# Peeking Blackjack

**Stanford CS221 Spring 2021-2022**

Owner CA: Lantao Yu

Version: 4/14/2022

Ed Release Post

## General Instructions

This (and every) assignment has a written part and a programming part.

The full assignment with our supporting code and scripts can be downloaded as blackjack.zip.

a. This icon means you should write responses in `blackjack.pdf`.

b. This icon means you should write code in `submission.py`.

All written answers must be **typeset (preferably in LaTeX)**. We strongly recommend using Overleaf. A link to a LaTeX template with prompts can be found on Ed, or further down this page.

Also note that your answers should be **in order** and **clearly and correctly labeled** to receive credit. Be sure to submit your final answers as a PDF and tag all pages correctly when submitting to Gradescope.

You should modify the code in `submission.py` between

```
# BEGIN_YOUR_CODE
```

and

```
# END_YOUR_CODE
```

but you can add other helper functions outside this block if you want. Do not make changes to files other than `submission.py`.

Your code will be evaluated on two types of test cases, **basic** and **hidden**, which you can see in `grader.py`. Basic tests, which are fully provided to you, do not stress your code with large inputs or tricky corner cases. Hidden tests are more complex and do stress your code. The inputs of hidden tests are provided in `grader.py`, but the correct outputs are not. To run the tests, you will need to have `graderUtil.py` in the same directory as your code and `grader.py`. Then, you can run all the tests by typing

```
python grader.py
```

This will tell you only whether you passed the basic tests. On the hidden tests, the script will alert you if your code takes too long or crashes, but does not say whether you got the correct output. You can also run a single test (e.g., `3a-0-basic`) by typing

```
python grader.py 3a-0-basic
```

We strongly encourage you to read and understand the test cases, create your own test cases, and not just blindly run `grader.py`.

The search algorithms explored in the previous assignment work great when you know exactly the results of your actions. Unfortunately, the real world is not so predictable. One of the key aspects of an effective AI is the ability to reason in the face of uncertainty.

Markov decision processes (MDPs) can be used to formalize uncertain situations. In this homework, you will implement algorithms to find the optimal policy in these situations. You will then formalize a modified version of Blackjack as an MDP, and apply your algorithm to find the optimal policy. Finally, you will explore using MDPs for modeling a real-world scenario, specifically rising sea levels.

## Problem 1: Value Iteration

In this problem, you will perform the value iteration updates manually on a very basic game just to solidify your intuitions about solving MDPs. The set of possible states in this game is $\mathcal{S} = \{-2, -1, 0, +1, +2\}$ and the set of possible actions is $\mathcal{A} = \{a_1, a_2\}$. The initial state is $0$ and there are two terminal states, $-2$ and $+2$. Recall that the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ encodes the probability of transitioning to a next state $s'$ after being in state $s$ and taking action $a$ as $\mathcal{T}(s'|s,a)$. In this MDP, the transition dynamics are given as follows:

$\forall i \in \{-1, 0, 1\} \subset \mathcal{S}$,

- $\mathcal{T}(i-1|i,a_1) = 0.8$ and $\mathcal{T}(i+1|i,a_1) = 0.2$
- $\mathcal{T}(i-1|i,a_2) = 0.7$ and $\mathcal{T}(i+1|i,a_2) = 0.3$

Think of this MDP as a chain formed by states $\{-2, -1, 0, +1, +2\}$. In words, action $a_1$ has a 80% chance of moving the agent backwards in the chain and a 20% chance of moving the agent forward. Similarly, action $a_2$ has a 70% of sending the agent backwards and a 30% chance of moving the agent forward. We will use a discount factor $\gamma = 1$.

The reward function for this MDP is $\mathcal{R}(s,a,s') = \begin{cases} 10 & s' = -2 \\ 50 & s' = +2 \\ -5 & \text{otherwise} \end{cases}$

a. [3 points] What is the value of $V^{(i)}_{\text{opt}}(s)$ for each state in $\mathcal{S}$ after each iteration $i = \{0, 1, 2\}$ of Value Iteration? Please write down the values from all iterations. Recall that $\forall s \in \mathcal{S}$, $V^{(0)}_{\text{opt}}(s) = 0$ and, for any terminal state $s_{\text{terminal}}$, $V_{\text{opt}}(s_{\text{terminal}}) = 0$. In other words, all values are $0$ after iteration 0 and terminate states always have a value of $0$.

**What we expect:** The $V^{(i)}_{\text{opt}}(s)$ of all 5 states after each iteration. In total, 15 values should be reported.

