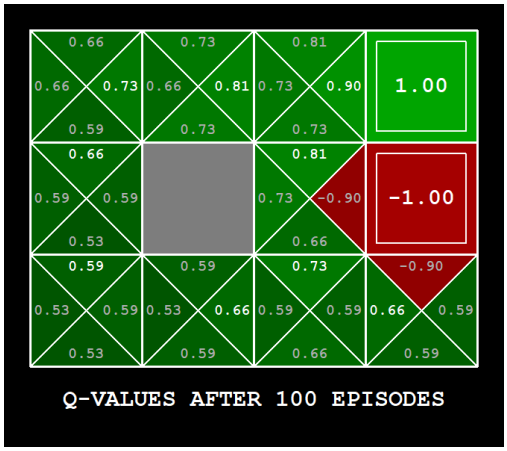
1. **Consider the following Q-values of a Q-learning agent mid-learning process, who has only tread the bottom path to each of the terminals a handful of times. Explain why the circled value of 0.07 is positive, even though the next state it leads to (next to the pit) has experientially encountered a larger negative q-value (-0.45) than positive (0.25.)**



   Since the agent has only tread the bottom path, it is quite naïve. As of right now, it doesn't know that there is an alternative, safer path to get to the good terminal reward. It only knows that if it goes North from the circled state it has a chance of getting the good reward even though it's risking (and has experienced) the bad terminal reward in the past.

Problem 4

1. **With all transition noise turned off, observe the effects of two different values of epsilon:**

Epsilon = 0.1	Epsilon = 0.9
	

Comparing these simulations answer the following questions:

a. **Which learns the optimal policy more quickly and why?**
   Epsilon = 0.1 as there is only a 10% chance it explores rather than exploiting what it knows is the best policy.

b. **Which learns more q-values and why?**
   Epsilon = 0.9 because it is curious and wants to explore the tiles it hasn't yet thus reaching punishing terminal states but then learning and finding Q-values that represent that.

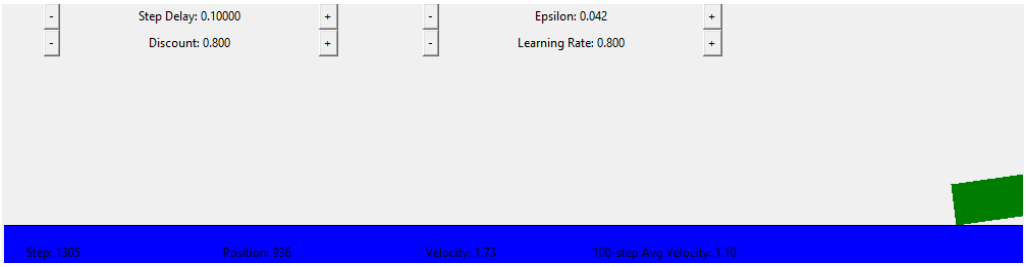
c. **Which could be described as "more reckless" in its exploration and why?**
   Epsilon = 0.9 because the exploration does not decrease over time even if it has learned Q-values early on in its iterations.

d. **Using your answer to the above, when would you want to use a higher exploration rate and when would you want a lower?**
   Higher exploration rates would be more suited towards tasks with very rewarding terminal states that the agent would not see without exploring the area. Lower exploration rates would be more suited when we want the agent to reach a specific terminal state that is within its reach, similar to the one used in the example.

e. **Why also would this algorithm be reckless/unethical to use in real-world scenarios rather than in simulations first before deploying to real-world settings?**

Online, we do not necessarily know where the terminal states are, and so deciding an arbitrary epsilon value for exploration could be highly detrimental to the specific task you're undertaking in a real-world scenario.

