```
import random
import numpy as np
from tqdm import tqdm
```

Rover

Monte Carlo Control

Consider a rover that is trying to move up the hill. The robot has solar powered motors. Once the robot **reaches the top** of the hill, the **episode ends**. The Robot can either be at **low, medium, high or top** of the hill at any given point of time.

The robot can either decide to spin it's wheels **slowly or rapidly** in order to move up on the gradient.

- If the **motor spins the wheel slowly**, with probability `0.6` it moves to the next higher state in one time step, and with probability `0.4`, it slides all the way down the slope to the low state.
- On the other hand, if the **motor spins the wheel rapidly**, with probability `0.9` it moves to the next higher state in one time step, and with probability `0.1`, it slides all the way down the slope to the low state.

The robot initially starts from low state with 10 units of energy (battery charge). The **episode ends** if the batteries get completely discharged or if rover reaches the top position.

The rover has a motor that can spin its wheel

- slowly at the expense of 2 unit of energy per time step;
- or rapidly at the expense of 4 units of energy per time step.

rover gains 4 units of energy per time step from its solar panels any time it transitions **upward from the medium position** as it gets exposed to sunlight.

Goal:

The Robot has to reach the top with maximum amount of energy left in the battery (Energy spent while moving to the top must be minimum)

Tasks

PART 1

- Create an "Environment" class that has the step() method which takes in an action and returns the next state and one step reward.
- Try taking random actions until the eposode terminates. Print the one-step reward, action and next state for all transitions in the episode. What is the average total reward if you run the experiment over many episodes?
- What is the average total reward over 10,000 episodes if:
    - Actions are chosen "randmoly" from the action space?
    - Action is always "spin_slowly"?
    - Acion is always "spin_rapidly"?

PART 2

- Create an "Agent" class with train() and policy() methoods which learns to take actions in this environment. Agent should be trainable using Monte-Carlo Control.
- Train the agent. What is the average total return after training the Agent?

  (optional)

- Perform monte-carlo prediction of state-values to evaluate the learned policy. How valuable is it to start from each state?

```
INITIAL_CHARGE = 10
INITIAL_STATE = "low"

class RoverEnv:
  """It can be inferred from the problem that:

  State space = {"low", "medium", "high", "top"}
  Action space = {spin_slowly, spin_rapidly}
  Ex. `action = spin_slowly` means the agent took the action spin_slowly.

  Start state
  -----------
  Robot starts low position with 10 units charge.

  Rewards
  -------
  Amount of energy left in the robot when episode ends is the reward.
  the one step-reward is received only once i.e, when the episode ends.

  Terminal conditions
  -------------------
  Episode end is:
  - Robot transitions to the top postion.
  - There is no charge left in the batteries.
  """

  def __init__(self):
    self.actions = ["spin_slowly", "spin_rapidly"]
    self.action_space = self.actions.copy()
    #self.action_space = ["spin_slowly", "spin_rapidly"]
    self.states = ["low", "medium", "high", "top"]
    self.observation_space = self.states.copy()
    #self.observation_space = ["low", "medium", "high", "top"]

    self.reset()

  def reset(self):
    self.state = INITIAL_STATE
    self.charge = INITIAL_CHARGE
    self.terminated = False

    self.total_reward = 0
```

```
    def get_transition_probability(self, start_state, end_state, action):

      # probability of moving "up" when motor spins "slowly"
      slow_spin_up_p = 0.6
      # probability of moving "up" when motor spins "rapidly"
      rapid_spin_up_p = 0.9

      # probability of moving "down" when motor spins "slowly"
      slow_spin_down_p = 1-slow_spin_up_p
      # probability of moving "down" when motor spins "rapidly"
      rapid_spin_down_p = 1-rapid_spin_up_p


      if action == "spin_slowly":
        if start_state == "low" and end_state == "medium":
          return slow_spin_up_p
        if start_state == "low" and end_state == "low":
          return slow_spin_down_p
        if start_state == "medium" and end_state == "high":
          return slow_spin_up_p
        if start_state == "medium" and end_state == "low":
          return slow_spin_down_p
        if start_state == "high" and end_state == "top":
          return slow_spin_up_p
        if start_state == "high" and end_state == "low":
          return slow_spin_down_p

      if action == "spin_rapidly":
        if start_state == "low" and end_state == "medium":
          return rapid_spin_up_p
        if start_state == "low" and end_state == "low":
          return rapid_spin_down_p
        if start_state == "medium" and end_state == "high":
          return rapid_spin_up_p
        if start_state == "medium" and end_state == "low":
          return rapid_spin_down_p
        if start_state == "high" and end_state == "top":
          return rapid_spin_up_p
        if start_state == "high" and end_state == "low":
          return rapid_spin_down_p

      return 0

    def get_charge_difference(self, start_state, end_state, action):
```

```
        charge_diff = 0


        if action == "spin_slowly":
          charge_diff -= 2
        elif action == "spin_rapidly":
          charge_diff -= 4


        if start_state == "medium" and end_state == "high":
          charge_diff += 4
        elif start_state == "medium" and end_state == "high":
          charge_diff += 4

        return charge_diff



      def get_reward(self, end_state):
        if end_state == "top":
          return self.charge
        return 0

      def step(self, action):

        if self.state == "top":
          self.terminated = True
          return self.state, self.charge, self.terminated

        next_states = self.observation_space
        transition_probs = [self.get_transition_probability(self.state, next_state, action) for next_state in next_states]
        next_state = np.random.choice(next_states, p=transition_probs)

        self.charge += self.get_charge_difference(self.state, next_state, action)
        one_step_reward = self.get_reward(next_state)

        self.state = next_state




        if self.state == "top" or self.charge <=0:
          self.terminated = True
```

```
        self.total_reward+=one_step_reward

        return self.state, one_step_reward, self.terminated



class Agent:
    def __init__(self, env):
        self.env = env
        self.Q = {state: {action: 0 for action in env.actions} for state in env.states}
        self.returns = {state: {action: [] for action in env.actions} for state in env.states}
        self.policy = {state: random.choice(env.actions) for state in env.states}

    def train(self, num_episodes=10000, epsilon=0.1):
        for _ in range(num_episodes):
            self.env.reset()
            state = self.env.state
            episode = []

            while not self.env.terminated:
                action = self.policy[state] if random.random() > epsilon else random.choice(self.env.actions)
                next_state, reward, _ = self.env.step(action)
                episode.append((state, action, reward))
                state = next_state

            G = 0
            for state, action, reward in reversed(episode):
                G = reward + G
                self.returns[state][action].append(G)
                self.Q[state][action] = np.mean(self.returns[state][action])
                self.policy[state] = max(self.Q[state], key=self.Q[state].get)

    def get_action(self, state):
        return self.policy[state]

    def reset(self):
        self.Q = {state: {action: 0 for action in env.actions} for state in env.states}
        self.returns = {state: {action: [] for action in env.actions} for state in env.states}
        self.policy = {state: random.choice(env.actions) for state in env.states}
```

```
env = RoverEnv()
agent = Agent(env)
agent.train()
```