Iteration 0: $\forall s \in \mathcal{S}, V^\star_0(s) = 0$.
Iteration 1: $V^\star_1(-2) = V^\star_1(+2) = 0$
$V^\star_1(-1) = \max\{0.8(10+0) + 0.2(-5+0), 0.3(-5+0) + 0.7(10+0)\} = 7$ ,
$V^\star_1(0) = \max\{0.8(-5+0) + 0.2(-5+0), 0.3(-5+0) + 0.7(-5+0)\} = -5$ ,
$V^\star_1(+1) = \max\{0.8(-5+0) + 0.2(50+0), 0.3(50+0) + 0.7(-5+0)\} = 11.5$ ,

Iteration 2: $V^\star_2(-2) = V^\star_2(+2) = 0$
$V^\star_2(-1) = \max\{0.8(10+0) + 0.2(-5-5), 0.3(-5-5) + 0.7(10+0)\} = 6$ ,
$V^\star_2(0) = \max\{0.8(-5+7) + 0.2(-5+11.5), 0.3(-5+11.5) + 0.7(-5+7)\} = 3.35$ ,
$V^\star_2(+1) = \max\{0.8(-5-5) + 0.2(50+0), 0.3(50+0) + 0.7(-5-5)\} = 8$ .

b. [1 point] Using $V^{(2)}_{\text{opt}}(\cdot)$, what is the corresponding optimal policy $\pi_{\text{opt}}$ for all non-terminal states?

**What we expect:** A few state action pairs to express the optimal policy.

$\pi^\star(-1) = a_1, \pi^\star(0) = \pi^\star(+1) = a_2$

## Problem 2: Transforming MDPs

Equipped with an understanding of a basic algorithm for computing optimal value functions in MDPs, let's gain intuition about the dynamics of MDPs which either carry some special structure, or are defined with respect to a different MDP.

a. [4 points] Suppose we have an MDP with states $\text{States}$ and a discount factor $\gamma < 1$, but we have an MDP solver that can only solve MDPs with discount factor of $1$. How can we leverage the MDP solver to solve the original MDP?

Let us define a new MDP with states $\text{States}' = \text{States} \cup \{o\}$, where $o$ is a new state. Let's use the same actions ($\text{Actions}'(s) = \text{Actions}(s)$), but we need to keep the discount $\gamma' = 1$. Your job is to define new transition probabilities $\mathcal{T}'(s'|s,a)$ and rewards $\text{Reward}'(s,a,s')$ in terms of the old MDP such that the optimal values $V_{\text{opt}}(s)$ for all $s \in \text{States}$ are equal under the original MDP and the new MDP.

*Hint: If you're not sure how to approach this problem, go back to Tatsu's notes from the first MDP lecture and read closely the slides on convergence, toward the end of the deck.*

**What we expect:** A few transition probabilities and reward functions written in mathematical expressions, followed by a short proof to show that the two optimal values are equal. Try to use the same symbols as the question.

The idea is to interpret the discount $\gamma$ as the probability of not transitioning into $o$. Let $o$ be a terminal state. This admits two solutions:

1. (correct)
   - $\mathcal{T}'(s'|s,a) \doteq \gamma\mathcal{T}(s'|s,a)$ for $s' \in \text{States}$
   - $\mathcal{T}'(o|s,a) \doteq 1-\gamma$
   - $\text{Reward}'(s,a,s') \doteq \text{Reward}(s,a,s')$ for all $s' \in \text{States}$
   - $\text{Reward}'(s,a,o) \doteq \sum_{s'\in\text{States}}\mathcal{T}(s'|s,a)\text{Reward}(s,a,s')$
2. (correct)
   - $\mathcal{T}'(s'|s,a) \doteq \gamma\mathcal{T}(s'|s,a)$ for $s' \in \text{States}$
   - $\mathcal{T}'(o|s,a) \doteq 1-\gamma$
   - $\text{Reward}'(s,a,s') \doteq \frac{1}{\gamma}\text{Reward}(s,a,s')$ for all $s' \in \text{States}$
   - $\text{Reward}'(s,a,o) \doteq 0$
3. (incorrect)
   - $\mathcal{T}'(s'|s,a) \doteq \gamma\mathcal{T}(s'|s,a)$ for $s' \in \text{States}$
   - $\mathcal{T}'(o|s,a) \doteq 1-\gamma$
   - $\text{Reward}'(s,a,s') \doteq \text{Reward}(s,a,s')$ for all $s' \in \text{States}$
   - $\text{Reward}'(s,a,o) \doteq 0$

Recall the recurrence for the new optimal value:

$$V'_{\text{opt}}(s) = \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}'} \mathcal{T}'(s'|s,a)[\text{Reward'}(s,a,s') + V'_{\text{opt}}(s')].$$

$$= \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \mathcal{T}'(s'|s,a)[\text{Reward'}(s,a,s') + V'_{\text{opt}}(s')]$$

$$+\mathcal{T}'(o|s,a)[\text{Reward'}(s,a,o) + V'_{\text{opt}}(o)]$$

Plugging in the definitions of the new transitions and rewards, we get that, for each solution:

