МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Обучение с подкреплением»

Тема: Реализация DQN для среды CartPole-v1

Студент гр. 0310 _____ Корсунов А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2025

Цель работы.

Реализация алгоритма DQN для среды CartPole-v1.

Задание.

1. Реализовать алгоритм DQN для среды CartPole-v1;
2. Изменить архитектуру нейросети;
3. Попробовать разные значения gamma и epsilon_decay;
4. Провести исследование как изначальное значение epsilon влияет на скорость обучения

Выполнение работы.

1. Алгоритм DQN реализован на ЯП Python с использованием библиотеки TenserFlow.

Код программы находится в Приложении А.

2. Были рассмотрены три архитектуры (рис. 1 – 4):

- **small**

(

nn.Linear(4, 64),

nn.ReLU(),

nn.Linear(64, 2)

)

- **medium**

(

nn.Linear(4, 128),

nn.ReLU(),

nn.Linear(128, 64),

nn.ReLU(),

nn.Linear(64, 2)

)

- **large**

(

nn.Linear(obs_size, 256),

nn.ReLU(),

nn.Linear(256, 128),

nn.ReLU(),

nn.Linear(128, 64),

nn.ReLU(),

nn.Linear(64, n_actions)

)

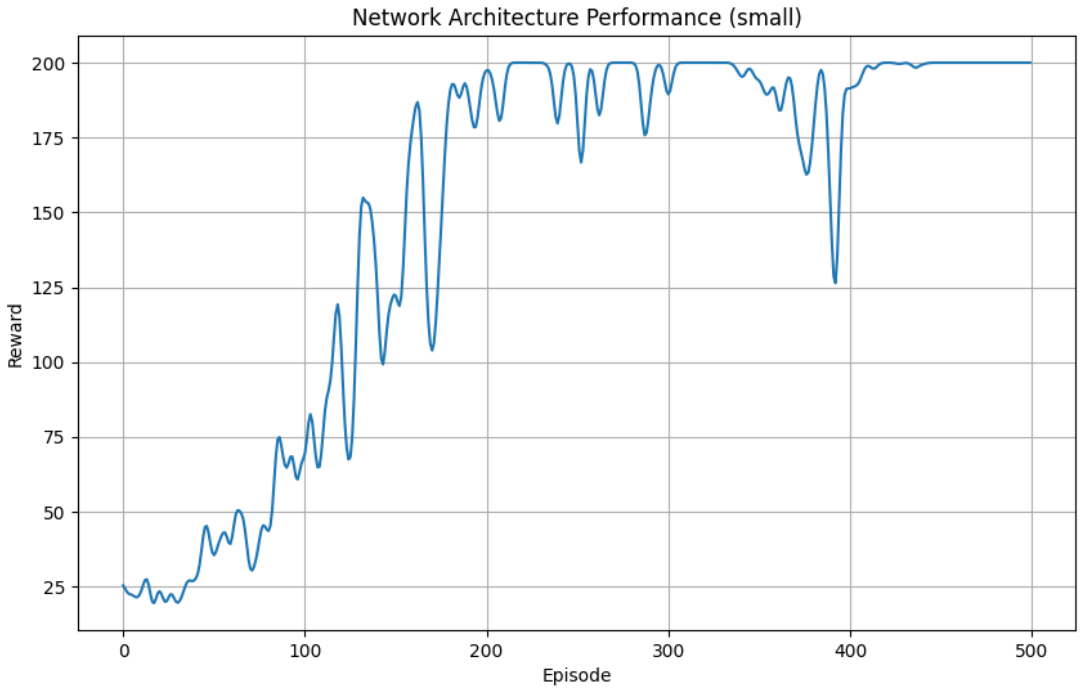

Рисунок 1 – График награды от номера эпизода архитектуры «small»

