COMPSCI 687 Homework 3 - Fall 2024

Due **November 3**, 11:55pm Eastern Time

SHREYA BIRTHARE

1 Instructions

This homework assignment consists of a written portion and a programming portion. While you may discuss problems with your peers (e.g., to discuss high-level approaches), you must answer the questions on your own. In your submission, do explicitly list all students with whom you discussed this assignment. Submissions must be typed (handwritten and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code. You **must** use Python 3 for your homework code. You may not use any reinforcement learning or machine learning specific libraries in your code, e.g., TensorFlow, PyTorch, or scikit-learn. You *may* use libraries like numpy and matplotlib, though. The automated system will not accept assignments after 11:55pm on November 3. The tex file for this homework can be found here.

Part One: Written (50 Points Total)

In all questions below, unless stated otherwise:

- Always show your work step by step, explaining your reasoning in detail and how it supports your answer or argument.
- Simplify all expressions where possible.
- If any part of a question is unclear (e.g., what is being asked, or the type of discussion required), please consult the TAs.
- For all questions—especially open-ended ones—clearly state your assumptions (e.g., "I interpret this question as X or Y. I find X more reasonable due to reasons A, B, and C, so I will proceed with that assumption").

1. (**8 Points**) An operator f is a contraction mapping if there exists some $\lambda \in [0,1)$ such that $d\big(f(x), f(y)\big) \leq \lambda d(x,y)$, where d is a distance metric. Notice, here, that we explicitly forbid that $\lambda = 1$; otherwise, we would have no guarantees that the distance between $f(x)$ and $f(y)$ is smaller than the distance between x and y. Assume, alternatively, that we try to "fix" this problem by defining f to be a contraction mapping if there exists some $\lambda \in [0,1]$ such that $d\big(f(x), f(y)\big) < \lambda d(x,y)$. Notice that we now allow $\lambda = 1$, but we require $d\big(f(x), f(y)\big)$ to be strictly less than $\lambda d(x,y)$. What is the problem with this alternative definition of contraction mapping? What could go wrong if we were to use it?

   In the original definition of contraction mapping we say that if there exists some $\lambda \in [0,1)$ such that $d\big(f(x), f(y)\big) \leq \lambda d(x,y)$, where d is a distance metric, then f is a contraction mapping. This is because by using this definition we are able to say that the above function contracts and eventually as part of banache fixed point theorem, it yields us a unique fixed point. This strict inequality ensures that each iteration pulls points closer by a factor less than 1, so they get exponentially closer with repeated applications of f. This exponential shrinking enforces convergence to a single, unique fixed point, no matter the starting points. However as given in the question if we try to assume that f is a contraction mapping if there exists some $\lambda \in [0,1]$ such that $d\big(f(x), f(y)\big) < \lambda d(x,y)$ then this definition still has a problem. Say when $\lambda = 1$, then the definition becomes: $d\big(f(x), f(y)\big) < d(x, y)$. This does not guarantee a unique fixed point. This scenario weakens the contraction condition. The mapping may fail to converge to a single fixed point. It may be the case that the f may not need to bring points significantly closer together with each application. For instance, it could reduce the distance by an arbitrarily small amount. This leads to very slow convergence or, in some cases, no convergence at all. When $\lambda = 1$, even if points are getting closer by a tiny amount with each application of f. there's no guarantee they will converge to a single fixed point. In fact, f could have multiple fixed points under these conditions because the lack of a strong contraction factor allows points to hover around various stable positions instead of moving toward a unique solution.

2. (**22 Points Total**) In class, we studied different ways in which we may want to quantify the expected return of a policy. In some cases, we are interested in the expected return if the agent starts in state s and follows a given policy π; that is, $v^\pi(s)$. In other cases, we are interested in the expected return if the agent starts in state s, executes action a, and then follows a given policy π; that is, $q^\pi(s, a)$. Consider yet another possible way in which we may want to quantify expected return: via a function $m^\pi(s, a, s')$ that represents the expected return if the agent starts in state s, executes action a, transitions to a particular next state s', and then follows policy π.

   (a) [**10 Points**] Use the definitions and properties of probability distributions discussed in Homework 1, as well as the Markov Property (when appropriate), to derive from "first principles" a Bellman Equation for m^π using only $m^\pi, \mathcal{S}, \mathcal{A}, p, R, d_0, \gamma$, and π (but <u>not</u> v^π or q^π). Your Bellman Equation for m^π should resemble the form of the first Bellman Equation for v^π we studied in class: $v^\pi(s) = \sum_a \pi(s, a) \sum_{s'} p(s, a, s') \big(R(s, a) + \gamma v^\pi(s')\big)$. In particular, your Bellman Equation for m^π should have m^π on both sides of the equation, but it should not depend on q^π or v^π. Show your step-by-step derivation of this new Bellman Equation.

$$m^\pi(s,a,s') = \mathbb{E}[G_t | S_s = s, A_t = a, S_{t+1} = s', \pi]$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_s = s, A_t = a, S_{t+1} = s', \pi\right]$$

(We are using reward function as $R(s,a)$ throughout)

$$= \mathbb{E}\left[R_t + \sum_{k=1}^{\infty} \gamma^k R_{t+k} | S_s = s, A_t = a, S_{t+1} = s', \pi\right]$$

$$= \mathbb{E}\left[R_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} | S_s = s, A_t = a, S_{t+1} = s', \pi\right]$$

$$= \mathbb{E}\left[R_t + \gamma \sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} | S_s = s, A_t = a, S_{t+1} = s', \pi\right]$$

$$= \mathbb{E}\left[R_t | S_s = s, A_t = a, S_{t+1} = s', \pi\right] + \gamma * \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} | S_s = s, A_t = a, S_{t+1} = s', \pi\right]$$

$$= R(s,a) + \gamma * \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} | S_s = s, A_t = a, S_{t+1} = s', \pi\right] \tag{1}$$

($R(s,a)$ does not depend on S_{t+1})

Now, solving $\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_s = s, A_t = a, S_{t+1} = s', \pi\right]$:

$$\begin{aligned}
&=\mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_{t+1} = s', \pi\right] \\
&= \sum_{a'\in\mathcal{A}} \Pr(A_{t+1} = a'|S_{t+1} = s') * \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_{t+1} = s', A_{t+1} = a'\pi\right] \\
&= \sum_{a'\in\mathcal{A}} \pi(s', a') * \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_{t+1} = s', A_{t+1} = a', \pi\right] \\
&= \sum_{a'\in\mathcal{A}} \pi(s', a') \sum_{s''\in\mathcal{S}} Pr(S_{t+2} = s''|S_{t+1} = s', A_{t+1} = a') * \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_{t+1} = s', A_{t+1} = a', S_{t+2} = s'', \pi\right] \\
&= \sum_{a'\in\mathcal{A}} \pi(s', a') \sum_{s''\in\mathcal{S}} p(s', a', s'') * \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{(t+1)+k}|S_{t+1} = s', A_{t+1} = a', S_{t+2} = s'', \pi\right] \\
&= \sum_{a'\in\mathcal{A}} \pi(s', a') \sum_{s''\in\mathcal{S}} p(s', a', s'') * m^\pi(s', a', s'') \qquad (2)
\end{aligned}$$

