In [1]:

```
!pip install gym
```

...

In [2]:

```
import gym
import random
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
```

In [3]:

```
# Define Deep Q-Network (DQN) model
class DQN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

In [4]:

```
# Define replay buffer
class ReplayBuffer:
    def __init__(self, capacity):
        self.capacity = capacity
        self.buffer = []

    def add(self, state, action, reward, next_state, done):
        experience = (state, action, reward, next_state, done)
        self.buffer.append(experience)
        if len(self.buffer) > self.capacity:
            self.buffer.pop(0)

    def sample(self, batch_size):
        return random.sample(self.buffer, batch_size)
```

In [5]:

```
# Define the DQN Agent
class DQNAgent:
    def __init__(self, input_size, output_size, hidden_size=64, learning_ra
                 epsilon_start=1.0, epsilon_decay=0.995, epsilon_min=0.01,
                 batch_size=64):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size = hidden_size
        self.learning_rate = learning_rate
        self.gamma = gamma

        self.epsilon = epsilon_start
        self.epsilon_decay = epsilon_decay
        self.epsilon_min = epsilon_min
        self.replay_buffer = ReplayBuffer(replay_buffer_capacity)
        self.batch_size = batch_size

        self.q_network = DQN(input_size, hidden_size, output_size)
        self.target_network = DQN(input_size, hidden_size, output_size)
        self.target_network.load_state_dict(self.q_network.state_dict())
        self.optimizer = optim.Adam(self.q_network.parameters(), lr=learnin
        self.loss_function = nn.MSELoss()

    def epsilon_greedy_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(range(self.output_size))
        else:
            with torch.no_grad():
                q_values = self.q_network(torch.FloatTensor(state))
                return torch.argmax(q_values).item()

    def train(self, state, action, reward, next_state, done):
        self.replay_buffer.add(state, action, reward, next_state, done)
        if len(self.replay_buffer.buffer) > self.batch_size:
            batch = self.replay_buffer.sample(self.batch_size)
            states, actions, rewards, next_states, dones = zip(*batch)

            states = torch.FloatTensor(states)
            actions = torch.LongTensor(actions)
            rewards = torch.FloatTensor(rewards)
            next_states = torch.FloatTensor(next_states)
            dones = torch.FloatTensor(dones)

            q_values = self.q_network(states)
            next_q_values = self.target_network(next_states).max(1)[0]
            target_q_values = rewards + (1 - dones) * self.gamma * next_q_v

            q_values = q_values.gather(1, actions.unsqueeze(1)).squeeze(1)

            loss = self.loss_function(q_values, target_q_values.detach())

            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

        self.epsilon = max(self.epsilon * self.epsilon_decay, self.epsilon_

    def update_target_network(self):
        self.target_network.load_state_dict(self.q_network.state_dict())
```

In [6]:

```
# Initialize environment and agent
env = gym.make('CartPole-v1')
input_size = env.observation_space.shape[0]
output_size = env.action_space.n
agent = DQNAgent(input_size, output_size)
```

In [7]:

```
# Training
num_episodes = 1000
for episode in range(num_episodes):
    state = env.reset()
    if isinstance(state, tuple):  # For Gymnasium compatibility
        state = state[0]

    total_reward = 0
    done = False

    while not done:
        action = agent.epsilon_greedy_action(state)
        next_state, reward, done, truncated, _ = env.step(action) if hasatt
        if isinstance(next_state, tuple):  # For Gymnasium compatibility
            next_state = next_state[0]

        agent.train(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward

    if episode % 100 == 0:
        agent.update_target_network()
        print(f"Episode {episode}, Total Reward: {total_reward}")

# Close the environment
env.close()
```

```
C:\Python310\lib\site-packages\gym\utils\passive_env_checker.py:233: Depre
cationWarning: `np.bool8` is a deprecated alias for `np.bool_`.  (Deprecat
ed NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
C:\Users\dell\AppData\Local\Temp\ipykernel_19552\133867816.py:38: UserWarn
ing: Creating a tensor from a list of numpy.ndarrays is extremely slow. Pl
ease consider converting the list to a single numpy.ndarray with numpy.arr
ay() before converting to a tensor. (Triggered internally at C:\actions-ru
nner\_work\pytorch\pytorch\pytorch\torch\csrc\utils\tensor_new.cpp:257.)
  states = torch.FloatTensor(states)

Episode 0, Total Reward: 23.0
Episode 100, Total Reward: 15.0
Episode 200, Total Reward: 19.0
Episode 300, Total Reward: 30.0
Episode 400, Total Reward: 8.0
Episode 500, Total Reward: 38.0
Episode 600, Total Reward: 40.0
Episode 700, Total Reward: 241.0
Episode 800, Total Reward: 189.0
Episode 900, Total Reward: 232.0
```

```
In [ ]:
```