

TEAM LEADER NAME – PRIYANSHI AGRAWAL

ROLL NO – 2301CS90

MEMBER 2 NAME – MANVITHA REDDY

ROLL NO – 2301CS29

MEMBER 3 NAME – SHIKSHA RAGINEE

ROLL NO – 2301AI13

MEMBER 4 NAME – GOMPA VASAVI

ROLL NO – 2301AI43

MEMBER 5 NAME – PREETHI

ROLL NO – 2301AI17

MEMBER 6 NAME – GURU SAI BHASKARI

ROLL NO – 2301AI19

```

9
10 import sys
11
12 Tabnine | Edit | Test | Explain | Document | Cody
13 def cliff_menu():
14     print("\nCliff Climbing Game selected.")
15     print("Choose algorithm:")
16     print("1. SARSA")
17     print("2. Q-Learning")
18     algo_choice = input("Enter your choice (1 or 2): ").strip()
19
20     if algo_choice == "1":
21         import cliff_climbing
22         return cliff_climbing.train_sarsa, cliff_climbing.test_sarsa
23     elif algo_choice == "2":
24         import cliff_climbing
25         return cliff_climbing.train_q_learning, cliff_climbing.test_q_learning
26     else:
27         print("Invalid algorithm choice.")
28         sys.exit()
29
30 Tabnine | Edit | Test | Explain | Document | Cody
31 def taxi_menu():
32     print("\nTaxi Game selected.")
33     import taxi
34     return taxi.train_taxi, taxi.test_taxi
35
36 Tabnine | Edit | Test | Explain | Document | Cody
37 def blackjack_menu():
38     print("\nBlackjack Game selected.")
39     import blackjack
40     return blackjack.train_blackjack, blackjack.test_blackjack
41
42 Tabnine | Edit | Test | Explain | Document | Cody
43 def mountaincar_menu():
44     print("\nMountainCar Game selected.")
45     import mountaincar
46     return mountaincar.train_mountaincar, mountaincar.test_mountaincar
47
48 Tabnine | Edit | Test | Explain | Document | Cody
49 def main():
50     print("Welcome to the Game Trainer & Tester!")
51     print("Select an operation:")
52     print("1. Train a game")
53     print("2. Test a game")
54     operation = input("Enter 1 or 2: ").strip()
55
56     if operation not in ["1", "2"]:
57         print("Invalid selection. Exiting.")
58         sys.exit()
59
60     print("\nSelect the game:")
61     print("1. Cliff Climbing")
62     print("2. Taxi")
63     print("3. Blackjack")
64     print("4. MountainCar") # <-- add this line
65
66     game_choice = input("Enter game choice (1, 2, 3 or 4): ").strip()
67
68     # game_choice = input("Enter game choice (1, 2 or 3): ").strip()
69
70     # if game_choice == "1":
71     #     train_func, test_func = cliff_menu()
72     # elif game_choice == "2":
73     #     train_func, test_func = taxi_menu()
74     # else:
75     #     print("Invalid game choice. Exiting.")
76     #     sys.exit()
77
78     if game_choice == "1":
79         train_func, test_func = cliff_menu()
80     elif game_choice == "2":
81         train_func, test_func = taxi_menu()
82     elif game_choice == "3":
83         train_func, test_func = blackjack_menu()
84     elif game_choice == "4":
85         train_func, test_func = mountaincar_menu()
86     else:
87         print("Invalid game choice. Exiting.")
88         sys.exit()
89
90     if operation == "1":
91         print("\nStarting training...")
92         train_func()
93     else:
94         print("\nStarting testing...")
95         test_func()
96
97 Cody
98 if __name__ == "__main__":
99     main()

```

Cliff_climbing.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
import os
from IPython.display import clear_output
import random
import pickle

#pip install gymnasium
import gymnasium as gym

Tabnine | Edit | Test | Explain | Document | Cody
def save_frames_as_gif(frames, episode, algorithm_type, path='./climbing_gif/', filename='gym_animation.gif'):
    #Mess with this to change frame size
    if not os.path.exists(path):
        os.makedirs(path)

    print(frames[0].shape)
    plt.figure(figsize=(frames[0].shape[1] / 72.0, frames[0].shape[0] / 72.0), dpi=72)

    patch = plt.imshow(frames[0])
    plt.axis('off')
    plt.title(f"Run from episode {episode} {algorithm_type}")

    def animate(i):
        patch.set_data(frames[i])

    anim = animation.FuncAnimation(plt.gcf(), animate, frames = len(frames), interval=50)
    # anim.save(path + filename, writer='imagemagick', fps=30)

    anim.save(os.path.join(path, filename), writer='imagemagick', fps=30)

Tabnine | Edit | Test | Explain | Document | Cody
def train_sarsa():
    #SARSA (On-Policy TD Control) for estimating Q
    env = gym.make('CliffWalking-v0', render_mode="rgb_array")
    # env = gym.make('CliffWalking-v0', render_mode="rgb_array", new_step_api=True)

    observation, info = env.reset()
    #print(env.action_space.n)
    #print(env.observation_space)
    action = env.action_space.sample()
    # initialize constants
    EPSILON = 0.01
    ALPHA = 0.50
    RENDER_AT_EPISODE = 10
    EPISODES = 100
    LEARNING_RATE = 0.99

    #initialize for each state and action
    episode_frames_SARSA = {}
    previous_state_action_reward = {}
    policy_given_state = {}
    Q_table = np.random.rand(env.observation_space.n, env.action_space.n) #THIS NEEDS TO BE A TABLE FOR EACH STATE AND ACTION
    Q_table[47] = np.zeros((1, env.action_space.n))

    #initialize the policy
    for interaction in range(Q_table.shape[0]):
        best_action = np.argmax(Q_table[interaction])
        policy_given_state[interaction] = np.ones((1, env.action_space.n))[0] * EPSILON / env.action_space.n
        policy_given_state[interaction][best_action] = 1 - EPSILON + (EPSILON / env.action_space.n)

    #SANITY CHECK
    for row in policy_given_state.keys():
        #print(policy_given_state[row])
        np.random.choice([0, 1, 2, 3], p=policy_given_state[row])

    time_step=0
    C = 0
    number_time_step = []
    for episode in range(EPISODES): #for each episode
        if episode%10_000 == 0:
            print(f"Currently on episode {episode}")
            frames=[]
            observation, info = env.reset() #initialize the environment
            #This is the epsilon greedy method
            if random.random() < EPSILON:
                action = random.randint(0, env.action_space.n-1)
            else:
                action = np.argmax(Q_table[observation])
            done=False
            while not done: #for each time step
                next_observation, reward, done, truncated, info = env.step(action) #take an action and get feedback from the environment

                if random.random() < EPSILON:
                    next_action = random.randint(0, env.action_space.n-1)
                else:
                    next_action = np.argmax(Q_table[next_observation])

                Q_table[observation][action] = Q_table[observation][action] + ALPHA*(reward + LEARNING_RATE*Q_table[next_observation][next_action] - Q_table[observation][action])
                observation = next_observation
                action = next_action
                time_step+=1
            if episode%RENDER_AT_EPISODE == 0:
                clear_output(wait=True)
                single_frame = env.render()
                frames.append(single_frame)
                path=os.path.join(path, f'episode_{episode}.gif')
```