Substituting equation 2 in equation 1:

$$\begin{aligned}
m^\pi(s, a, s') =& R(s, a) + \gamma * \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_s = s, A_t = a, S_{t+1} = s', \pi\right] \\
m^\pi(s, a, s') =& R(s, a) + \gamma * \sum_{a'\in\mathcal{A}} \pi(s', a') \sum_{s''\in\mathcal{S}} p(s', a', s'') * m^\pi(s', a', s'') \qquad (3)
\end{aligned}$$

(b) **[12 Points]** Recall that we showed, in class, how the Bellman Optimality Equation for v^*,

$$v^*(s) = \max_a \sum_{s'} p(s, a, s') \left(R(s, a) \;+\; \gamma v^*(s')\right),$$

could be turned into an *update equation*, used by the Value Iteration algorithm, to iteratively estimate v^*:

$$v_{i+1}(s) := \max_a \sum_{s'} p(s, a, s') \left(R(s, a) \;+\; \gamma v_i(s')\right).$$

To show that this update equation converges to v^*, we showed that the operator that implements it, $\mathcal{T}(v_i) := \max_a \sum_{s'} p(s, a, s') \left(R(s, a) \,+\, \gamma v_i(s')\right)$, is a contraction mapping. Similarly, consider now the Bellman Equation for m^π that you derived in the previous question. Turn it into an update equation that takes as input a current estimate, m_i, of m^π, and returns an updated estimate, m_{i+1}. Let $\Psi(m_i)$ be the operator that implements this update equation; that is, $\Psi(m_i) := m_{i+1}$. Prove, step-by-step, that Ψ is a contraction mapping under the L^∞ norm (i.e., the same Max-Norm used in our proof that the Bellman Operator is a contraction).

In this question, the following identities/properties might be helpful:

⋆ **Property 1**: The Max-Norm over m^π is $||m^\pi||_\infty = \max_s \max_a \max_{s'} |m^\pi(s,a,s')|$.

⋆ **Property 2**:

$$\Big|\sum_{x\in\mathcal{X}} \Pr(X{=}x|Y{=}y, Z{=}z, \ldots) f(x,y,z,\ldots)\Big| \leq \sum_{x\in\mathcal{X}} \Pr(X{=}x|Y{=}y, Z{=}z, \ldots)\Big|f(x,y,z,\ldots)\Big|,$$

where $X, Y, Z, \ldots$, are random variables with possible outcomes (respectively) in $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$, and where f is a function.

⋆ **Property 3**:

$$\sum_{x\in\mathcal{X}} \Pr(X{=}x|Y{=}y, Z{=}z, \ldots) f(x,y,z,\ldots) \leq \max_{x\in\mathcal{X}} f(x,y,z,\ldots),$$

where $X, Y, Z, \ldots$, are random variables with possible outcomes (respectively) in $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$, and where f is a function.

We will first convert m^π to its updated equation and then show ψ is a contraction mapping. Note that we have used reward function as $R(s,a)$ throughout question 2.

Equation 3 from Q2(a) gave us:

$$m^\pi(s,a,s') = R(s,a) + \gamma * \sum_{a'\in\mathcal{A}} \pi(s',a') \sum_{s''\in\mathcal{S}} p(s',a',s'') * m^\pi(s',a',s'')$$

Now we know using the concept of dynamic programming and policy evaluation that we can estimate a policy at state i using the above formula to get a new estimate at state $i+1$. And this is done until m_i converges to m^π eventually.
So we can say,

$$m_{i+1}(s,a,s') = R(s,a) + \gamma * \sum_{a'\in\mathcal{A}} \pi(s',a') \sum_{s''\in\mathcal{S}} p(s',a',s'') * m_i(s',a',s'') \tag{4}$$

Let π^* be an optimal policy, then by bellman optimality for m^{π^*}, we can rewrite equation 3 as:

$$m^{\pi^*}(s,a,s') = R(s,a) + \gamma * \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'') * m^{\pi^*}(s',a',s'')$$

(Since we want to maximize the expected reward, we choose the action a' that gives the maximum expected value)

$$m^{\pi^*}(s,a,s') = \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'') \Big(R(s,a) + \gamma * m^{\pi^*}(s',a',s'')\Big) \tag{5}$$

(as $\sum_{s''\in\mathcal{S}} p(s',a',s'') = 1$)

And policy evaluation is nothing but selecting the max deterministic policy π greedily that results in the optimal results. So π^* is nothing but:

$$\pi^* = \arg\max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'') \Big(R(s,a) + \gamma * m^{\pi^*}(s',a',s'')\Big) \tag{6}$$

Value iteration is one step policy evaluation and 1 step policy improvement to estimate v^*. So, similarly to estimate m^* we can rewrite equation 4, with the help of equation 5 and equation 6 as:

$$m_{i+1}(s,a,s') = \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(R(s,a) + \gamma * m_i(s',a',s'')\right) \tag{7}$$

Equation 7 is our final update equation for m to estimate m^{π^*} .

Now that we have found the update equation for m, we proceed with the second part, i.e prooving that ψ is a contraction mapping.

Given that:

$$\psi(m_i) = m_{i+1}$$
$$||m^\pi||_\infty = \max_s \max_a \max_{s'} |m^\pi(s,a,s')|$$

Let m and m' be two m^π functions and $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. So,

$$||\psi(m(s,a,s')) - \psi(m'(s,a,s'))||_\infty = \max_s \max_a \max_{s'} |\psi(m(s,a,s')) - \psi(m'(s,a,s'))| \tag{8}$$

Solving $\max_s \max_a \max_{s'} |\psi(m) - \psi(m')|$ now:

$$= \max_s \max_a \max_{s'} |\psi(m) - \psi(m')|$$

$$= \max_s \max_a \max_{s'} \left| \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(R(s,a) + \gamma * m(s',a',s'')\right) - \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(R(s,a) + \gamma m'(s',a',s'')\right)\right|$$

$$\leq \max_s \max_a \max_{s'} \max_{a'\in\mathcal{A}} \left| \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(R(s,a) + \gamma * m(s',a',s'')\right) - \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(R(s,a) + \gamma m'(s',a',s'')\right)\right|$$

(known property as discussed in class)

$$\leq \gamma * \max_s \max_a \max_{s'} \max_{a'\in\mathcal{A}} \left| \sum_{s''\in\mathcal{S}} p(s',a',s'')\left(m(s',a',s'') - m'(s',a',s'')\right)\right|$$

$$\left(\textit{Taking constant } \gamma \textit{ out and cancelling out } \sum_{s''\in\mathcal{S}} p(s',a',s'')R(s,a)\right)$$

$$\leq \gamma * \max_s \max_a \max_{s'} \max_{a'\in\mathcal{A}} \sum_{s''\in\mathcal{S}} p(s',a',s'') \Big| (m(s',a',s'') - m'(s',a',s''))\Big|$$

(Using Property 2)

$$\leq \gamma * \max_s \max_a \max_{s'} \max_{a'\in\mathcal{A}} \max_{s''\in\mathcal{S}} \Big| (m(s',a',s'') - m'(s',a',s''))\Big|$$

(Using Property 3)

$$\leq \gamma * \max_{s'\in\mathcal{S}} \max_{a'\in\mathcal{A}} \max_{s''\in\mathcal{S}} \Big| (m(s',a',s'') - m'(s',a',s''))\Big|$$

($\max_s, \max_a$ *don't affect computation here)*

$$\leq \gamma * ||m(s',a',s'') - m'(s',a',s'')||_\infty \tag{9}$$

Substituting equation 9 in equation 8 gives:

$$||\psi(m(s,a,s')) - \psi(m'(s,a,s'))||_\infty \leq \gamma * ||m(s',a',s'') - m'(s',a',s'')||_\infty \tag{10}$$

Equation 10, proves that ψ is a contraction mapping.