1. $$V'_{\text{opt}}(s) = \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \gamma\mathcal{T}(s'|s,a)[\text{Reward}(s,a,s') + V'_{\text{opt}}(s')] +$$
$$(1-\gamma) \sum_{s'\in\text{States}} \mathcal{T}(s'|s,a)\text{Reward}(s,a,s'),$$
$$= \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \mathcal{T}(s'|s,a)[\text{Reward}(s,a,s') + \gamma V'_{\text{opt}}(s')].$$

2. $$V'_{\text{opt}}(s) = \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \gamma\mathcal{T}(s'|s,a)[\frac{1}{\gamma}\text{Reward}(s,a,s') + V'_{\text{opt}}(s')] +$$
$$(1-\gamma) * 0$$
$$= \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \mathcal{T}(s'|s,a)[\text{Reward}(s,a,s') + \gamma V'_{\text{opt}}(s')]$$

3. $$V'_{\text{opt}}(s) = \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \gamma\mathcal{T}(s'|s,a)[\text{Reward}(s,a,s') + V'_{\text{opt}}(s')] +$$
$$(1-\gamma) * 0$$
$$= \max_{a\in\text{Actions}(s)} \sum_{s'\in\text{States}} \mathcal{T}(s'|s,a)[\gamma\text{Reward}(s,a,s') + \gamma V'_{\text{opt}}(s')]$$

In all cases except for the third, we have recovered the original recurrence:

$$V'_{\text{opt}}(s) = \max_{a \in \text{Actions}(s)} \sum_{s' \in \text{States}} \mathcal{T}(s' | s, a)[\text{Reward}(s, a, s') + \gamma V'_{\text{opt}}(s')].$$

Therefore, the new MDP and the old MDP have the same optimal values.

## Problem 3: Peeking Blackjack

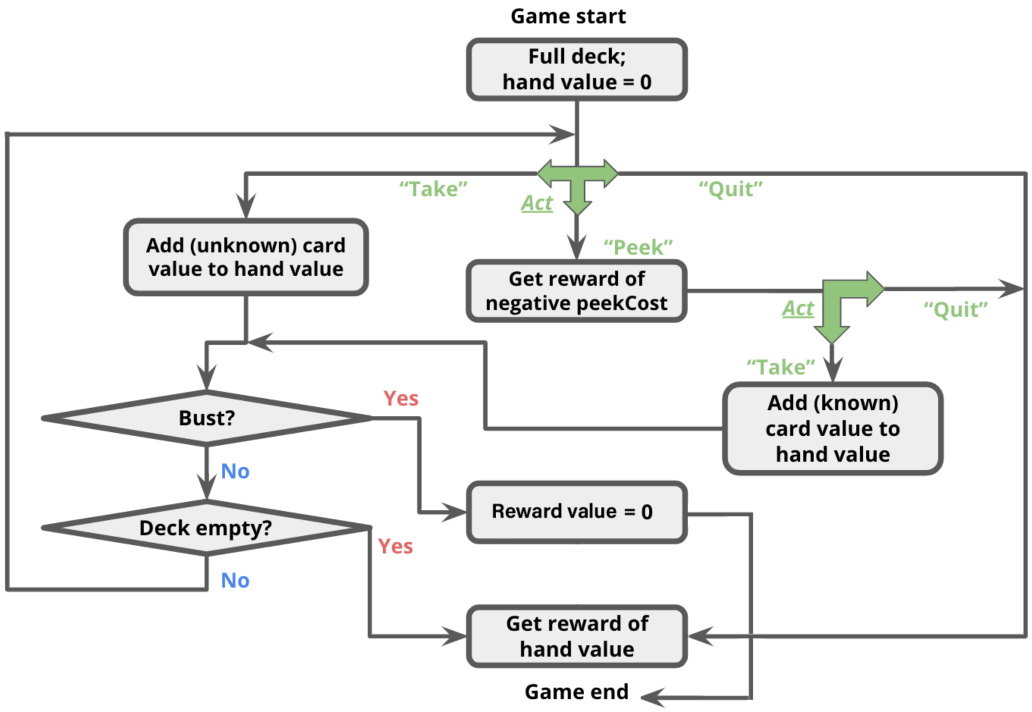
Now that we have gotten a bit of practice with general-purpose MDP algorithms, let's use them to play (a modified version of) Blackjack. For this problem, you will be creating an MDP to describe states, actions, and rewards in this game. More specifically, after reading through the description of the state representation and actions of our Blackjack game below, you will implement the transition and reward function of the Blackjack MDP inside `succAndProbReward()`.

For our version of Blackjack, the deck can contain an arbitrary collection of cards with different face values. At the start of the game, the deck contains the same number of each cards of each face value; we call this number the 'multiplicity'. For example, a standard deck of 52 cards would have face values $[1, 2, \ldots, 13]$ and multiplicity 4. You could also have a deck with face values $[1, 5, 20]$; if we used multiplicity 10 in this case, there would be 30 cards in total (10 each of 1s, 5s, and 20s). The deck is shuffled, meaning that each permutation of the cards is equally likely.