```

106
107     if episode%RENDER_AT_EPISODE ==0:
108         frames.append(single_frame)
109         frames.append(single_frame)
110         frames.append(single_frame)
111         frames.append(single_ (variable) episode: int
112         episode_frames_SARSA[episode] = frames
113         save_frames_as_gif(frames,episode,"SARSA",filename=f"SARSA episode {episode}.gif")
114
115     number_time_step.append(time_step)
116     # After training is complete
117     with open('sarsa_q_table.pkl', 'wb') as f:
118         pickle.dump(Q_table, f)
119

```

Tabnine | Edit | Test | Explain | Document | Cody

```

120 def test_sarsa(): # Final test for SARSA agent
121     with open('sarsa_q_table.pkl', 'rb') as f:
122         Q_table = pickle.load(f)
123     env = gym.make('CliffWalking-v0', render_mode="rgb_array")
124     observation, info = env.reset()
125     done = False
126     frames = []
127
128     while not done:
129         action = np.argmax(Q_table[observation]) # Use the trained SARSA Q-table
130         observation, reward, done, truncated, info = env.step(action)
131         frames.append(env.render())
132
133     save_frames_as_gif(frames, episode="Final", algorithm_type="SARSA", filename="SARSA Final Test.gif")
134

```

Tabnine | Edit | Test | Explain | Document | Cody

```

135 def train_q_learning(): # THIS IS FOR Q-LEARNING
136     env = gym.make('CliffWalking-v0',render_mode="rgb_array")
137
138     ALPHA = 0.85
139     EPSILON = 0.1
140     RENDER_AT_EPISODE = 10
141     EPISODES = 100
142     LEARNING_RATE = 0.99
143     alpha = np.linspace(0.01,0.99,10)
144     learning_rate= np.linspace(0.01,0.99,10)
145
146     episode_frames_Q = {}
147     time_step=0
148     sum_of_rewards = {}
149
150     Q_table = np.zeros((env.observation_space.n,env.action_space.n)) #THIS NEEDS TO BE A TABLE FOR EACH STATE AND ACTION
151     for episode in range(EPISODES): #for each episode
152         observation,info = env.reset() #initialize the environment
153         done=False
154         rewards = 0
155         frames=[]
156         while not done: #for each time step
157             # SELECT AN ACTION USING EPSILON GREEDY METHOD
158             #if random.random() < EPSILON:
159             #    action = random.randint(0,env.action_space.n-1)
160             #else:
161             action = np.argmax(Q_table[observation])
162
163             next_observation, reward, done, truncated,info = env.step(action) #take an action and get feedback from the environment
164             Q_table[observation][action] = Q_table[observation][action] + ALPHA*(reward + LEARNING_RATE*np.max(Q_table[next_observation]) - Q_table[observation][action])
165             observation = next_observation
166             time_step+=1
167             rewards+=reward
168             if episode%RENDER_AT_EPISODE == 0:
169                 clear_output(wait=True)
170                 single_frame = env.render()
171                 frames.append(single_frame)
172                 plt.imshow(single_frame)
173                 plt.show()
174             sum_of_rewards[episode] = rewards
175             if episode%RENDER_AT_EPISODE == 0:
176                 frames.append(single_frame)
177                 frames.append(single_frame)
178                 episode_frames_Q[episode] = frames
179                 save_frames_as_gif(frames,episode,"Q-Learning",filename=f"Q-Learning episode {episode}.gif")
180         with open('sarsa_q_table.pkl', 'wb') as f:
181             pickle.dump(Q_table, f)
182

```

Tabnine | Edit | Test | Explain | Document | Cody

```

183 def test_q_learning() : # Final test for Q-Learning agent
184     with open('sarsa_q_table.pkl', 'rb') as f:
185         Q_table = pickle.load(f)
186     env = gym.make('CliffWalking-v0', render_mode="rgb_array")
187     observation, info = env.reset()
188     done = False
189     frames = []
190
191     while not done:
192         action = np.argmax(Q_table[observation]) # Use the trained Q-Learning Q-table
193         observation, reward, done, truncated, info = env.step(action)
194         frames.append(env.render())
195
196     save_frames_as_gif(frames, episode="Final", algorithm_type="Q-Learning", filename="Q-Learning Final Test.gif")

```

GAME 1 – cliff climbing—sarsa algorithm

```
PS D:\AI game\aigame> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 1

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 1

Cliff Climbing Game selected.
Choose algorithm:
1. SARSA
2. Q-Learning
Enter your choice (1 or 2): 1

Starting training...
Currently on episode 0
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
PS D:\AI game\aigame> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2

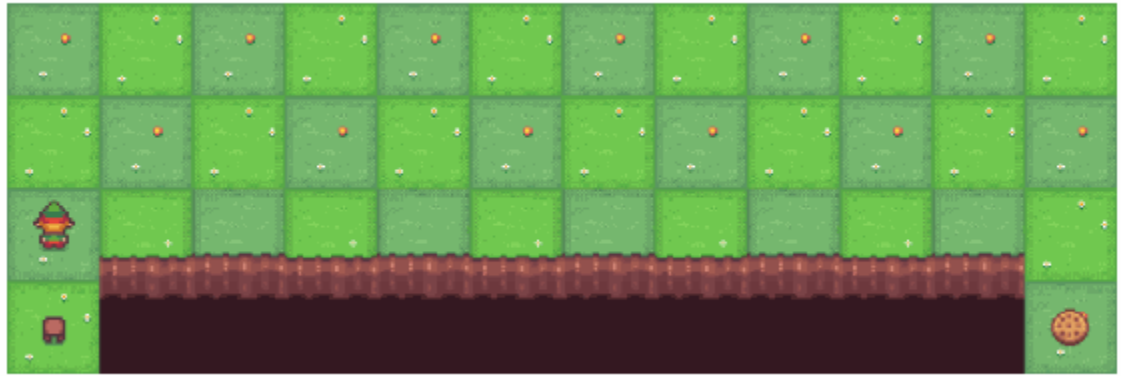
Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 1

Cliff Climbing Game selected.
Choose algorithm:
1. SARSA
2. Q-Learning
Enter your choice (1 or 2): 1