3. **(8 Points)** The First-Visit Monte Carlo algorithm uses an estimator, $\hat{v}(s)$, of the value of a state s, constructed by taking the average of all returns G_i^s, where G_i^s is the return observed from the first occurrence of state s in the i-th trajectory. In class, we showed that First-Visit Monte Carlo is unbiased. However, Monte Carlo methods are known for having high variance. For instance, suppose we collect k trajectories and compute the First-Visit Monte Carlo estimator, $\hat{v}_k^{\text{ori}}(s)$, for $v^\pi(s)$. If we repeat this process with a different set of k trajectories, yielding a new estimator $\hat{v}_k^{\text{new}}(s)$, it is possible that $\hat{v}_k^{\text{new}}(s)$ could differ significantly from $\hat{v}_k^{\text{ori}}(s)$, even though both are unbiased. This variability between runs highlights the high variance of Monte Carlo estimates. Write down an equation for the variance of this estimator. How does the variance depend on k? Given that this estimator is unbiased, what does the expression for its variance tell you about the "accuracy" of First-Visit estimators as k (the number of trajectories) increases?

   The First-Visit Monte Carlo Algorithm yields to an unbiased result as we know. This is because for a trajectory i, we take the first occurrence of it only and these i's are independent and identically distributed variables or trajectories. Results of a trajectory dont affect the results of other trajectory. Let $v_k(s) = \frac{1}{k}\sum_{i=1}^{k} G_i$ where G_i is the i^{th} return for a state s.
   $\mathbb{E}[G_i] = v^\pi(s)$ and G_i are i.i.d because they are sampled from independent trajectories. So by law of large numbers we know that $v_k(s)$ uniformly converges to $v^\pi(s)$.
   Now as we said that these are i.i.d trajectories, we can express the variance as:

$$\begin{aligned}
Var(v_k(s)) =& Var\left(\frac{1}{k}\sum_{i=1}^{k} G_i\right)\\
=& \frac{1}{k^2} Var\left(\sum_{i=1}^{k} G_i\right)\\
& \textit{(known property for i.i.d variables for variance, constant is squared out)}\\
=& \frac{1}{k^2} k Var(G_i)\\
& \textit{(Summing over k)}\\
=& \frac{1}{k} Var(G_i) \qquad (11)
\end{aligned}$$

   Equation 11 shows that the variance of the trajectories is inversely proportional to k. This means that on increasing k, i.e. running more trajectories, the variance decreases and that is what we want also. This indicates better accuracy as variance decreases. This also intuitively implies that as we increase k, variance decreases and we get the expected results closer to the true mean.

4. **(12 points)** The First-Visit Monte Carlo algorithm estimates the value of a state s using an estimator, $\hat{v}(s)$, which is the average of all corresponding returns G_i^s, where G_i^s is the return observed after the first occurrence of state s in the i-th trajectory. In class, we showed that First-Visit Monte Carlo is unbiased.

   Now, consider a variant of this algorithm called *Performance-Improving Monte Carlo Estimation* (PIMCE). Unlike the original algorithm, PIMCE does not use returns from all trajectories when estimating the value of s. Instead, it only takes into account returns G_i^s that are at least as good as (and hopefully higher than) the return from the previous iteration. That is, PIMCE only includes returns where $G_i^s \geq G_{i-1}^s$. For simplicity, assume the first return, G_0^s, is always included. The motivation behind PIMCE is to encourage the reinforcement learning agent to prioritize higher returns, rather than considering all returns, including poorly performing ones. Assume you are given a function IncreasedPerformance(i) defined as:

$$\text{IncreasedPerformance}(i) = \begin{cases} 1 & \text{if } i = 0 \text{ or } G_i^s \geq G_{i-1}^s, \\ 0 & \text{otherwise.} \end{cases}$$

The Performance-Improving Monte Carlo estimator of the value of some state s is defined as:

$$\hat{v}_{\text{PIMCE}}(s) = \frac{1}{\sum_{i=1}^{k} \text{IncreasedPerformance}(i)} \sum_{i=1}^{k} \left(\text{IncreasedPerformance}(i)\, G_i^s\right).$$

Is Performance-Improving Monte Carlo unbiased? If so, prove it. If not, formally argue/explain why.

In the First-Visit Monte Carlo algorithm, the value of a state s is estimated by averaging all returns observed after the first occurrence of state s in trajectory i. The unbiasedness of the First-Visit Monte Carlo estimator follows from the fact that, as the trajectories are i.i.d variables. Results of a trajectory dont affect the results of other trajectory. We can say that $v_k(s) = \frac{1}{k}\sum_{i=1}^{k} G_i$ where G_i is the i^{th} return for a state s. The expected return for each trajectory is equal to the true value function v(s) of state s. As we increase k, we get the expected results closer to the true mean. This is because we are taking all the values of G_i for our computation here.
In the case of PIMCE estimator, we see that only those returns that are greater than or equal to the return from the previous iteration, are included in the average. This is formalized by the indicator function IncreasedPerformance(i) function. Essentially, its value function is the average of the returns that are considered as improved. For an estimator to be unbiased, the expected value of the estimator must equal the true value of the state. In the case of PIMCE estimator, it only uses a subset of the observed returns, specifically those that satisfy the condition mentioned in the question. This introduces potential bias because the returns that are not included in the sum could be systematically lower than the true value of $\hat{v}(s)$. The returns used in the PIMCE estimator are not randomly selected; they are biased towards higher returns (those greater than or equal to the previous one). As a result, the PIMCE estimator may overestimate the true value $\hat{v}(s)$, since it is systematically excluding lower returns, which could be closer to the true average return for the state. The fact that PIMCE selectively includes only returns that meet the improvement criterion means that the selected returns are not representative of the entire distribution of returns from state s. In particular, this could skew the estimator upward, as the returns with lower values are excluded, leading to an overestimation of $\hat{v}(s)$. Since the inclusion of a return depends on the comparison to the previous return, the estimator's behavior is inherently dependent on the trajectory of previous returns. This further complicates the expectation of the estimator, as it is no longer averaging over all possible returns but only over a subset, which is not guaranteed to be representative of the full distribution of returns. Given the selective inclusion of returns based on their value relative to previous returns, the PIMCE Estimator introduces a form of bias. Specifically, it tends to overestimate the true value of state s by excluding lower returns that might more accurately reflect the true expected return of the state. Therefore, PIMCE is not an unbiased estimator of $\hat{v}(s)$.