The game occurs in a sequence of rounds. In each round, the player has three actions available to her:

- $a_{\text{take}}$ - Take the next card from the top of the deck.
- $a_{\text{peek}}$ - Peek at the next card on the top of the deck.
- $a_{\text{quit}}$ - Stop taking any more cards.

In this problem, your state $s$ will be represented as a 3-element tuple:


Game start
Full deck; hand value = 0
"Take"
Act
"Quit"
"Peek"
Add (unknown) card value to hand value
Get reward of negative peekCost
Act
"Quit"
"Take"
Add (known) card value to hand value
Bust?
Yes
No
Deck empty?
Yes
No
Reward value = 0
Get reward of hand value
Game end

```
(totalCardValueInHand, nextCardIndexIfPeeked, deckCardCounts)
```

As an example, assume the deck has card values $[1, 2, 3]$ with multiplicity 2, and the threshold is 4. Initially, the player has no cards, so her total is 0; this corresponds to state `(0, None, (2, 2, 2))`.

- For $a_{\text{take}}$, the three possible successor states (each with equal probability of $1/3$) are:

```
(1, None, (1, 2, 2))
(2, None, (2, 1, 2))
(3, None, (2, 2, 1))
```

  Three successor states have equal probabilities because each face value had the same amount of cards in the deck. In other words, a random card that is available in the deck is drawn and its corresponding count in the deck is then decremented. Remember that `succAndProbReward()` will expect you return all three of the successor states shown above. Note that $\mathcal{R}(s, a_{\text{take}}, s') = 0, \forall s, s' \in \mathcal{S}$. Even though the agent now has a card in her hand for which she may receive a reward at the end of the game, the reward is not actually granted until the game ends (see termination conditions below).

- For $a_{\text{peek}}$, the three possible successor states are:

```
(0, 0, (2, 2, 2))
(0, 1, (2, 2, 2))
```

`(0, 2, (2, 2, 2))`

Note that it is not possible to peek twice in a row; if the player peeks twice in a row, then `succAndProbReward()` should return `[]`. Additionally, $\mathcal{R}(s, a_{\text{peek}}, s') = -\text{peekCost}, \forall s, s' \in \mathcal{S}$. That is, the agent will receive an immediate reward of `-peekCost` for reaching any of these states.
Things to remember about the states after taking $a_{\text{peek}}$:

- From `(0, 0, (2, 2, 2))`, taking a card will lead to the state `(1, None, (1, 2, 2))` deterministically (that is, with probability 1.0).
- The second element of the state tuple is not the face value of the card that will be drawn next, but the index into the deck (the third element of the state tuple) of the card that will be drawn next. In other words, the second element will always be between 0 and `len(deckCardCounts)-1`, inclusive.

- For $a_{\text{quit}}$, the resulting state will be `(0, None, None)`. (Remember that setting the deck to `None` signifies the end of the game.)

The game continues until one of the following termination conditions becomes true:

- The player chooses $a_{\text{quit}}$, in which case her reward is the sum of the face values of the cards in her hand.
- The player chooses $a_{\text{take}}$ and "goes bust". This means that the sum of the face values of the cards in her hand is strictly greater than the threshold specified at the start of the game. If this happens, her reward is 0.
- The deck runs out of cards, in which case it is as if she selects $a_{\text{quit}}$, and she gets a reward which is the sum of the cards in her hand. *Make sure that if you take the last card and go bust, then the reward becomes 0 not the sum of values of cards.*

As another example with our deck of $[1, 2, 3]$ and multiplicity 1, let's say the player's current state is `(3, None, (1, 1, 0))`, and the threshold remains 4.

- For $a_{\text{quit}}$, the successor state will be `(3, None, None)`.
- For $a_{\text{take}}$, the successor states are `(3 + 1, None, (0, 1, 0))` or `(3 + 2, None, None)`. Each has a probability of $1/2$ since 2 cards remain in the deck. Note that in the second successor state, the deck is set to `None` to signify the game ended with a bust. You should also set the deck to `None` if the deck runs out of cards.

a. [15 points] Implement the game of Blackjack as an MDP by filling out the `succAndProbReward()` function of class `BlackjackMDP`. Note: if you are experiencing TimeOut, it's very likely due to incorrect implementations instead of optimization related issues. Also be careful with numbers in if conditions. 0 is equivalent to False in Python and can potentially cause if statements to not execute as expected.

## Problem 4: Learning to Play Blackjack

So far, we've seen how MDP algorithms can take an MDP which describes the full dynamics of the game and return an optimal policy. But suppose you go into a casino, and no one tells you the rewards or the transitions. We will see how reinforcement learning can allow you to play the game and learn its rules & strategy at the same time!

a. [8 points] You will first implement a generic Q-learning algorithm `QLearningAlgorithm`, which is an instance of an `RLAlgorithm`. As discussed in class, reinforcement learning algorithms are capable of executing a policy while simultaneously improving that policy. Look in `simulate()`, in `util.py` to see how the `RLAlgorithm` will be used. In short, your `QLearningAlgorithm` will be run in a simulation of the MDP, and will alternately be asked for an action to perform in a given state (`QLearningAlgorithm.getAction`), and then be informed of the result of that action (`QLearningAlgorithm.incorporateFeedback`), so that it may learn better actions to perform in the future.