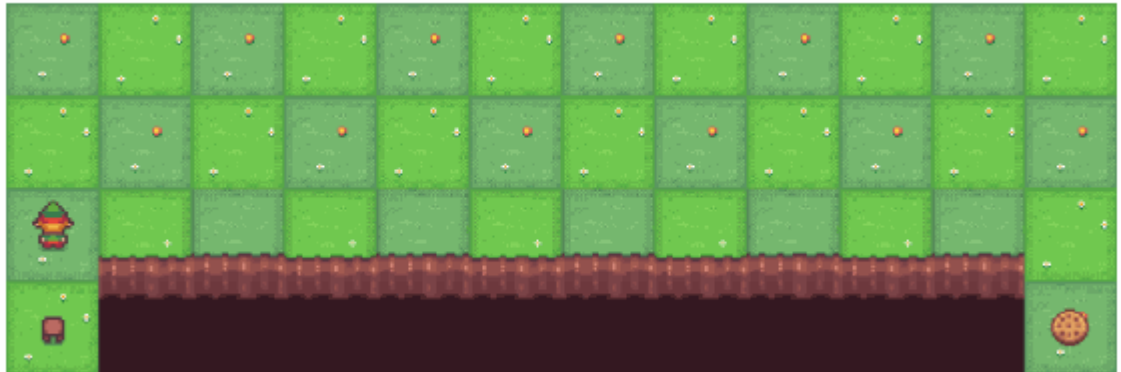
Starting testing...
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
PS D:\AI game\aigame> |
```

Result --

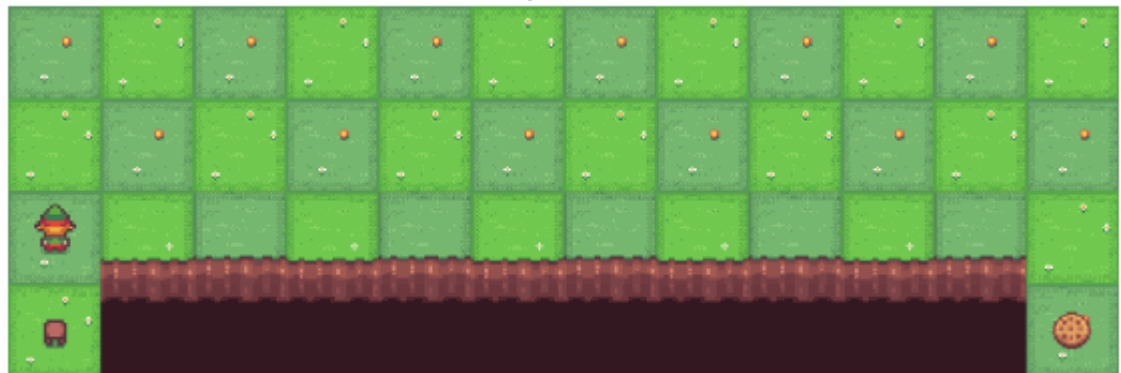
Run from episode 70 SARSA



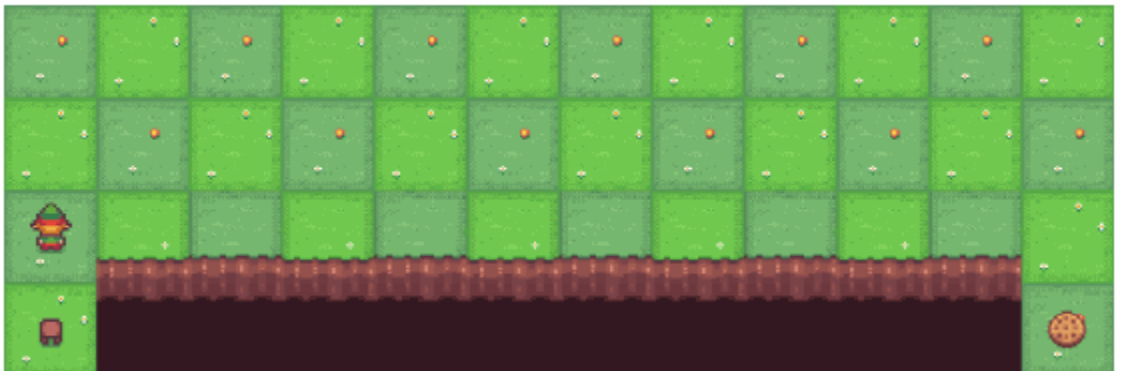
Run from episode 80 SARSA



Run from episode 90 SARSA



Run from episode Final SARSA



```

PS D:\AI game\ai_game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 1

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 1

Cliff Climbing Game selected.
Choose algorithm:
1. SARSA
2. Q-Learning
Enter your choice (1 or 2): 2

Starting training...
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
D:\AI game\ai_game\cliff_climbing.py:175: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
plt.show()
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.
PS D:\AI game\ai_game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 1

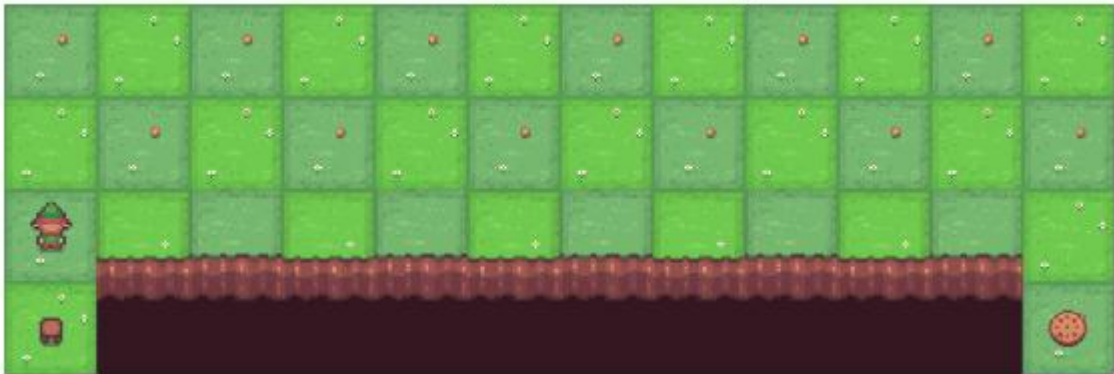
Cliff Climbing Game selected.
Choose algorithm:
1. SARSA
2. Q-Learning
Enter your choice (1 or 2): 2

Starting testing...
(240, 720, 3)
MovieWriter imagemagick unavailable; using Pillow instead.