Part Two: Programming (50 Points Total)

In this question, you will implement the Value Iteration algorithm and use it to find the optimal value function and the optimal policy for a domain called the Cat-vs-Monsters domain. **Notice that you may not use existing RL code for this problem—you must implement the learning algorithm and the environment entirely on your own and from scratch.** The domain you should implement is described below:

- **States**: This problem consists of a 5×5 environment where each state $s = (r, c)$ describes the current coordinates/location of a cat. In particular, $r \in [0, 4]$ is the current row where the cat is located, and $c \in [0, 4]$ is the current column where the cat is located. Refer to Figure 1 for an example. In this figure, the topmost row is row zero, and the leftmost column is column zero—i.e., *State1* corresponds to $s = (0, 0)$ and *State14* corresponds to $s = (3, 1)$.
- **Actions**: There are four actions: AttemptUp (AU), AttemptDown (AD), AttemptLeft (AL), and AttemptRight (AR).
- **Dynamics**: This is a *stochastic* MDP:
  - With 70% probability, the cat moves in the specified direction.
  - With 12% probability, the cat gets confused and moves to the right with respect to the intended direction.
  - With 12% probability, the cat gets confused and moves to the left with respect to the intended direction.
  - With 6% probability, the cat gets sleepy and decides not to move.
  - The environment is surrounded by walls. If the cat hits a wall, it gets scared and does not move.
  - There are four *Forbidden Furniture* locations in this environment: one in $(2, 1)$, one in $(2, 2)$, one in $(2, 3)$, and one $(3, 2)$. If the cat touches a Forbidden Furniture, it gets paralyzed and remains in its current state. The cat cannot go on the furniture.
  - There are two *Monsters*: one in $(0, 3)$ and one in $(4, 1)$.
  - There is a *Food state* located at $(4, 4)$.
- **Rewards**: The reward is always -0.05, except when transitioning to (entering) the Food state, in which case the reward is 10; or when transitioning to (entering) a state containing a Monster, in which case the reward is -8. Notice that to model this type of reward function, you will need to use a reward function in the form $R(S_t, A_t, S_{t+1})$ instead of $R(S_t, A_t)$. This requires a small modification to the Value Iteration update equation: $v_{i+1}(s) := \max_{a \in \mathcal{A}} \sum_{s'} p(s, a, s')\Big(R\big(s, a, s'\big) + \gamma v_i(s')\Big)$.
- **Terminal State**: The Food state is terminal. Any actions executed in this state always transition to s_∞ with reward 0.
- **Discount**: Unless otherwise specified, $\gamma = 0.925$.
- **Initial State**: The cat deterministically wakes up at the beginning of each episode on its bed; that is, $S_0 = (0, 0)$, always.

Questions (Programming)

General instructions:

- You should initialize the value of all states to zero.
- Whenever displaying values (e.g., the value of a state), use 4 decimal places; e.g., $v(s) = 9.4312$.

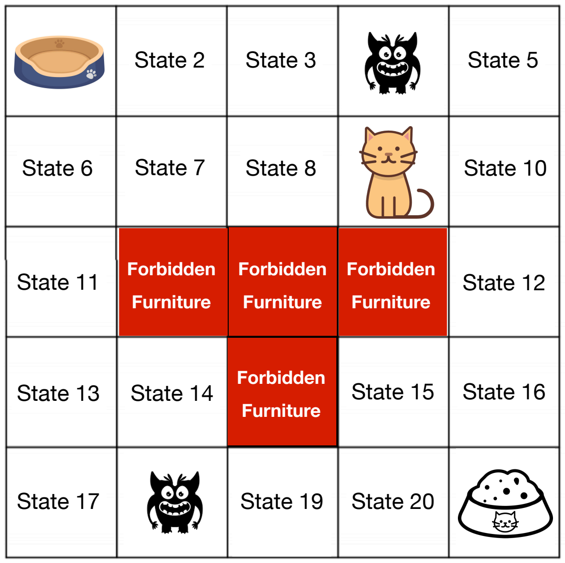
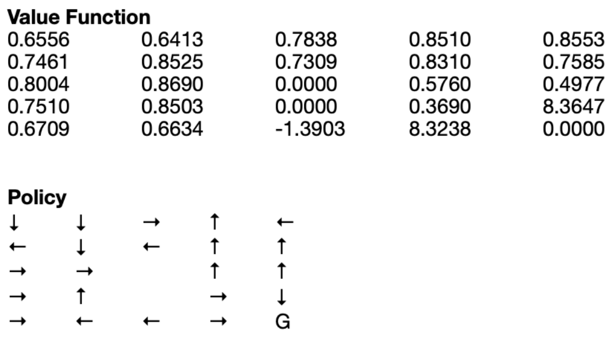


Figure 1: The Cat-vs-Monsters domain.

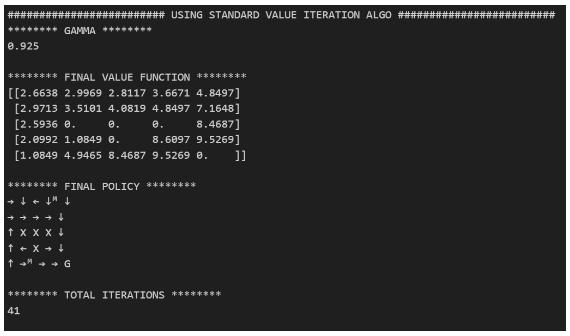
- In the following questions, the Value Iteration algorithm should stop whenever $\Delta < 0.0001$.
- Whenever showing the value function and policy learned by your algorithm, present them in a format resembling that depicted in Figure 2.



```
Value Function
0.6556      0.6413      0.7838      0.8510      0.8553
0.7461      0.8525      0.7309      0.8310      0.7585
0.8004      0.8690      0.0000      0.5760      0.4977
0.7510      0.8503      0.0000      0.3690      8.3647
0.6709      0.6634      -1.3903     8.3238      0.0000

Policy
↓    ↓    →    ↑    ←
←    ↓    ←    ↑    ↑
→    →         ↑    ↑
→    ↑         →    ↓
→    ←    ←    →    G
```

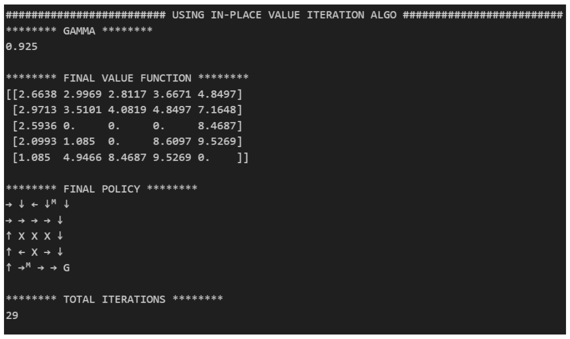
Figure 2: Example of the output format your code should generate. Note that the values provided are solely for illustrative purposes to demonstrate the format for presenting your experimental results, and may not reflect the actual state values for a given policy.