We are using Q-learning with function approximation, which means $\hat{Q}^\star(s, a) = \mathbb{w} \cdot \phi(s, a)$, where in code, $\mathbb{w}$ is `self.weights`, $\phi$ is the `featureExtractor` function, and $\hat{Q}^\star$ is `self.getQ`.

We have implemented `QLearningAlgorithm.getAction` as a simple $\epsilon$-greedy policy. Your job is to implement `QLearningAlgorithm.incorporateFeedback()`, which should take an $(s, a, r, s')$ tuple and update `self.weights` according to the standard Q-learning update.

b. [2 points] This function simulates using your Q-learning code and the `identityFeatureExtractor()` on 1) a small MDP and 2) a large MDP, each with 30000 trials. For the small MDP, how does the Q-learning policy compare with a policy learned by value iteration (i.e., for how many states do they produce a different action)? We have provided a function in the grader `4b-helper` which does this for you.

Now, for the large MDP, how does the policy learned in this case compare to the policy learned by value iteration? What went wrong?

**What we expect:** A sentence explaining whether Q-learning is better or worse than ValueIteration on the small MDP. A sentence explaining whether Q-learning is better or worse than ValueIteration on the large MDP. 2-3 sentences describing why you think this behavior occurred.

The proportion of states where Q-learning's actions differ from the actions chosen by value iteration will vary slightly due to differences in implementation, as well as whether the simulation was run just once for each MDP, or multiple times. (On each run, there is some randomness in the results due to the random exploration behavior of the Q-learning algorithm.) In general, though, the expected ranges for this problem are 0-10% of states for smallMDP, and 30-33% of states for largeMDP.

In providing an explanation for Q-learning's worse performance on largeMDP vs. smallMDP, the most important factor to mention for full credit is the fact that the state space for largeMDP is much larger than the state space for smallMDP. Random exploration of (state, action) pairs over a large state space is often not sufficient to allow the Q-learning algorithm to learn accurate Q-values for each such pair, even with a large number of iterations. Q-learning does better on smallMDP because the state space is relatively small, so the algorithm is likely to encounter each possible (state, action) pair many times, and can thus learn more accurate Q-values.

A secondary factor worth noting is that our identityFeatureExtractor uses features that are unique to each (state, action) pair, so our function approximation weights cannot be used to generalize learned Q-values to (state, action) pairs that haven't been seen.

Some common mistakes:

- Mentioning the problem of feature generalization without mentioning the large state space for largeMDP. (The former problem wouldn't matter as much if not for the latter problem, as we can see in the case of smallMDP, where our feature extractor still isn't great but Q-learning nevertheless performs pretty well.)
- Highlighting the specific behavior of the Q-learning algorithm in terms of how and why specific actions differed. A number of students gave an explanation about how Q-learning on either MDP was more "aggressive" or more "conservative" based on the number of times it chose to take/peek/quit. Remember that the Q-learning algorithm has no notion of what it means to be "aggressive" or "conservative" in this or any other game; it simply tries to estimate rewards over a sequence of states/actions, but can't do that very well if it hasn't explored enough of those states/actions.
- Giving an explanation of how Q-learning performs worse than value iteration because it doesn't know the rewards or transition probabilities in advance. While this is true, this fact doesn't offer us any insight into why Q-learning does a good job approximating VI on smallMDP, but does a relatively poor job approximating the optimal solution on largeMDP.

c. [5 points] To address the problems explored in the previous exercise, let's incorporate some domain knowledge to improve generalization. This way, the algorithm can use what it has learned about some states to improve its prediction performance on other states. Implement `blackjackFeatureExtractor` as described in the code comments. Using this feature extractor, you should be able to get pretty close to the optimum values on the `largeMDP`. Note that the policies are not necessarily the same.

d. [2 points] Sometimes, we might reasonably wonder how an optimal policy learned for one MDP might perform if applied to another MDP with similar structure but slightly different characteristics. For example, imagine that you created an MDP to choose an optimal strategy for playing "traditional" blackjack, with a standard card deck and a threshold of 21. You're living it up in Vegas every weekend, but the casinos get wise to your approach and decide to make a change to the game to disrupt your strategy: going forward, the threshold for the blackjack tables is 17 instead of 21. If you continued playing the modified game with your original policy, how well would you do? (This is just a hypothetical example; we won't look specifically at the blackjack game in this problem.)