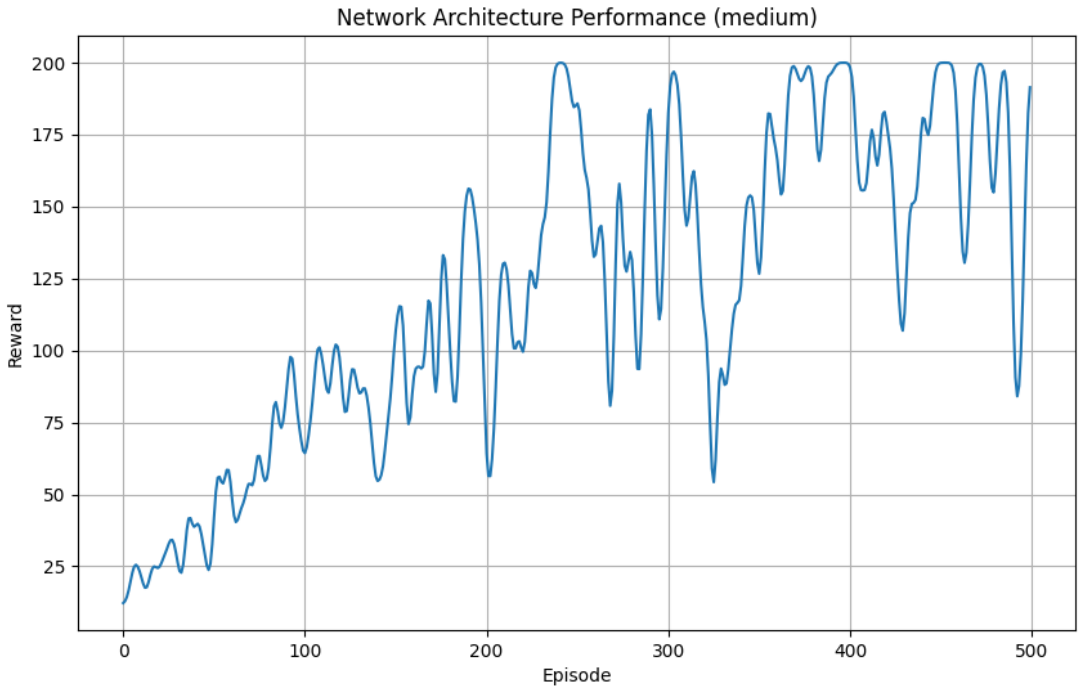

Рисунок 2 – График награды от номера эпизода архитектуры «medium»