1. (**14 Points**) Run the Value Iteration algorithm in the Cat-vs-Monsters domain as defined above until its stopping criterion is met. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* How many iterations did it take for the algorithm to stop? *(c)* Describe, in English, the behavior being implemented by the learned policy.



```
######################## USING STANDARD VALUE ITERATION ALGO ########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[2.6638 2.9969 2.8117 3.6671 4.8497]
 [2.9713 3.5101 4.0819 4.8497 7.1648]
 [2.5936 0.     0.     0.     8.4687]
 [2.0992 1.0849 0.     8.6097 9.5269]
 [1.0849 4.9465 8.4687 9.5269 0.    ]]

******** FINAL POLICY ********
→ ↓ ← ↓M ↓
→ → → → ↓
↑ X X X ↓
↑ ← X → ↓
↑ →M → → G

******** TOTAL ITERATIONS ********
41
```

Figure 3: Output when running standard value iteration for Q1



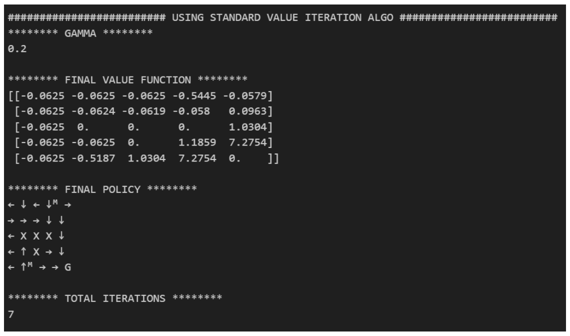
```
######################## USING IN-PLACE VALUE ITERATION ALGO ########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[2.6638 2.9969 2.8117 3.6671 4.8497]
 [2.9713 3.5101 4.0819 4.8497 7.1648]
 [2.5936 0.     0.     0.     8.4687]
 [2.0993 1.085  0.     8.6097 9.5269]
 [1.085  4.9466 8.4687 9.5269 0.    ]]

******** FINAL POLICY ********
→ ↓ ← ↓M ↓
→ → → → ↓
↑ X X X ↓
↑ ← X → ↓
↑ →M → → G

******** TOTAL ITERATIONS ********
29
```

Figure 4: Output when running in-place value iteration for Q1

In Figure 3, we see that it took 41 iterations for the value function to converge to yield an optimal policy using the standard value iteration algorithm. The final value functions for the food state (final state) and forbidden furniture states are zero. This is obvious as we shouldnt be running the algo starting from those states. The learned policy directs the agent through a grid towards the goal, marked by 'G' in the bottom right corner, using an efficient and strategic path. Guided by directional arrows, it avoids going in the direction of forbidden furniture states (goes away from it) and tries to also avoid the monster state by going to other states that are not forbidden/monster state. For example at $(4,0)$ index, it tries to move up instead of right to avoid the monster state at $(4,1)$. Similarly at state $(3,1)$ it goes left to $(3,0)$ as it cant go to $(3,2)$ as its forbidden furniture and avoids going down to $(4,1)$ as its monster state. So in general it is taking the bath that is yielding in best rewards avoiding penalties and unreachable states.

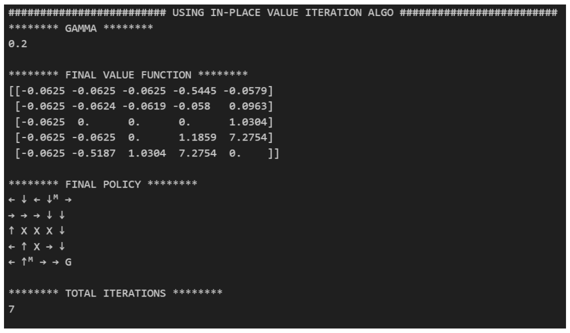
Figure 4 shows similar final value function and policy as figure 3, only difference is in the number of iterations. Here it takes 29 iterations for it to converge to yield optimal policy. This is because we have implemented the in place variant of the value iteration algo which updates the value function for each state immediately, using the latest available values from neighboring states rather than waiting until the end of an iteration. Regarding describing the policy, its the same logic and reasoning as how we described it for the standard algo implementation.

2. (**10 Points**) Run the same experiment as above, but now using $\gamma = 0.2$. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* Did the value function identified when $\gamma = 0.2$ change with respect to the value function identified when $\gamma = 0.925$? How about the policy? If any of these quantities changed, explain, intuitively, why you think that happened. *(c)* Finally, describe how many iterations it took for the algorithm to stop. Did it take more iterations or fewer iterations, compared to when $\gamma = 0.925$? Explain, intuitively, why you think that is the case.



```
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.2

******** FINAL VALUE FUNCTION ********
[[-0.0625 -0.0625 -0.0625 -0.5445 -0.0579]
 [-0.0625 -0.0624 -0.0619 -0.058   0.0963]
 [-0.0625  0.      0.      0.      1.0304]
 [-0.0625 -0.0625  0.      1.1859  7.2754]
 [-0.0625 -0.5187  1.0304  7.2754  0.    ]]

******** FINAL POLICY ********
← ↓ ← ↓M →
→ → → ↓ ↓
← X X X ↓
← ↑ X → ↓
← ↑M → → G

******** TOTAL ITERATIONS ********
7
```

Figure 5: Output when running standard value iteration for Q2



```
######################### USING IN-PLACE VALUE ITERATION ALGO #########################
******** GAMMA ********
0.2

******** FINAL VALUE FUNCTION ********
[[-0.0625 -0.0625 -0.0625 -0.5445 -0.0579]
 [-0.0625 -0.0624 -0.0619 -0.058   0.0963]
 [-0.0625  0.      0.      0.      1.0304]
 [-0.0625 -0.0625  0.      1.1859  7.2754]
 [-0.0625 -0.5187  1.0304  7.2754  0.    ]]

******** FINAL POLICY ********
← ↓ ← ↓M →
→ → → ↓ ↓
↑ X X X ↓
← ↑ X → ↓
← ↑M → → G

******** TOTAL ITERATIONS ********
7
```

Figure 6: Output when running in-place value iteration for Q2

In Figure 5, we see that with $\gamma = 0.2$ it took 7 iterations to converge. This is much less than with $\gamma = 0.925$ which took 29 iterations to converge using the standard algo.

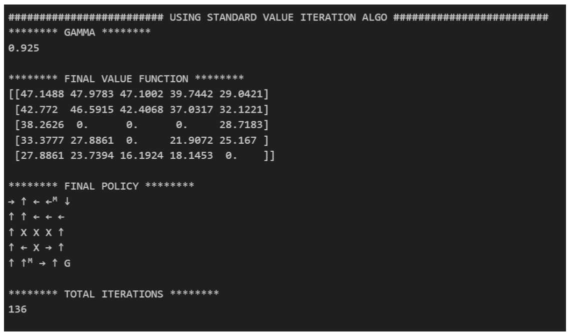
Talking about the part (b) first. When $\gamma = 0.2$, we see that the values for goal and forbidden state are still zero as expected but the values in the rest of the grid are much smaller overall, with most values around zero or slightly negative, EXCEPT near the goal states. The maximum values are concentrated close to the goal. Positive values are restricted closer to the goal, as the impact of future rewards quickly diminishes. The policy focuses on reaching the goal in as few steps as possible, as future rewards are heavily discounted. Thus, the learned policy in states far from the goal may direct movement towards terminal states or quick rewards, disregarding longer paths with higher cumulative rewards.
Whereas when $\gamma = 0.925$, the values were larger throughout the grid. States further from the goal have higher values due to the larger discount factor, which allows future rewards to have a more significant influence on the value function. Positive values spread further across the grid, as rewards in the distant future still contribute substantially to the value at each state. The policy is more patient here and willing to navigate through paths with higher cumulative rewards, even if it requires more steps. The policy reflects a long-term perspective, focusing on maximizing the sum of rewards over a longer period.