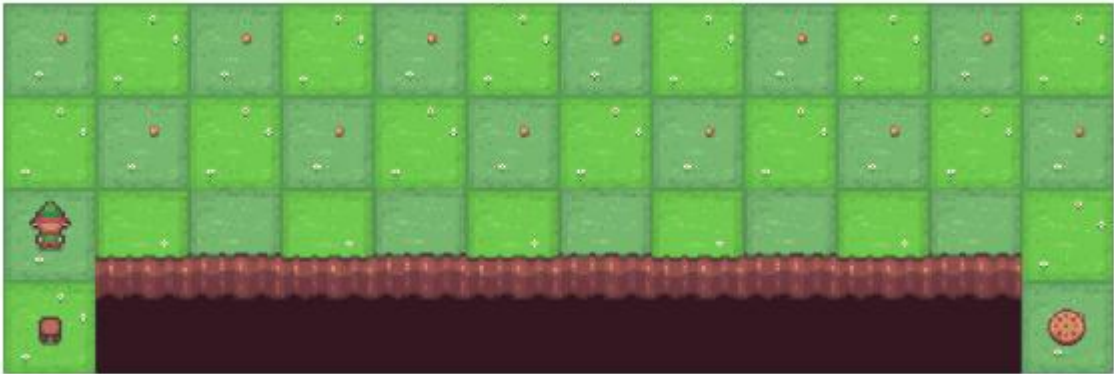
```


RESULT—

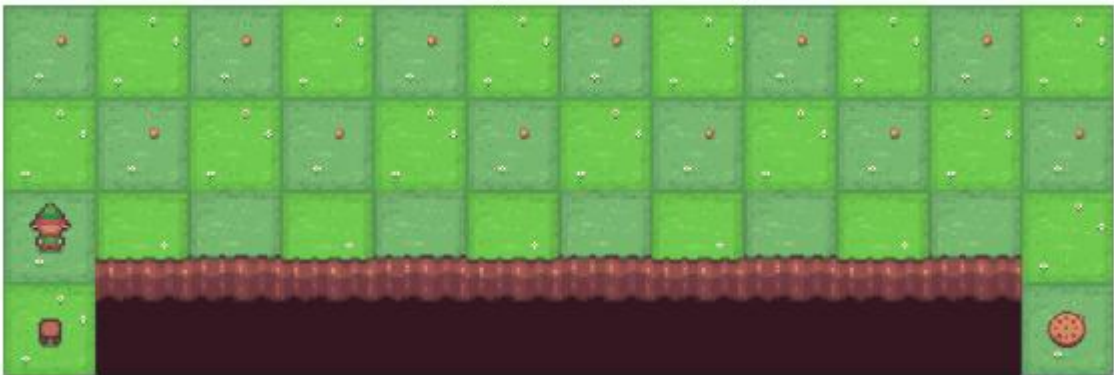
Run from episode 70 Q-Learning



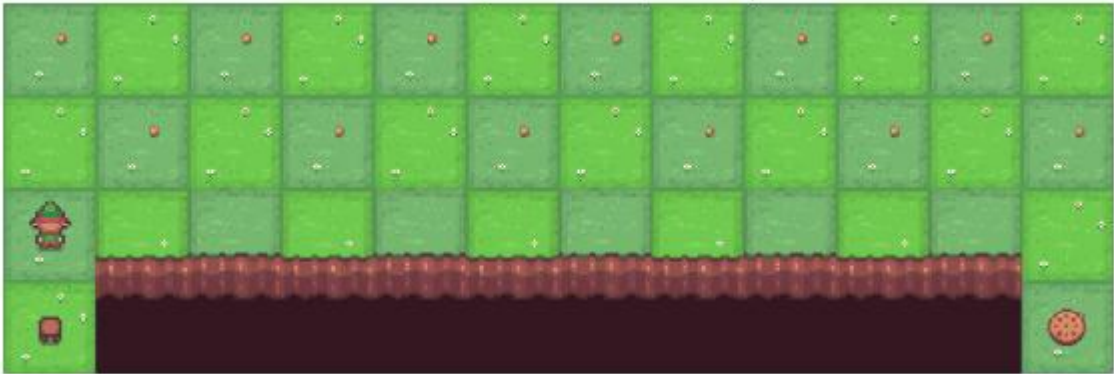
Run from episode 80 Q-Learning



Run from episode 90 Q-Learning



Run from episode Final Q-Learning



taxi.py-

```
from matplotlib import animation
import os
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import pickle
Tabnine | Edit | Test | Explain | Document | Cody
def save_frames_as_gif(frames, episode, filename='taxi_test.gif', path='./taxi_gif/'):
    if not os.path.exists(path):
        os.makedirs(path)

    plt.figure(figsize=(frames[0].shape[1]/72.0, frames[0].shape[0]/72.0), dpi=72)
    patch = plt.imshow(frames[0])
    plt.axis('off')

    def animate(i):
        patch.set_data(frames[i])

    anim = animation.FuncAnimation(plt.gcf(), animate, frames=len(frames), interval=200)
    anim.save(os.path.join(path, filename), writer='pillow')
    plt.close()
```

```

def train_taxi(epochs=15000):
    import gymnasium as gym
    import numpy as np
    import pickle

    env = gym.make('Taxi-v3')
    q = np.zeros((env.observation_space.n, env.action_space.n))

    learning_rate_a = 0.9
    discount_factor_g = 0.9
    epsilon = 1.0
    epsilon_decay_rate = 0.0001
    rng = np.random.default_rng()

    for i in range(epochs):
        state = env.reset()[0]
        terminated = False
        truncated = False

        while not terminated and not truncated:
            if rng.random() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(q[state, :])

            new_state, reward, terminated, truncated, _ = env.step(action)
            q[state, action] = q[state, action] + learning_rate_a * (
                reward + discount_factor_g * np.max(q[new_state, :]) - q[state, action]
            )
            state = new_state

        epsilon = max(epsilon - epsilon_decay_rate, 0)
        if epsilon == 0:
            learning_rate_a = 0.0001

    with open("taxi.pkl", "wb") as f:
        pickle.dump(q, f)

    env.close()

```

```

def test_taxi(epochs=5):
    import gymnasium as gym
    import pickle
    import numpy as np

    env = gym.make('Taxi-v3', render_mode='rgb_array')

    with open("taxi.pkl", "rb") as f:
        q = pickle.load(f)

    for i in range(epochs):
        state = env.reset()[0]
        terminated = False
        truncated = False
        frames = []

        while not terminated and not truncated:
            action = np.argmax(q[state, :])
            state, reward, terminated, truncated, _ = env.step(action)
            frames.append(env.render())

        save_frames_as_gif(frames, i, filename=f"taxi_episode_{i}.gif")

    env.close()

```

```
PS D:\AI game\aigame> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 1

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 2

Taxi Game selected.

Starting training...
PS D:\AI game\aigame> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2

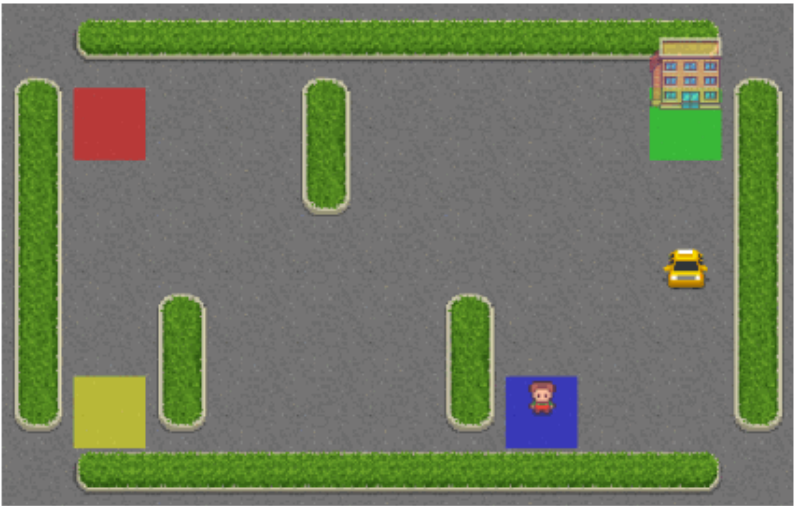
Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 2

Taxi Game selected.