We have provided a function in the grader `4d-helper` which does the following: (1) First, the function runs value iteration on the `originalMDP` to compute an optimal policy for that MDP. Then, the optimal policy for `originalMDP` is fixed and simulated on `newThresholdMDP`. (2) Then, the function simulates Q-learning directly on `newThresholdMDP` with `blackjackFeatureExtractor` and the default exploration probability. What is the expected (average) rewards from the simulation in (1)? What is the expected (average) rewards under the new Q-learning policy in (2)? Provide some explanation for how the rewards compare, and why they are different.

**What we expect:** One sentence giving the approximated expected rewards for the transferred originalMDP and the directly learned Q-learning policy on the `modifiedMDP` and which policy is better. 2-3 sentences explaining why you think the one policy is better than the other.

> You get relatively low rewards (approximately 6.84) for `FixedRLAlgorithm` because you are passing in the policy learned for originalMDP. Because `FixedRLAlgorithm` tries to take actions which are not optimal actions for `newThresholdMDP`.
>
> Running Q-learning directly on the `newThresholdMDP` produces a policy with higher rewards (approximately 9.57) because it is able to take into consideration the new threshold for `newThresholdMDP` -- in this case, the higher threshold value of 15 (vs. 10 in the original MDP).

## Problem 5: Modeling Sea Level Rise

Sometimes the skills we learn by playing games can serve us well in real life. In this assignment, you've created a MDP that can learn an effective policy for Blackjack. Now let's see how an MDP can help us make decisions in a scenario with much higher stakes: climate change mitigation strategy [1]. Climate change can cause sea level rise, higher tides, more frequent storms, and other events that can damage a coastal city. Cities want to build infrastructure to protect their citizens, such as seawalls or landscaping that can capture storm surge, but have limited budgets. For this problem, we have implemented an MDP `SeaLevelRiseMDP` in `submission.py` that models how a coastal city government adapts to rising sea levels over the course of multiple decades. There are 2 actions available to the government at each timestep:

- $a_{Invest}$- Invest in infrastructure during this budget cycle
- $a_{Wait}$- Hold off in investing in infrastructure and save your surplus budget

Every simulation starts out in the year **2000** and with an initial sea level of **0** (in centimeters). The initial amount of money (in millions of USD), initial amount of infrastructure (unitless), and number of years to run the simulation for are parameters to the model. Every **10 years**, the city government gets a chance to make an infrastructure decision. If the city government *chooses to invest* in infrastructure for that 10 year cycle, then **the current infrastructure state is incremented by 3** and **the current budget decreases by $2 mil**. If the city government government *chooses to wait* and not invest this cycle, then the infrastructure state remains the same and **the budget increases by $2 mil**. Typically, **no reward is given until the end of the simulation**. However, if discounting is being applied, then at each time step, the **current budget** is given as reward.

However, the budget is not the only thing increasing in our simulation. At each 10 year timestep, the sea level rises by a non-deterministic amount. Specifically, it can **rise a little (1 cm.), a moderate amount (2 cm.), or a lot (3 cm.)** similar to the IPCC sea level rise projection [2]. A moderate rise in sea level is the most likely at each timestep (50% probability), but there is some probability a less or more extreme rise could occur (25% each). Normally, so long as the current sea level below the current infrastructure level, the city can go about business as usual with no punishment. However, if at any point the current sea level surpasses the current infrastructure, then **the city is immediately flooded**, the simulation ends, and the city incurs a large negative reward, simulating the cost of the city becoming uninhabitable.

The threat of sea level is not equal across coastal cities, however. For example, Tampa Bay, FL also experiences hurricanes regularly, and rising sea levels significantly exacerbate the damage caused by these extreme weather events [3]. In order to better model cities vulnerable to extreme weather events, we can toggle a boolean `disaster`. When `True`, at each time step there is a small possibility that the city is immediately flooded by a natural disaster, which is higher when the sea level is close to the infrastructure level and lower when the sea level is much lower than the infrastructure level. If the city manages to avoid being flooded by the sea until the final year of simulation, then the **current budget is given as reward** and the simulation is ended. However, if the sea level has overtaken the city's infrastructure in this final year, the city does **not** receive the reward and receives the same negative reward as before.

Using this MDP, you will explore how the optimal policy changes under different scenarios and reason about the strengths and drawbacks of modeling social decisions with machine learning and search.

a. [2 points] Imagine you're on L.A.'s city council and you're interested in understanding how sea level rise will impact the city in the coming years. To do so, you will compare 2 simple setups of the `SeaLevelRiseMDP`: one where the simulation is run for **40 years** and one where the simulation is run for **100 years**. We have already implemented functions with the proper parameters to do this with you; all you need to do is run grader test `5a-helper` to examine the optimal policy for these two MDPs. Note that the only parameter difference between the two MDPs is **the number of years to run the simulation for**. Looking at only a 40 year time horizon, interpret the MDP's optimal sequence of actions into the most economical plan for the city government to take. What about for a 100 year time horizon? Using your understanding of the MDPs, why do you think the two MDPs recommended these different economic policies?