Now regarding part (c), $\gamma = 0.2$ took 7 iterations to converge while $\gamma = 0.925$ took 29 iterations to converge using the standard algo. The algorithm converged faster with the lower discount factor ecause the impact of future rewards is much more limited. With a low γ, the values of states are primarily influenced by immediate rewards, and the algorithm quickly settles on values close to their final estimates. This reduced reliance on distant rewards leads to quicker stabilization of the value function, as updates propagate only a short distance in each iteration.
In contrast, a higher discount factor places significant weight on future rewards. As a result, changes in value estimates propagate more extensively through the grid, taking many iterations to reach a stable state where the long-term future rewards are fully accounted for. This explains why it took more

iterations for convergence with a higher γ.

In Figure 6, we see that with $\gamma = 0.2$ it took 7 iterations to converge with the in-place algo implementation. This is the same number of iterations it took for the standard algo to converge. The value function and policy are the same as standard ones as expected. The reasoning for part (b) and (c) with the in-place algo is the same as for the standar algo. We get similar number of ietartions, this means that very low γ (0.2), reduced the influence of future rewards, making convergence relatively quick and localized for both methods. This result suggests that the benefits of in-place updates are more prominent when γ is higher.

3. (**10 Points**) In this question, you will once again use the Value Iteration algorithm with $\gamma = 0.925$. You will now create a state in which the cat finds catnip. To do so, change the reward given to the cat when entering the state $s = (0, 1)$. Whenever that happens, the reward should be 5.0. This should *not* be a terminal state. *(a)* Show the final value function and the final policy identified by this algorithm. *(b)* Describe, in English, the behavior being implemented by the learned policy, and explain intuitively why this is the best policy for this variant of the domain.

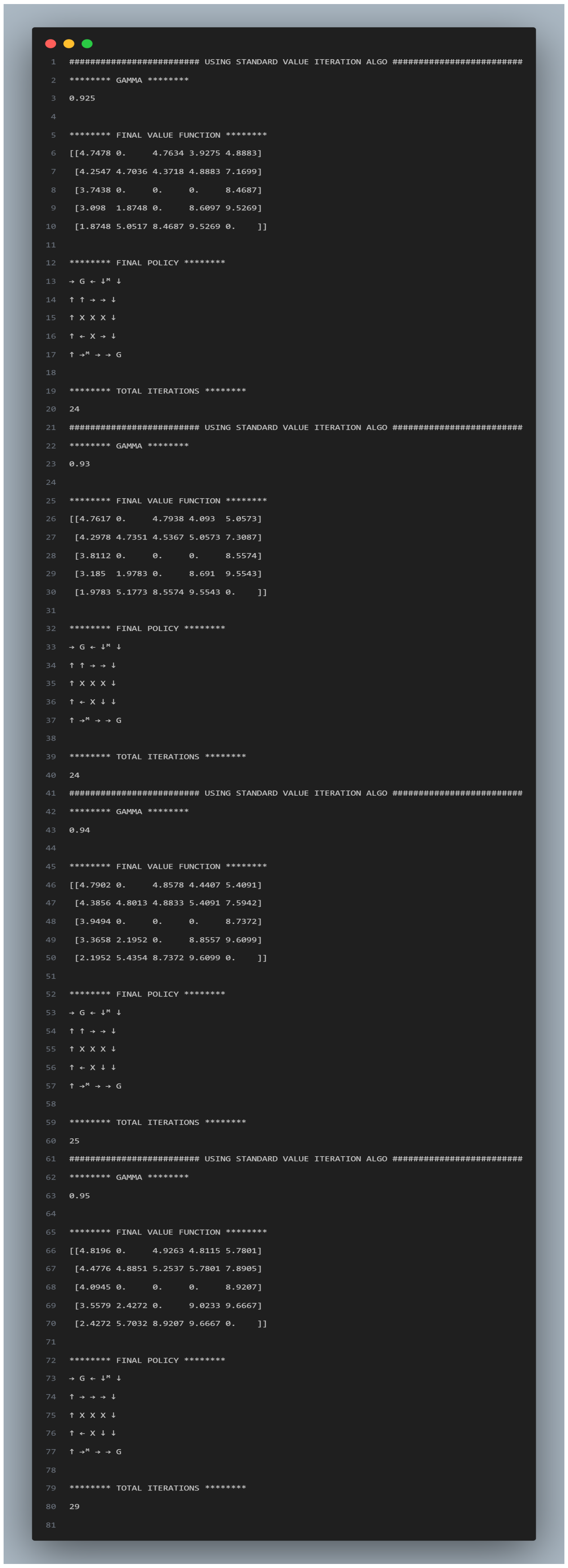


```
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[47.1488 47.9783 47.1002 39.7442 29.0421]
 [42.772  46.5915 42.4068 37.0317 32.1221]
 [38.2626  0.      0.      0.     28.7183]
 [33.3777 27.8861  0.     21.9072 25.167 ]
 [27.8861 23.7394 16.1924 18.1453  0.    ]]

******** FINAL POLICY ********
→ ↑ ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
136
```

Figure 7: Output when running standard value iteration for Q3

Figure 7 shows the results when we use the standard algo and make state $(0, 1)$ give higher reward (5) then other states except the food state. We see that it took 136 iterations to converge.
The learned policy suggests that the cat should move toward the state with the catnip ($s = (0, 1)$) as part of the optimal path. The cat will first move towards the state $(0, 1)$, where it can gain a high reward of 5, as this is the newly assigned reward. After reaching that state, the policy leads the cat to explore neighboring states while considering the future rewards and the influence of the high reward at $(0, 1)$. It avoids unnecessary states that lead to zero or lower rewards. The cat's movement towards (0, 1) is prioritized, given the substantial reward that state offers. Intuitively, this is the best policy because the high reward at the catnip state (5.0) creates a strong incentive to move toward it. In a domain with a high discount factor ($\gamma = 0.925$), future rewards are still significant, so the cat is motivated to navigate toward the state where the immediate reward (5) is most advantageous, while also considering how this impacts future steps. The policy optimally balances both short-term and long-term rewards, ensuring that the cat takes the most rewarding path given the setup.

4. (**8 Points**) Repeat the experiment above, but now make the catnip state, $s = (0, 1)$, a terminal state, similar to the Food state. Any action taken in this state always leads to s_∞ with a reward of 0. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* Describe, in English, the behavior being implemented by the learned policy, and explain intuitively why this is the best policy for this variant of the domain. *(c)* Finally, experiment with larger values of γ. What is the first larger value of γ that causes the optimal policy to change—e.g., that causes the cat to avoid the new rewarding state? Describe intuitively why changing γ in this way results in such a different behavior.