2. **The Crawler**
    a. **Briefly describe the following pieces of the MDP that would reuse your Q-learning code for the crawler environment: (1) state, (2) actions, (3) rewards.**
    State: The position of the crawler including how its arms are situated.
    Actions: How it can pivot and adjust its arms to move the green box as a whole.
    Rewards: The velocity it gains from the actions it chooses and the state it ends in.
    b. **Let ole Crawley explore for a bit with a high value of epsilon. Around what trial does that sucker start scooting pretty reliably if you turn epsilon down low? Take a screenshot of Crawley reaching the right side of the panel for his photo finish.**
    After having had epsilon at 0.889, then brought down to 0.042 at trial 500, it starts scooting forward slowly



Problem 5

1. **Observe a few early training rounds of Pacman learning on the smallGrid environment. Explain why Pacman doesn't appear to move a lot, either away from the ghosts or towards the pellets, in these early training samples.**
    Because Exact Q-learning requires Pacman to make random exploratory moves so he can learn the exact values of each position and action, he needs more time to train. As the specifications for this problem state, it takes between 1000 and 1400 games before Pacman starts to do well and win on the small grid. Since for this test, we set the training number to 10, that's not enough for Pacman to get better.
2. **Let's kick things up a notch and pivot to the mediumGrid environment, in which we'll let Pacman learn via the same number trials as the smallGrid.**

Explain Pacman's behavior here and why he was not able to win in this new environment with 2000 training rounds.
Because we have increased the size of our grid up to medium, Pacman will need a lot more training data to be able to start winning. We would need to increase the training number so Pacman can spend more time exploring and getting more information about the values of positions and locations.

3. **Incrementing your training by 1000 rounds each time (i.e., trying 3000 then 4000 then 5000 etc.) determine how much training is needed with exact Q-learning before Pacman consistently wins on the mediumGrid. (hint: you'll need to change the -x and -n parameters above and MAY want to do a binary search to find the answer -- it's quite large!)**
When we set the training value to 30,000, we saw that Pacman started winning consistently and got 10/10 wins.

Problem 6

1. **Explain how your agent is able to learn the optimal policy so much more quickly on the mediumGrid compared to the exact Q learner from the previous problem:**
Since approximate Q-learning uses approximation, Pacman can use these values to generalize across states. However, in Exact Q-learning, Pacman needs to explore every single state to find out the exact q value associated which takes a lot longer than approximation.
2. **Note the implementation of the IdentityExtractor that returns the current state-action pair as its own feature with value of 1 (and therefore, all other state-action pairs would be features with value of 0 in the feature vector, by default). Explain how this makes an Approximate Q Learner with these features precisely the same as an Exact Q Learner:**
The difference between Approximate Q-Learning and Exact Q-Learning is that there are feature representations that target specific things to learn from. There are weights associated with these features that get updated over time. Since all of the features are already set to 1 in the feature dictionary that gets returned by the Identity Extractor, they never provide any extra information and so we're just utilizing a "feature" to represent (state, action) pairs, which we were already doing in exact q-learning.

3. **Note also how the SimpleExtractor feature extractor is implemented and answer the questions that follow:**
    a. **Note the conditional addition of the food feature starting with the comment # if there is no danger of ghosts then add the food feature. Knowing that getting eaten by ghosts carries a large penalty, what**

would happen if this condition were removed, i.e., that the food feature is added whether or not ghosts are nearby?
The reward of the food would possibly overtake the punishment of being eaten by a ghost.

b. **Note also the scaling of distance-based features to be less than one in the logic starting with the comment # make the distance a number less than one otherwise the update will diverge wildly. Ask ChatGPT to explain why it's preferable to keep feature values between [0, 1] in approximate Q learning and interpret its answer.**
According to ChatGPT, there are several reasons why keeping feature values between [0,1] is preferable. The first reason is for numerical stability and this helps to to keep numbers within a reasonable range when approximating Q-Values. It also helps to avoid divergence since it prevents the feature values from getting too large and ensures the q-value updates remain well-behaved. It can also help the agent effectively learn and generalize better.