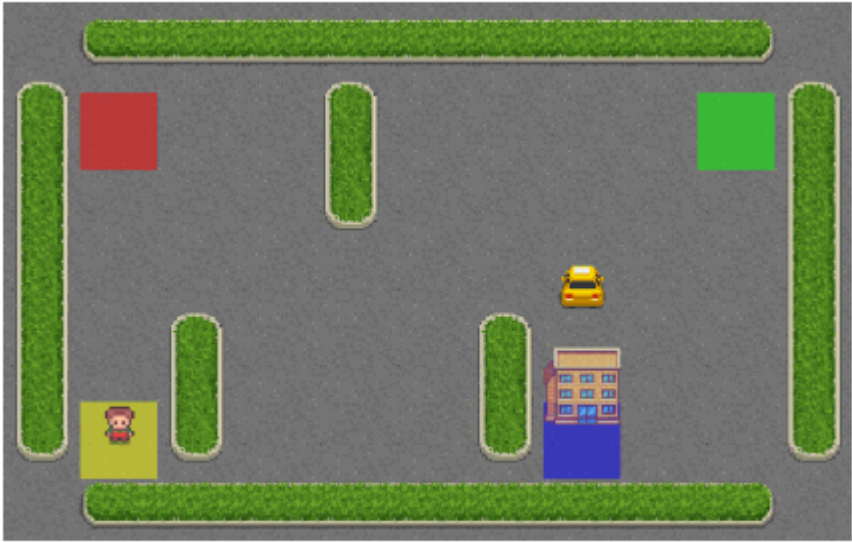
Starting testing...
```

RESULT--

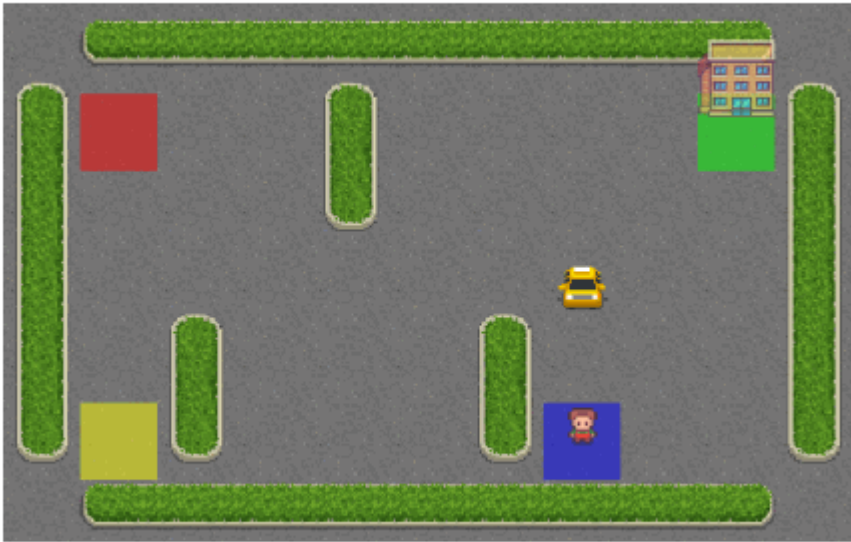
EPISODE = 2



EPISODE=3



EPISODE=4



```

10 import matplotlib.pyplot as plt
11 from matplotlib import animation
12 from matplotlib.animation import PillowWriter
13 from collections import defaultdict
14 import numpy as np
15 from tqdm import tqdm
16 import gymnasium as gym
17 import pickle
18
19 env = gym.make("Blackjack-v1", sab=True)
20 def default_q_values():
21     return np.zeros(env.action_space.n)
22 class BlackjackAgent:
23     def __init__(self, learning_rate, initial_epsilon, epsilon_decay, final_epsilon, discount_factor=0.95):
24         # self.q_values = defaultdict(lambda: np.zeros(env.action_space.n))
25         self.q_values = defaultdict(default_q_values)
26         self.lr = learning_rate
27         self.discount_factor = discount_factor
28         self.epsilon = initial_epsilon
29         self.epsilon_decay = epsilon_decay
30         self.final_epsilon = final_epsilon
31         self.training_error = []
32
33     def get_action(self, obs):
34         if np.random.random() < self.epsilon:
35             return env.action_space.sample()
36         return int(np.argmax(self.q_values[obs]))
37
38     def update(self, obs, action, reward, terminated, next_obs):
39         future_q_value = (not terminated) * np.max(self.q_values[next_obs])
40         td = reward + self.discount_factor * future_q_value - self.q_values[obs][action]
41         self.q_values[obs][action] += self.lr * td
42         self.training_error.append(td)
43
44     def decay_epsilon(self):
45         self.epsilon = max(self.final_epsilon, self.epsilon - self.epsilon_decay)
46
47
48 def train_blackjack(n_episodes=100000):
49     global agent
50     agent = BlackjackAgent(
51         learning_rate=0.1,
52         initial_epsilon=1.0,
53         epsilon_decay=1.0 / (n_episodes / 2),
54         final_epsilon=0.1,
55     )
56     env = gym.make("Blackjack-v1", sab=True)
57     env = gym.wrappers.RecordEpisodeStatistics(env, n_episodes)
58     for episode in tqdm(range(n_episodes)):
59         obs, info = env.reset()
60         done = False
61         while not done:
62             action = agent.get_action(obs)
63             next_obs, reward, terminated, truncated, info = env.step(action)
64             agent.update(obs, action, reward, terminated, next_obs)
65             done = terminated or truncated
66             obs = next_obs
67             agent.decay_epsilon()
68     with open("blackjack_agent.pkl", "wb") as f:
69         pickle.dump(agent, f)
70     print("Training complete and agent saved.")
71
72
73 def test_blackjack_one_episode():
74     try:
75         with open("blackjack_agent.pkl", "rb") as f:
76             agent = pickle.load(f)
77     except FileNotFoundError:
78         print("No trained agent found.")
79         return
80
81     agent.epsilon = 0.0
82     env = gym.make("Blackjack-v1", sab=True, render_mode="rgb_array")
83     obs, info = env.reset()
84     done = False
85     frames = []
86
87     while not done:
88         frames.append(env.render())
89         action = agent.get_action(obs)
90         obs, reward, terminated, truncated, info = env.step(action)
91         done = terminated or truncated
92
93     print("Final Reward:", reward)
94     plt.imshow(frames[-1])
95     plt.axis("off")
96     plt.title(f"Final Reward: {reward}")
97     plt.show()
98
99     fig = plt.figure(figsize=(6, 4))
100     img = plt.imshow(frames[0])
101     plt.axis("off")
102
103     def animate(i):
104         img.set_data(frames[i])
105         return [img]

```

```

106     anim = animation.FuncAnimation(fig, animate, frames=len(frames), interval=500, blit=True)
107     anim.save("blackjack_agent.gif", writer=PillowWriter(fps=2))
108     print("Animation saved as blackjack_agent.gif")
109
110
111
112 Tabnine | Edit | Test | Explain | Document | ↺ Cody
113 def test_blackjack_multiple_episodes(num_test_episodes=5):
114     try:
115         with open("blackjack_agent.pkl", "rb") as f:
116             agent = pickle.load(f)
117     except FileNotFoundError:
118         print("No trained agent found.")
119         return
120
121     agent.epsilon = 0.0
122     for episode_num in range(1, num_test_episodes + 1):
123         env = gym.make("Blackjack-v1", sab=True, render_mode="rgb_array")
124         obs, info = env.reset()
125         done = False
126         frames = []
127         print(f"\n--- Test Episode {episode_num} ---")
128         print("Initial Observation:", obs)
129         while not done:
130             frames.append(env.render())
131             action = agent.get_action(obs)
132             print(f"Agent action: {'Hit' if action == 1 else 'Stick'}, Obs: {obs}")
133             obs, reward, terminated, truncated, info = env.step(action)
134             done = terminated or truncated
135         print("Final Reward:", reward)
136
137         fig = plt.figure(figsize=(6, 4))
138         img = plt.imshow(frames[0])
139         plt.axis("off")
140
141         def animate(i):
142             img.set_data(frames[i])
143             return [img]
144
145         anim = animation.FuncAnimation(fig, animate, frames=len(frames), interval=500, blit=True)
146         anim.save(f"blackjack_ep{episode_num}.gif", writer=PillowWriter(fps=2))
147         print(f"Saved GIF as blackjack_ep{episode_num}.gif")
148

```