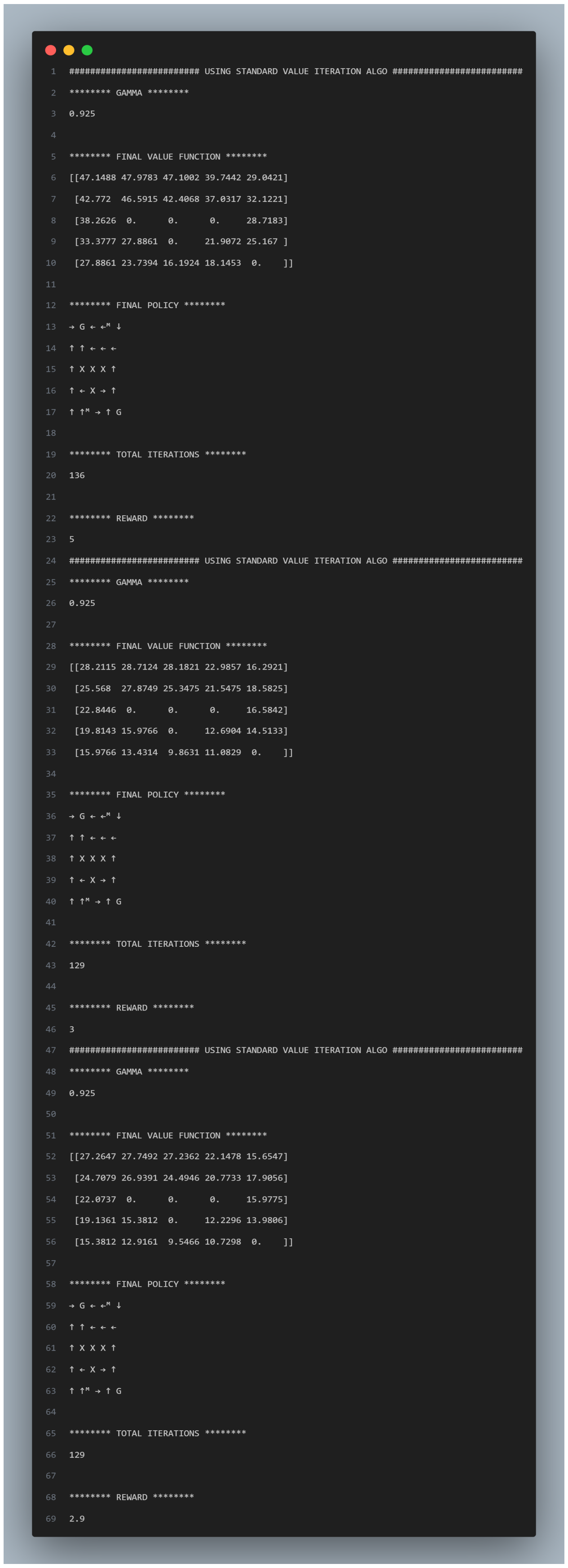
```
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[4.7478 0.     4.7634 3.9275 4.8883]
 [4.2547 4.7036 4.3718 4.8883 7.1699]
 [3.7438 0.     0.     0.     8.4687]
 [3.098  1.8748 0.     8.6097 9.5269]
 [1.8748 5.0517 8.4687 9.5269 0.    ]]

******** FINAL POLICY ********
→ G ← ↓ᴹ ↓
↑ ↑ → → ↓
↑ X X X ↓
↑ ← X → ↓
↑ →ᴹ → → G

******** TOTAL ITERATIONS ********
24
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.93

******** FINAL VALUE FUNCTION ********
[[4.7617 0.     4.7938 4.093  5.0573]
 [4.2978 4.7351 4.5367 5.0573 7.3087]
 [3.8112 0.     0.     0.     8.5574]
 [3.185  1.9783 0.     8.691  9.5543]
 [1.9783 5.1773 8.5574 9.5543 0.    ]]

******** FINAL POLICY ********
→ G ← ↓ᴹ ↓
↑ ↑ → → ↓
↑ X X X ↓
↑ ← X ↓ ↓
↑ →ᴹ → → G

******** TOTAL ITERATIONS ********
24
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.94

******** FINAL VALUE FUNCTION ********
[[4.7902 0.     4.8578 4.4407 5.4091]
 [4.3856 4.8013 4.8833 5.4091 7.5942]
 [3.9494 0.     0.     0.     8.7372]
 [3.3658 2.1952 0.     8.8557 9.6099]
 [2.1952 5.4354 8.7372 9.6099 0.    ]]

******** FINAL POLICY ********
→ G ← ↓ᴹ ↓
↑ ↑ → → ↓
↑ X X X ↓
↑ ← X ↓ ↓
↑ →ᴹ → → G

******** TOTAL ITERATIONS ********
25
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.95

******** FINAL VALUE FUNCTION ********
[[4.8196 0.     4.9263 4.8115 5.7801]
 [4.4776 4.8851 5.2537 5.7801 7.8905]
 [4.0945 0.     0.     0.     8.9207]
 [3.5579 2.4272 0.     9.0233 9.6667]
 [2.4272 5.7032 8.9207 9.6667 0.    ]]

******** FINAL POLICY ********
→ G ← ↓ᴹ ↓
↑ → → → ↓
↑ X X X ↓
↑ ← X ↓ ↓
↑ →ᴹ → → G

******** TOTAL ITERATIONS ********
29

```

Figure 8: Output when running standard value iteration for Q4 for different γs

Figure 8 shows the results when we apply the standard algo on different values of γ.
When $\gamma = 0.925$, in this case where the catnip state ($s = (0, 1)$) is made a terminal state, the policy suggests a strategy where the cat avoids reaching the catnip state ($s = (0, 1)$) since it leads to the terminal state with no further reward. The cat's movement is directed away from the catnip state, instead focusing on paths that lead to more rewarding states, particularly those near the goal (G) especially the food state. The learned policy reflects this avoidance by steering the cat away from the catnip state and instead guiding it toward states that can provide higher rewards before reaching the goal. The value function shows higher values near the goal (G), indicating the highest rewards the cat can achieve by reaching the goal. The catnip state, now a terminal state, has a value of 0, reflecting the absence of further rewards once entered. States near the goal or catnip have moderate values, guiding the cat towards optimal paths. Zero values in isolated states represent dead zones, with no available rewards, emphasizing the importance of avoiding these areas.