**What we expect:** A 1-2 sentence indication of what series of action/s are economically preferable for both the 40 year horizon MDP and the 100 year horizon MDP. A 1-2 sentence explanation as to why this would be the case.

Note: you do not need to write out a full action sequence for the entire simulation - just a general description of what the most economic policy is will suffice.

> The 40 year horizon MDP will return only wait actions, which means that if only looking short-term, it's most economical for the government to make no infrastructure improvements to ward off the encroaching sea. However, the 100 year MDP returns a mix of wait and invest actions, so in the long term it's most economical to invest occasionally in infrastructure, rougly every 30 years. The reason that the MDP suggests waiting for short time horizons but investing for long time horizons is because the city already has a decent amount of infrastructure (12) and in 40 years, the sea level rise hasn't caught up yet. However, after 40 years, the sea level will overtake the infrastructure quickly, so the city needs to invest to prevent flooding.

b. [2 points]

In addition to the economic reasons for choosing one economic plan over another that you just explored, there are ethical considerations as well. Consider the following arguments:

- **Veil of Ignorance**: Imagine that you did not know what year you would be born and therefore what state of climate change you would be experiencing. The policy you would choose on that basis would be the fairest policy, as it is the one you would subject yourself to if you did not know what decade you would be born. [4]
- **Uncertain Future**: The future is uncertain. We should not sacrifice present well-being or consumption to ward off a future that might or might not happen.
- **Precautionary Risks**: People have different attitudes towards risk and uncertainty. Doing nothing now represents a significant risk of degraded living conditions for those who will be alive in the future. Since we do not know the risk-tolerance of future people, we ought to act conservatively on their behalf. [5]
- **Symmetry of Future Generations**: People born today and people born in 50 years are equally morally important and deserving of well-being. Privileging the well-being of our generation would be arbitrary; instead we should give their interests and ours equal consideration.
- **Vulnerability of Future Generations**: The well-being of people in future generations is completely dependent on the environmental choices we make today, yet they have no say in those choices. There is no morally relevant reason why we get to make those choices for them; we just happened to be born earlier. Thus we should give their interests and ours equal consideration. [1]

Using your answer for 5a, what investment policy would you advocate for the L.A. city government to use to decide its economic agenda for the next 50 years? Support your argument using one of the above ethical considerations, or another one with proper definition and citation.

**What we expect:** 2-3 sentences advocating for predicted economic policy, justified with one of the ethical considerations above (or self-defined, with references). Be sure to explicitly state which MDP you chose and reiterate the optimal investment strategy this MDP suggests. Also be sure to explicitly state which ethical argument you are using in your answer for full credit.

> An MDP / economic policy should have been mentioned or inferred from the description, along with an ethical consideration and that consideration should be connected to the chosen MDP with a logical argument that makes sense. An example answer would be: "I would advocate the city council use the 100 year MDP using the Vulnerability of Future Generations argument. Since the 40 year MDP leads to no investment and the sea level oftentimes gets very close to the infrastructure level at the end of the simulation, this means people born around or after 2040 will have a higher likelihood that the city will be flooded in their lifetime than for the people born in 2000. If we care about the future well-being as equally as the well-being now, we should keep the risk of flooding equal across generations and thus invest more to reduce the chances of flooding after 2040. The 100 year MDP recommends a balance of investing and waiting, so that economic policy is closest to what we want."

c. [2 points]

Not all cities have as consistent weather as L.A. Imagine instead that you're on the city council for Tampa, Florida - a city almost as well known for its hurricanes as it is its beaches. The Tampa city council is making a centennial plan for the city. Recent climate models indicate that the sea level may rise as much as 200 cm over the next century, with much uncertainty. A friend of yours on L.A city council has recommended using the same MDP from above to help plan your actions.

To adapt the MDP for Tampa, we need to model an increase in devastating weather events like hurricanes in addition to the steady rise of sea level. You will compare 2 similar setups of the `SeaLevelRiseMDP` where there is a small chance of major disaster every 10 years. The key difference between these two MDPs is that one **discounts future rewards** [6], while the other considers rewards from all states equally. We have already implemented functions with the proper parameters to do this with you; all you need to do is run grader test `5c-helper` to examine the optimal policy for these two MDPs. Note that the only parameter difference between the two MDPs is the discount factor. How much infrastructure should you be considering adding in your centennial plan according to the discounted model? What about for the non-discounted model? Using your understanding of the MDP, why do you think the two MDPs recommended these actions?

**What we expect:** A 1-2 sentence indication of how much infrastructure additions both the discounted and non-discounted MDPs recommend. A 1-2 sentence explanation as to why the different strategies would be recommended by the two MDPs. Please make sure your answer includes indications and explanations for both the discounted and non-discounted MDPs. Note: you do not need to quantify anything, just simply a general explanation of a policy will do.