Random Actions

```
total_rewards_random = 0
for _ in range(10000):
    env.reset()
    while not env.terminated:
        action = random.choice(env.actions)
        _, reward, _ = env.step(action)
        total_rewards_random += reward

print("Average total reward with random actions: ", total_rewards_random / 10000)
```

```
Average total reward with random actions:  2.084
```

With always spin slowly

```
# Evaluate the agent with always 'spin_slowly'
total_rewards_slowly = 0
for _ in range(10000):
    env.reset()
    while not env.terminated:
        action = 'spin_slowly'
        _, reward, _ = env.step(action)
        total_rewards_slowly += reward

print("Average total reward with 'spin_slowly': ", total_rewards_slowly / 10000)
```

```
Average total reward with 'spin_slowly':  3.0948
```

With always spin rapidly

```
# Evaluate the agent with always 'spin_rapidly'
total_rewards_rapidly = 0
for _ in range(10000):
    env.reset()
```

```
    while not env.terminated:
        action = 'spin_rapidly'
        _, reward, _ = env.step(action)
        total_rewards_rapidly += reward

print("Average total reward with 'spin_rapidly': ", total_rewards_rapidly / 10000)
```

```
Average total reward with 'spin_rapidly':  1.2872
```

```
total_rewards_random = 0
num_episodes = 10000
for _ in range(num_episodes):
    env.reset()
    while not env.terminated:
        action = random.choice(env.actions)
        next_state, reward, _ = env.step(action)
        print(f"Action: {action}, Reward: {reward}, Next State: {next_state}")
        total_rewards_random += reward

print("Average total reward with random actions: ", total_rewards_random / num_episodes)
```

Streaming output truncated to the last 5000 lines.

```
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 4, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 0, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 4, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 4, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 4, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 6, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 0, Next State: low
Action: spin_rapidly, Reward: 0, Next State: medium
```

```
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 2, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 6, Next State: top
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: -2, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 6, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 8, Next State: top
Action: spin_slowly, Reward: 0, Next State: low
Action: spin_rapidly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 2, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_slowly, Reward: 0, Next State: high
Action: spin_rapidly, Reward: 6, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
Action: spin_rapidly, Reward: 0, Next State: high
Action: spin_slowly, Reward: 6, Next State: top
Action: spin_slowly, Reward: 0, Next State: medium
```

```
env = RoverEnv()
agent = Agent(env)
```