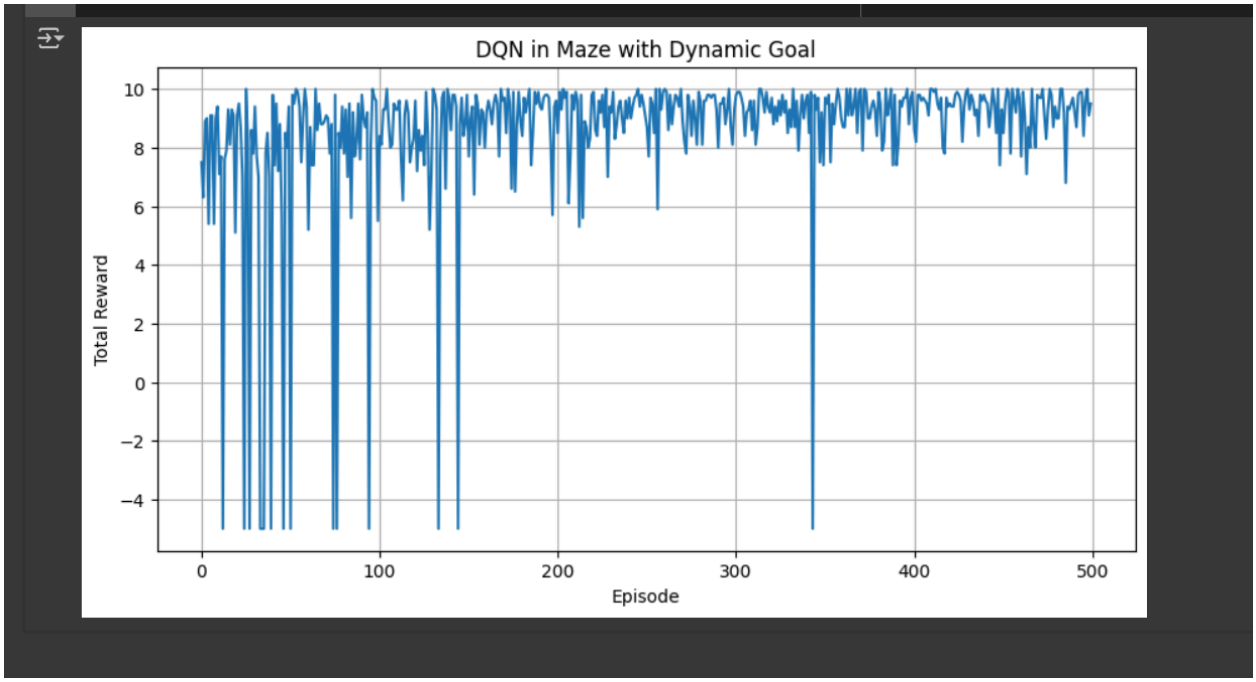
```python
import numpy as np
import random
import matplotlib.pyplot as plt
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim


np.random.seed(42)
random.seed(42)
torch.manual_seed(42)


GRID_SIZE = 5
WALL_PROB = 0.1
EPISODES = 500
MAX_STEPS = 50
ACTIONS = [(0, -1), (0, 1), (-1, 0), (1, 0)]

class DynamicGoalMaze:
    def __init__(self, size, wall_prob):
        self.size = size
        self.wall_prob = wall_prob
        self.reset()

    def random_empty_cell(self):
        while True:
            cell = (np.random.randint(0, self.size), np.random.randint(0, self.size))
            if self.grid[cell] == 0 and cell != self.agent_pos:
                return cell
```

```python
    def reset(self):
        self.grid = np.zeros((self.size, self.size), dtype=int)
        for i in range(self.size):
            for j in range(self.size):
                if random.random() < self.wall_prob:
                    self.grid[i][j] = 1
        self.agent_pos = (0, 0)
        self.agent_pos = self.random_empty_cell()
        self.goal = self.random_empty_cell()
        return self.get_state()


    def get_state(self):
        return np.array([*self.agent_pos, *self.goal], dtype=np.float32) / self.size

    def step(self, action_idx):
        dx, dy = ACTIONS[action_idx]
        x, y = self.agent_pos[0] + dx, self.agent_pos[1] + dy
        if 0 <= x < self.size and 0 <= y < self.size and self.grid[x][y] == 0:
            self.agent_pos = (x, y)
        reward = 10 if self.agent_pos == self.goal else -0.1
        done = self.agent_pos == self.goal
        self.goal = self.random_empty_cell()
        return self.get_state(), reward, done
```

```python
class DQN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(4, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 4)
        )

    def forward(self, x):
        return self.net(x)


class ReplayBuffer:
    def __init__(self, maxlen=10000):
        self.buffer = deque(maxlen=maxlen)

    def push(self, *transition):
        self.buffer.append(transition)

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        return zip(*batch)

    def __len__(self):
        return len(self.buffer)
```

```python
env = DynamicGoalMaze(GRID_SIZE, WALL_PROB)
dqn = DQN()
target_dqn = DQN()
target_dqn.load_state_dict(dqn.state_dict())
optimizer = optim.Adam(dqn.parameters(), lr=1e-3)
replay = ReplayBuffer()

gamma = 0.99
epsilon = 1.0
epsilon_min = 0.1
epsilon_decay = 0.995
batch_size = 64
episode_rewards = []

for ep in range(EPISODES):
    state = env.reset()
    total_reward = 0
    for step in range(MAX_STEPS):
        if random.random() < epsilon:
            action = random.randint(0, 3)
        else:
            with torch.no_grad():
                q_values = dqn(torch.tensor(state).float())
                action = torch.argmax(q_values).item()

        next_state, reward, done = env.step(action)
        replay.push(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward
```

```
        if len(replay) >= batch_size:
            s, a, r, s2, d = replay.sample(batch_size)
            s = torch.tensor(np.array(list(s))).float()
            a = torch.tensor(list(a)).long().unsqueeze(1)
            r = torch.tensor(list(r)).float().unsqueeze(1)
            s2 = torch.tensor(np.array(list(s2))).float()
            d = torch.tensor(list(d)).float().unsqueeze(1)

            q_val = dqn(s).gather(1, a)
            max_q = target_dqn(s2).max(1)[0].unsqueeze(1)
            target = r + gamma * max_q * (1 - d)

            loss = nn.MSELoss()(q_val, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if done:
            break

    epsilon = max(epsilon_min, epsilon * epsilon_decay)
    episode_rewards.append(total_reward)

    if ep % 10 == 0:
        target_dqn.load_state_dict(dqn.state_dict())

plt.figure(figsize=(10, 5))
plt.plot(episode_rewards)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("DQN in Maze with Dynamic Goal")
plt.grid(True)
plt.show()
```


DQN in Maze with Dynamic Goal
Total Reward
Episode
10
8
6
4
2
0
−2
−4
0
100
200
300
400
500