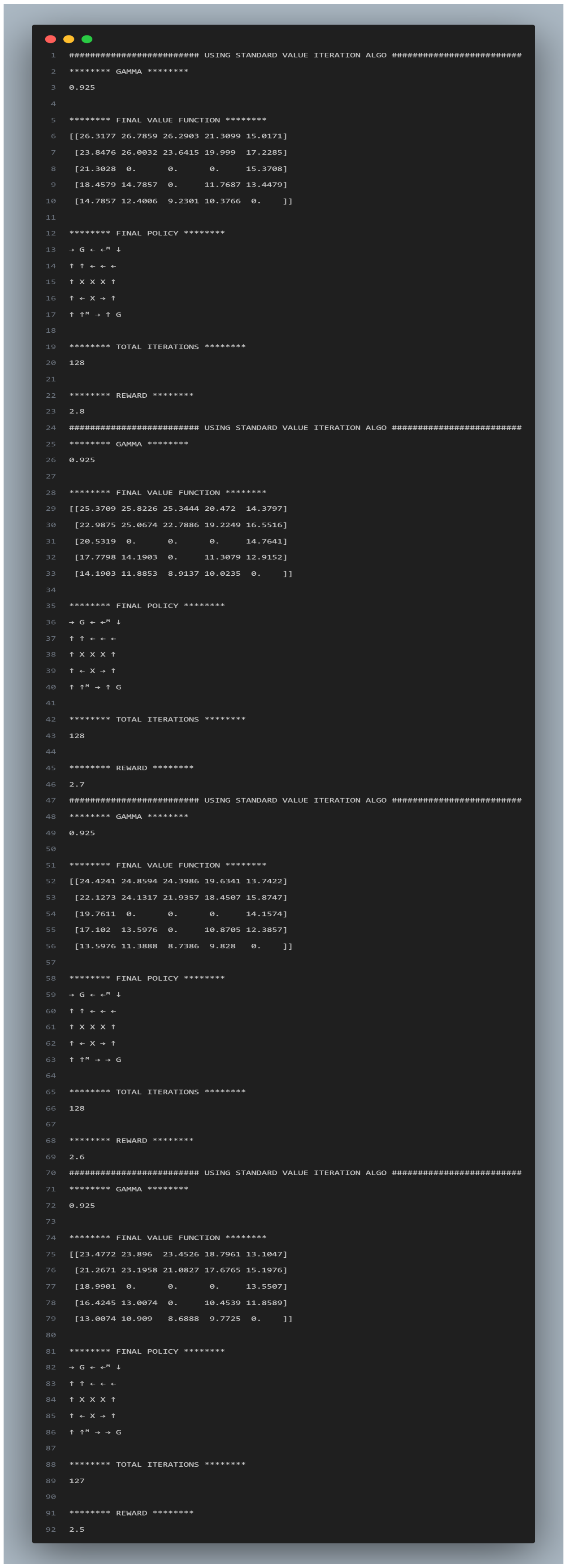
On experimenting with larger values of γ $(0.925, 0.93, 0.94, 0.95)$, the first larger value of γ that causes the optimal policy to change is $\gamma = 0.95$. At this point, the cat begins to prioritize reaching the food state (G) over reaching the catnip state, even though it provides a reward of 5. This happens because the higher γ increases the importance of future rewards, making the long-term goal more valuable compared to the immediate reward at the catnip state. The cat now seeks to maximize its cumulative future rewards, moving towards the food state more aggressively and avoiding the catnip state despite its immediate reward.

5. (**8 Points**) Repeat the experiment above (using $\gamma = 0.925$), but now experiment with different rewards for reaching the catnip at $s = (0, 1)$—i.e., rewards other than 5.0. *(a)* What is the first (lower) reward value that causes the optimal policy to change, for example, causing the cat to avoid the new rewarding state? Explain intuitively why changing the reward in this way leads to this different behavior. *(b)* Show the final value function and the final policy identified by the algorithm.



```
######################## USING STANDARD VALUE ITERATION ALGO ########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[47.1488 47.9783 47.1002 39.7442 29.0421]
 [42.772  46.5915 42.4068 37.0317 32.1221]
 [38.2626  0.       0.       0.      28.7183]
 [33.3777 27.8861  0.      21.9072 25.167 ]
 [27.8861 23.7394 16.1924 18.1453  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
136

******** REWARD ********
5
######################## USING STANDARD VALUE ITERATION ALGO ########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[28.2115 28.7124 28.1821 22.9857 16.2921]
 [25.568  27.8749 25.3475 21.5475 18.5825]
 [22.8446  0.       0.       0.      16.5842]
 [19.8143 15.9766  0.      12.6904 14.5133]
 [15.9766 13.4314  9.8631 11.0829  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
129

******** REWARD ********
3
######################## USING STANDARD VALUE ITERATION ALGO ########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[27.2647 27.7492 27.2362 22.1478 15.6547]
 [24.7079 26.9391 24.4946 20.7733 17.9056]
 [22.0737  0.       0.       0.      15.9775]
 [19.1361 15.3812  0.      12.2296 13.9806]
 [15.3812 12.9161  9.5466 10.7298  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
129

******** REWARD ********
2.9
```

Figure 9: Output when running standard value iteration for Q5 for different rewards



```
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[26.3177 26.7859 26.2903 21.3099 15.0171]
 [23.8476 26.0032 23.6415 19.999  17.2285]
 [21.3028  0.       0.       0.      15.3708]
 [18.4579 14.7857  0.      11.7687 13.4479]
 [14.7857 12.4006  9.2301 10.3766  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
128

******** REWARD ********
2.8
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[25.3709 25.8226 25.3444 20.472  14.3797]
 [22.9875 25.0674 22.7886 19.2249 16.5516]
 [20.5319  0.       0.       0.      14.7641]
 [17.7798 14.1903  0.      11.3079 12.9152]
 [14.1903 11.8853  8.9137 10.0235  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → ↑ G

******** TOTAL ITERATIONS ********
128

******** REWARD ********
2.7
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[24.4241 24.8594 24.3986 19.6341 13.7422]
 [22.1273 24.1317 21.9357 18.4507 15.8747]
 [19.7611  0.       0.       0.      14.1574]
 [17.102  13.5976  0.      10.8705 12.3857]
 [13.5976 11.3888  8.7386  9.828   0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → → G

******** TOTAL ITERATIONS ********
128

******** REWARD ********
2.6
######################### USING STANDARD VALUE ITERATION ALGO #########################
******** GAMMA ********
0.925

******** FINAL VALUE FUNCTION ********
[[23.4772 23.896  23.4526 18.7961 13.1047]
 [21.2671 23.1958 21.0827 17.6765 15.1976]
 [18.9901  0.       0.       0.      13.5507]
 [16.4245 13.0074  0.      10.4539 11.8589]
 [13.0074 10.909   8.6888  9.7725  0.    ]]

******** FINAL POLICY ********
→ G ← ←ᴹ ↓
↑ ↑ ← ← ←
↑ X X X ↑
↑ ← X → ↑
↑ ↑ᴹ → → G

******** TOTAL ITERATIONS ********
127

******** REWARD ********
2.5
```

Figure 10: Output when running standard value iteration for Q5 for different rewards

Figures 9 and 10 show the results for different rewards on the catnip state ($(0, 1)$) when run with the standard implementation of the in-place value iteration algorithm. We see that the first lower reward that causes the optimal policy to change is when reward$=2.6$ Below this threshold, the cat starts avoiding the catnip state ($s = (0, 1)$), even though it still offers a reward. This shift in behavior occurs because the cat's policy now prioritizes reaching the food state (G) rather than the catnip state (G). As the reward for the catnip state decreases, it becomes less attractive in comparison to the long-term reward of reaching the goal. The cat begins to optimize for higher cumulative rewards, and the value of the catnip state is no longer sufficient to outweigh the benefits of continuing toward the goal, leading the cat to avoid the catnip state altogether.