```

149 Tabnine | Edit | Test | Explain | Document | ↺ Cody
150 def test_blackjack_statistics(n=1000):
151     try:
152         with open("blackjack_agent.pkl", "rb") as f:
153             agent = pickle.load(f)
154     except FileNotFoundError:
155         print("No trained agent found.")
156         return
157
158     agent.epsilon = 0.0
159     env = gym.make("Blackjack-v1", sab=True)
160     wins, draws, losses = 0, 0, 0
161
162     for _ in range(n):
163         obs, info = env.reset()
164         done = False
165         while not done:
166             action = agent.get_action(obs)
167             obs, reward, terminated, truncated, info = env.step(action)
168             done = terminated or truncated
169             if reward == 1.0:
170                 wins += 1
171             elif reward == 0.0:
172                 draws += 1
173             else:
174                 losses += 1
175
176     print(f"Wins: {wins}, Draws: {draws}, Losses: {losses}")

```

```

177 Tabnine | Edit | Test | Explain | Document | ↺ Cody
178 def test_blackjack():
179     print("\nBlackjack Testing Options:")
180     print("1. Run and visualize one episode")
181     print("2. Run and visualize multiple episodes")
182     print("3. Evaluate statistics over 1000 episodes")
183     choice = input("Enter choice (1, 2, or 3): ").strip()
184
185     if choice == "1":
186         test_blackjack_one_episode()
187     elif choice == "2":
188         test_blackjack_multiple_episodes()
189     elif choice == "3":
190         test_blackjack_statistics()
191     else:
192         print("Invalid choice.")

```


GAME 3-- BLACKJACK --

```
PS D:\AI game\aigame> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 1
```

```
Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 3
```

Blackjack Game selected.

```
Starting training...  
100% |████████████████████████████████████████████████████████████████████████████████| 1000000/1000000 [03:14<00:00, 5154.01it/s]  
Training complete and agent saved.
```

```
PS D:\AI game\ai game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2
```

```
Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 3
```

Blackjack Game selected.

```
Starting testing...
```

```
Blackjack Testing Options:
1. Run and visualize one episode
2. Run and visualize multiple episodes
3. Evaluate statistics over 1000 episodes
Enter choice (1, 2, or 3): 1
Final Reward: 1.0
```

```
Animation saved as blackjack_agent.gif
PS D:\AI game\ai game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2
```

```
Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 3
```

Blackjack Game selected.

Starting testing...

Blackjack Testing Options:

1. Run and visualize one episode
2. Run and visualize multiple episodes
3. Evaluate statistics over 1000 episodes

Enter choice (1, 2, or 3): 2

--- Test Episode 1 ---

Initial Observation: (21, 10, 1)

Agent action: Stick, Obs: (21, 10, 1)

Final Reward: 1.0

Saved GIF as blackjack_ep1.gif

--- Test Episode 2 ---

Initial Observation: (7, 6, 0)

Agent action: Hit, Obs: (7, 6, 0)

Agent action: Hit, Obs: (11, 6, 0)

Agent action: Hit, Obs: (12, 6, 0)

Final Reward: -1.0

Saved GIF as blackjack_ep2.gif

--- Test Episode 3 ---

Initial Observation: (11, 8, 0)

Agent action: Hit, Obs: (11, 8, 0)

Agent action: Hit, Obs: (12, 8, 0)

Agent action: Stick, Obs: (21, 8, 0)

Final Reward: 1.0

Saved GIF as blackjack_ep3.gif

--- Test Episode 4 ---

Initial Observation: (11, 4, 0)

Agent action: Hit, Obs: (11, 4, 0)

Agent action: Stick, Obs: (21, 4, 0)

Final Reward: 1.0

Saved GIF as blackjack_ep4.gif

--- Test Episode 5 ---

Initial Observation: (16, 1, 0)

Agent action: Hit, Obs: (16, 1, 0)

Final Reward: -1.0

Saved GIF as blackjack_ep5.gif

PS D:\AI game\aigame> **python** main.py

Welcome to the Game Trainer & Tester!

Select an operation:

1. Train a game
2. Test a game

Enter 1 or 2: 2

Select the game:

1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar

Enter game choice (1, 2, 3 or 4): 3

Blackjack Game selected.

Starting testing...

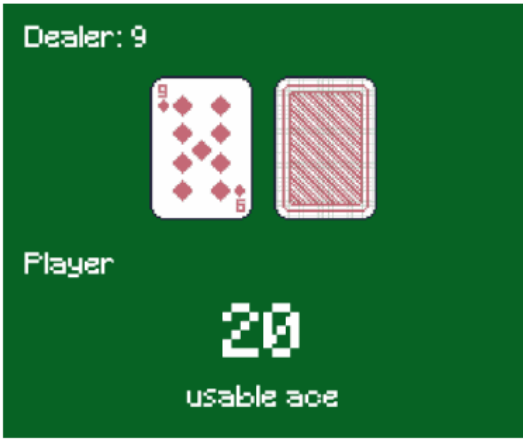
Blackjack Testing Options:

1. Run and visualize one episode
2. Run and visualize multiple episodes
3. Evaluate statistics over 1000 episodes

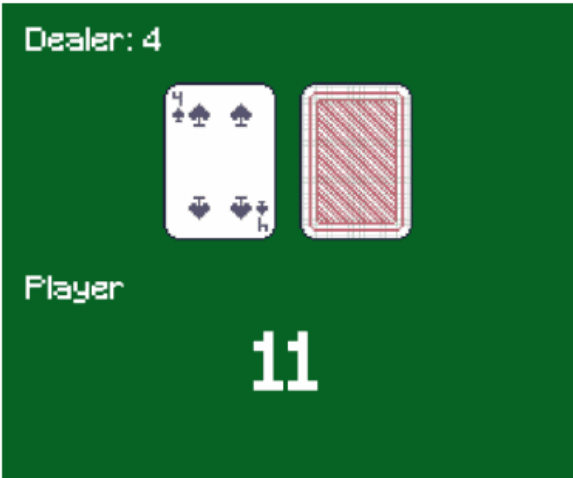
Enter choice (1, 2, or 3): 3

Wins: 432, Draws: 71, Losses: 497

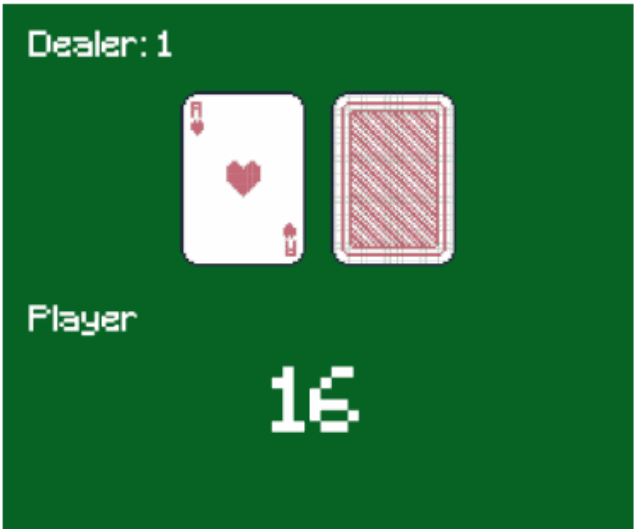
RESULT-- FOR 1 EPISODE



FOR MORE EPISODE—
EPISODE 4



EPISODE 5