Students should have mentioned 4 things:

1. That the discounted MDP recommends few infrastructure additions
2. The non-discounted MDP recommends a half and half policy on infrastructure and waiting
3. An explanation for the discounted MDP's recommendation of mainly waiting and
4. An explanation for the non-discounted MDP's recommendation of half waiting and investing

   An example would be "the discounted MDP recommends few infrastructure upgrades, while the non-discounted MDP recommends significant infrastructure additions. The reason that the discounted MDP suggests waiting is that flooded states disproportionately occur in later years, so the discount punishment doesn't outweigh the immediate reward of more budget given at each time step. With the non-discounted MDP, reward is only given at the end of the simulation and so minimizing the risk of flooding before the end of simulation is paramount, so the MDP chooses invest half the time to prevent ever reaching the flooded state, while still maintaining some budget for eventual reward.
   Common mistakes were:

   - Not explaining the non-discounted MDP's behavior.
   - Mixing up the output for the discounted and non-discounted MDP.
   - Failing to explain why the discounting of future reward leads the discounted MDP to invest less.
   - Failing to explain why considering future reward as equally as current reward would lead the non-discounted MDP to invest and wait equally.

d. [2 points] Finally, remember in question 4d we compared how well (or poorly!) one MDP's optimal policy would transfer to another. If we're no longer quite as good at playing blackjack, that's one thing. However, in a high-stakes scenario like this, supposedly optimal policies that are in fact suboptimal can have severe consequences that affect the livelihoods and well-being of potentially millions of people.

We will now explore what happens when we mis-estimate the cost of climate change. Imagine you're still on Tampa's city council. Recently, the city was hit by a small hurricane which flooded a few parts of the outskirts of the city. The estimated cost of the flooding was -$10 million. You decide to run your sea level rise MDP with this as the negative reward and you present the optimal policy to city council, which decides to use the MDP's policy to set their infrastructure economic agenda for the foreseeable future.

**Part 1.)** Run `5d-helper` to output the expected reward of the optimal policy from ValueIteration running the `SeaLevelRiseMDP` for 100 years with a flooding cost of -$10 million.

**Part 2.)** Now, let's imagine that this flooding cost underestimates the cost of flooding by 3 orders of magnitude (read: a flooding cost of -$10 billion). Next in the output from `5d-helper`, you'll find the expected reward of the above fixed optimal policy re-run on an MDP with the more realistic flooding cost.

**Part 3.)** Finally, you will see the list of actions predicted by the optimal policy for the -10 mil. flooding cost MDP and the optimal policy for the -$10,000 mil. flooding cost MDP.

Given these findings, what do you advise city council to do in regards to their infrastructure economic agenda?

**What we expect:** 1-2 sentences discussing whether or not you think the city council should still use the -10 mil. flooding cost MDP to make infrastructure economic decisions. If you think city council should still use the -10 mil. flooding cost MDP, provide justification through either the results from `5d-helper` or other outside sources. If you think city council should not use the -10 mil. flooding cost MDP, suggest an alternate way city council can still safely incorporate the predictions from `SeaLevelRiseMDP` into their economic decisions (i.e., suggest a reasonable change to fix the potential issue making you think that the city council should not use the current -$10 mil. flooding cost MDP.)

Students should have stated whether they would use the -$10mil MDP or not (or have it be inferrable from context.) A reasonable additon / change should have been given if the recommendation was for city council to not use the MDP. If it was recommended the council still use the MDP, then some citations or interpretation of the data from `5d-helper` with a logical interpretation should have been provided.
An example solution would be: "I believe Tampa city council should not use the -10 mil MDP because if the -10mil cost estimate of flooding is lower than expected, then from Part 2 the city will have catastrophic flooding and potentially be uninhabitable from the level of damage. One way to still incorporate the MDP's predictions would be to use a more conservative estimate of the flooding cost, either using -$10bil as the flooding cost or better yet consulting economic and climate experts to provide a more realistic value of future flooding costs."
Common mistakes included:

- Mentioning Q learning would fix the problem. If the estimated cost of flooding is wrong, Q learning will still learn a sub-optimal policy.
- Forgetting to suggest a change the city council should make to still use the MDP.

[1] Shuvo et al. A Markov Decision Process Model for Socio-Economic Systems Impacted by Climate Change. ICML. 2020.

[2] IPCC. IPCC AR6 Sea Level Projection Tool. IPCC 6th Assessment Report. 2021.

[3] Marsooli et al. Climate change exacerbates hurricane flood hazards along US Atlantic and Gulf Coasts in spatially varying patterns. Nature Communications. 2019.

[4] Moellendorf et al. Justice and the Assignment of the Intergenerational Costs of Climate Change.

[5] Buchak et al. Weighing the Risks of Climate Change .

[6] Discount Rates: a boring thing you should know about.

[7] Trump Put a Low Cost on Carbon Emissions. Here's Why It Matters.