```

10 from matplotlib import animation
11 import os
12 import matplotlib.pyplot as plt
13 import gymnasium as gym
14 import numpy as np
15 import matplotlib.pyplot as plt
16 from collections import deque
17 import random
18 import torch
19 from torch import nn
20 import torch.nn.functional as F
21 Tabnine | Edit | Test | Explain | Document | Cody
22 def save_frames_as_gif(frames, filename='mountaincar_test.gif', path='./mountaincar_gifs/'):
23     if not os.path.exists(path):
24         os.makedirs(path)
25
26     plt.figure(figsize=(frames[0].shape[1]/72.0, frames[0].shape[0]/72.0), dpi=72)
27     patch = plt.imshow(frames[0])
28     plt.axis('off')
29
30     def animate(i):
31         patch.set_data(frames[i])
32
33     anim = animation.FuncAnimation(plt.gcf(), animate, frames=len(frames), interval=50)
34     anim.save(os.path.join(path, filename), writer='pillow')
35     plt.close()
36
37 # Define model
38 class DQN(nn.Module):
39     Tabnine | Edit | Test | Explain | Document | Cody
40     def __init__(self, in_states, h1_nodes, out_actions):
41         super().__init__()
42
43         # Define network layers
44         self.fc1 = nn.Linear(in_states, h1_nodes) # first fully connected layer
45         self.out = nn.Linear(h1_nodes, out_actions) # output layer w
46
47     Tabnine | Edit | Test | Explain | Document | Cody
48     def forward(self, x):
49         x = F.relu(self.fc1(x)) # Apply rectified linear unit (ReLU) activation
50         x = self.out(x) # Calculate output
51         return x
52
53 # Define memory for Experience Replay
54 class ReplayMemory():
55     Tabnine | Edit | Test | Explain | Document | Cody
56     def __init__(self, maxlen):
57         self.memory = deque([], maxlen=maxlen)
58
59     Tabnine | Edit | Test | Explain | Document | Cody
60     def append(self, transition):
61         self.memory.append(transition)
62
63     Tabnine | Edit | Test | Explain | Document | Cody
64     def sample(self, sample_size):
65         return random.sample(self.memory, sample_size)
66
67     Tabnine | Edit | Test | Explain | Document | Cody
68     def __len__(self):
69         return len(self.memory)
70
71 # MountainCar Deep Q-Learning
72 class MountainCarDQL():
73     # Hyperparameters (adjustable)
74     learning_rate_a = 0.01 # Learning rate (alpha)
75     discount_factor_g = 0.9 # discount rate (gamma)
76     network_sync_rate = 50000 # number of steps the agent takes before syncing the policy and target network
77     replay_memory_size = 100000 # size of replay memory
78     mini_batch_size = 32 # size of the training data set sampled from the replay memory
79
80     num_divisions = 20
81
82     # Neural Network
83     loss_fn = nn.MSELoss() # NN Loss function. MSE=Mean Squared Error can be swapped to something else.
84     optimizer = None # NN Optimizer. Initialize Later.
85
86     # Train the environment
87     Tabnine | Edit | Test | Explain | Document | Cody
88     def train(self, episodes, render=False):
89         # Create FrozenLake instance
90         env = gym.make('MountainCar-v0', render_mode='human' if render else None)
91         num_states = env.observation_space.shape[0] # expecting 2: position & velocity
92         num_actions = env.action_space.n
93
94         # Divide position and velocity into segments
95         self.pos_space = np.linspace(env.observation_space.low[0], env.observation_space.high[0], self.num_divisions) # Between -1.2 and 0.6
96         self.vel_space = np.linspace(env.observation_space.low[1], env.observation_space.high[1], self.num_divisions) # Between -0.07 and 0.07
97
98         epsilon = 1 # 1 = 100% random actions
99         memory = ReplayMemory(self.replay_memory_size)
100
101         # Create policy and target network. Number of nodes in the hidden layer can be adjusted.
102         policy_dqn = DQN(in_states=num_states, h1_nodes=10, out_actions=num_actions)
103         target_dqn = DQN(in_states=num_states, h1_nodes=10, out_actions=num_actions)
104
105         # Make the target and policy networks the same (copy weights/biases from one network to the other)
106         target_dqn.load_state_dict(policy_dqn.state_dict())
107
108         # Policy network optimizer. "Adam" optimizer can be swapped to something else.
109         self.optimizer = torch.optim.Adam(policy_dqn.parameters(), lr=self.learning_rate_a)

```

```

# List to keep track of rewards collected per episode. Initialize List to 0's.
rewards_per_episode = []

# List to keep track of epsilon decay
epsilon_history = []

# Track number of steps taken. Used for syncing policy => target network.
step_count=0
goal_reached=False
best_rewards=-200

for i in range(episodes):
    state = env.reset()[0] # Initialize to state 0
    terminated = False     # True when agent falls in hole or reached goal

    rewards = 0

    # Agent navigates map until it falls into hole/reaches goal (terminated), or has taken 200 actions (truncated).
    while(not terminated and rewards>-1000):
        # Select action based on epsilon-greedy
        if random.random() < epsilon:
            # select random action
            action = env.action_space.sample() # actions: 0=left,1=idle,2=right
        else:
            # select best action
            with torch.no_grad():
                action = policy_dqn(self.state_to_dqn_input(state)).argmax().item()

        # Execute action
        new_state,reward,terminated,truncated,_ = env.step(action)

        # Accumulate reward
        rewards += reward

        # Save experience into memory
        memory.append((state, action, new_state, reward, terminated))

        # Move to the next state
        state = new_state

        # Increment step counter
        step_count+=1

    # Keep track of the rewards collected per episode.
    rewards_per_episode.append(rewards)
    if(terminated):
        goal_reached = True

    # Graph training progress
    if(i!=0 and i%1000==0):
        print(f'Episode {i} Epsilon {epsilon}')

        # self.plot_progress(rewards_per_episode, epsilon_history)

    if rewards>best_rewards:
        best_rewards = rewards
        print(f'Best rewards so far: {best_rewards}')
        # Save policy
        torch.save(policy_dqn.state_dict(), f"mountaincar_dqn_{i}.pt")
        torch.save(policy_dqn.state_dict(), "best_model.pt")

    # Check if enough experience has been collected
    if len(memory)>self.mini_batch_size and goal_reached:
        mini_batch = memory.sample(self.mini_batch_size)
        self.optimize(mini_batch, policy_dqn, target_dqn)

        # Decay epsilon
        epsilon = max(epsilon - 1/episodes, 0)
        epsilon_history.append(epsilon)

        # Copy policy network to target network after a certain number of steps
        if step_count > self.network_sync_rate:
            target_dqn.load_state_dict(policy_dqn.state_dict())
            step_count=0

# Close environment
env.close()

```


optimize policy network

Tabnine | Edit | Test | Explain | Document | Cody

```
def optimize(self, mini_batch, policy_dqn, target_dqn):
```

```
    current_q_list = []
    target_q_list = []

    for state, action, new_state, reward, terminated in mini_batch:

        if terminated:
            # Agent receive reward of 0 for reaching goal.
            # When in a terminated state, target q value should be set to the reward.
            target = torch.FloatTensor([reward])
        else:
            # Calculate target q value
            with torch.no_grad():
                target = torch.FloatTensor(
                    reward + self.discount_factor_g * target_dqn(self.state_to_dqn_input(new_state)).max()
                )

            # Get the current set of Q values
            current_q = policy_dqn(self.state_to_dqn_input(state))
            current_q_list.append(current_q)

            # Get the target set of Q values
            target_q = target_dqn(self.state_to_dqn_input(state))
            # Adjust the specific action to the target that was just calculated
            target_q[action] = target
            target_q_list.append(target_q)

    # Compute loss for the whole minibatch
    loss = self.loss_fn(torch.stack(current_q_list), torch.stack(target_q_list))

    # Optimize the model
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

Cody

```
...

Converts a state (position, velocity) to tensor representation.
Example:
Input = (0.3, -0.03)
Return = tensor([16, 6])
...
```

Tabnine | Edit | Test | Explain | Document | Cody

```
def state_to_dqn_input(self, state)->torch.Tensor:
    state_p = np.digitize(state[0], self.pos_space)
    state_v = np.digitize(state[1], self.vel_space)

    return torch.FloatTensor([state_p, state_v])
```

Run the environment with the learned policy

Tabnine | Edit | Test | Explain | Document | Cody

```
def test(self, episodes, model_filepath):
    # Create FrozenLake instance
    # env = gym.make('MountainCar-v0', render_mode='human')
    env = gym.make('MountainCar-v0', render_mode='rgb_array')
    num_states = env.observation_space.shape[0]
    num_actions = env.action_space.n

    self.pos_space = np.linspace(env.observation_space.low[0], env.observation_space.high[0], self.num_divisions) # Between -1.2 and 0.6
    self.vel_space = np.linspace(env.observation_space.low[1], env.observation_space.high[1], self.num_divisions) # Between -0.07 and 0.07


    # Load Learned policy
    policy_dqn = DQN(in_states=num_states, h1_nodes=10, out_actions=num_actions)
    # policy_dqn.load_state_dict(torch.load(model_filepath))
    policy_dqn.load_state_dict(torch.load(model_filepath, weights_only=True))
    policy_dqn.eval() # switch model to evaluation mode
    # Add this before the loop
    for i in range(episodes):
        frames = []
        state = env.reset()[0] # Initialize to state 0
        terminated = False # True when agent falls in hole or reached goal
        truncated = False # True when agent takes more than 200 actions

        # Agent navigates map until it falls into a hole (terminated), reaches goal (terminated), or has taken 200 actions (truncated).
        while(not terminated and not truncated):
            # Select best action
            frame = env.render()
            frames.append(frame)


            # if hasattr(env, 'render'):
            #     frame = env.render() # Get a rendered frame
            #     if frame is not None:
            #         frames.append(frame)
            with torch.no_grad():
                action = policy_dqn(self.state_to_dqn_input(state)).argmax().item()

            # Execute action
            state, reward, terminated, truncated, _ = env.step(action)
        if len(frames) > 0:
            save_frames_as_gif(frames, filename=f"mountaincar_test_episode_{i}.gif")

    env.close()
```

Tabnine | Edit | Test | Explain | Document |  Cody

```
def train_mountaincar():  
    mountaincar = MountainCarDQL()  
    mountaincar.train(20000, False)
```

Tabnine | Edit | Test | Explain | Document |  Cody

```
def test_mountaincar():  
    mountaincar = MountainCarDQL()  
    mountaincar.test(10, "best_model.pt")
```

```

PS D:\AI game\ai game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 1

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 4

MountainCar Game selected.

Starting training...
Episode 1000 Epsilon 1
Episode 2000 Epsilon 1
Episode 3000 Epsilon 0.95860000000000046
Episode 4000 Epsilon 0.90860000000000101
Episode 5000 Epsilon 0.85860000000000156
Episode 6000 Epsilon 0.80860000000000211
Episode 7000 Epsilon 0.75860000000000266
Episode 8000 Epsilon 0.70860000000000321
Episode 9000 Epsilon 0.65860000000000376
Episode 10000 Epsilon 0.60860000000000431
Episode 11000 Epsilon 0.55860000000000486
Best rewards so far: -194.0
Best rewards so far: -176.0
Episode 12000 Epsilon 0.50860000000000541
Best rewards so far: -175.0
Best rewards so far: -169.0
Episode 13000 Epsilon 0.45860000000000596
Episode 14000 Epsilon 0.408600000000006513
Best rewards so far: -164.0
Best rewards so far: -162.0
Episode 15000 Epsilon 0.358600000000007064
Episode 16000 Epsilon 0.308600000000007615
Best rewards so far: -144.0
Episode 17000 Epsilon 0.258600000000008165
Episode 18000 Epsilon 0.208600000000008716
Episode 19000 Epsilon 0.158600000000009267
PS D:\AI game\ai game> python main.py
Welcome to the Game Trainer & Tester!
Select an operation:
1. Train a game
2. Test a game
Enter 1 or 2: 2

Select the game:
1. Cliff Climbing
2. Taxi
3. Blackjack
4. MountainCar
Enter game choice (1, 2, 3 or 4): 4

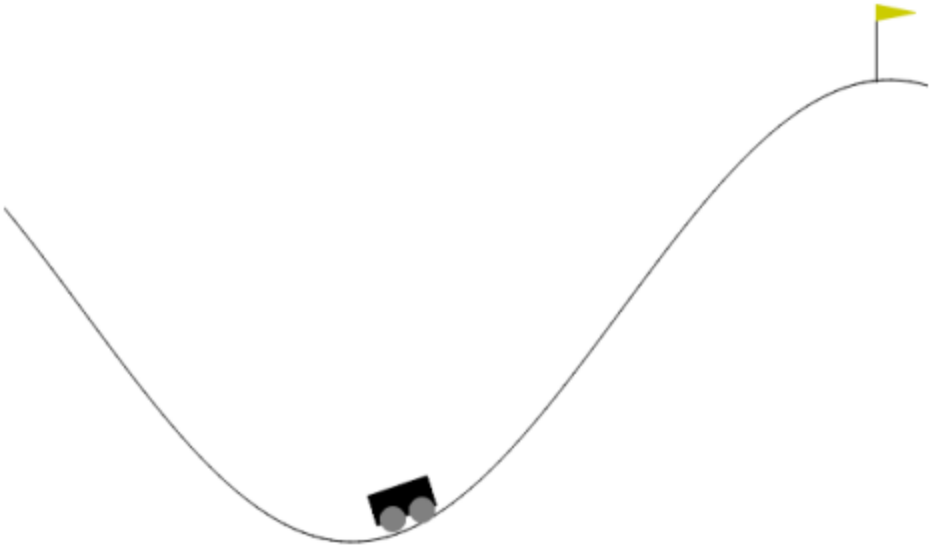
MountainCar Game selected.

Starting testing...
PS D:\AI game\ai game>

```

RESULT-

EPOSODE -8



EPISODE - 9