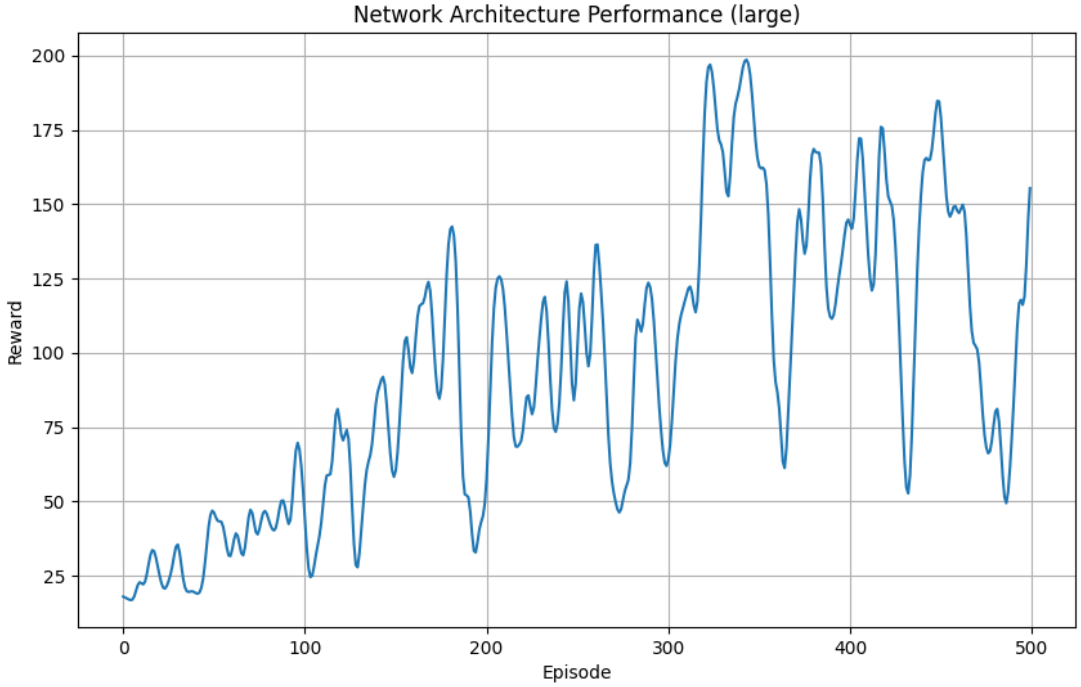

Рисунок 3 – График награды от номера эпизода архитектуры «large»

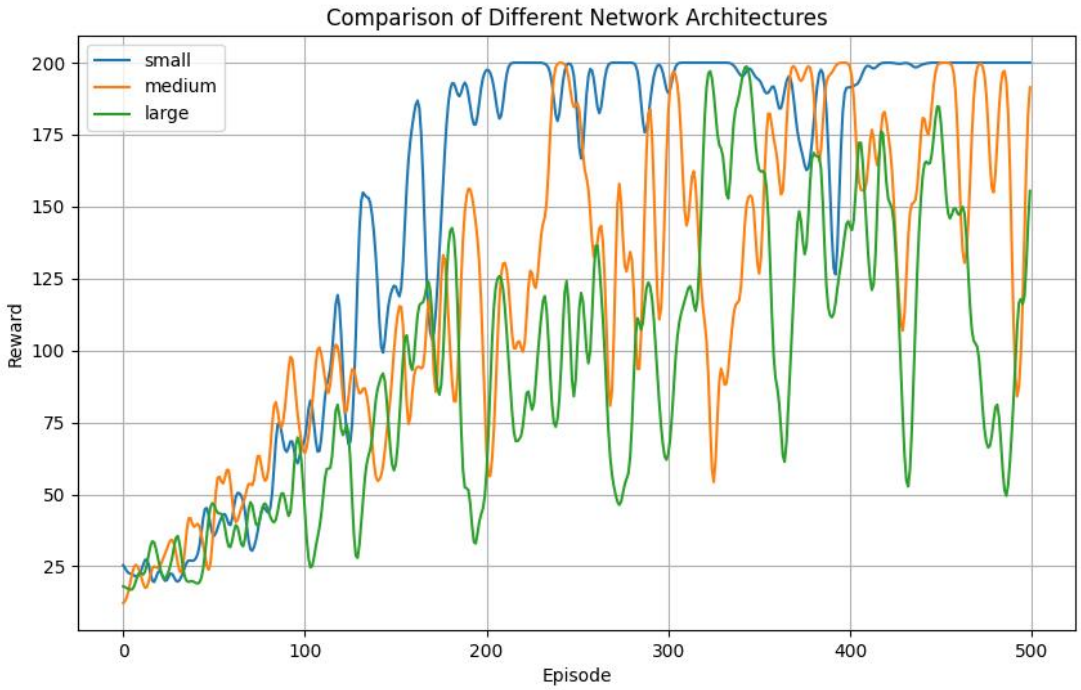

Рисунок 4 – Графики наград от номера эпизода для рассматриваемых в работе архитектур

На основе рисунков 1 – 4 можно сделать следующие выводы:

При архитектуре «small» сеть обучается быстрее всего – уже примерно к 200-у эпизоду наиболее стабильно достигает 200 очков. Другие сети обучаются медленнее и сильно колеблются, сеть на архитектуре «large» обучается медленнее всех и сильнее всех колеблется.

3. Были рассмотрены разные значения epsilon_decay при архитектуре «medium» при фиксированной gamma (рис. 5 – 8). Также были рассмотрены разные значения gamma при архитектуре «medium» при фиксированной epsilon_decay (рис. 9 – 12).

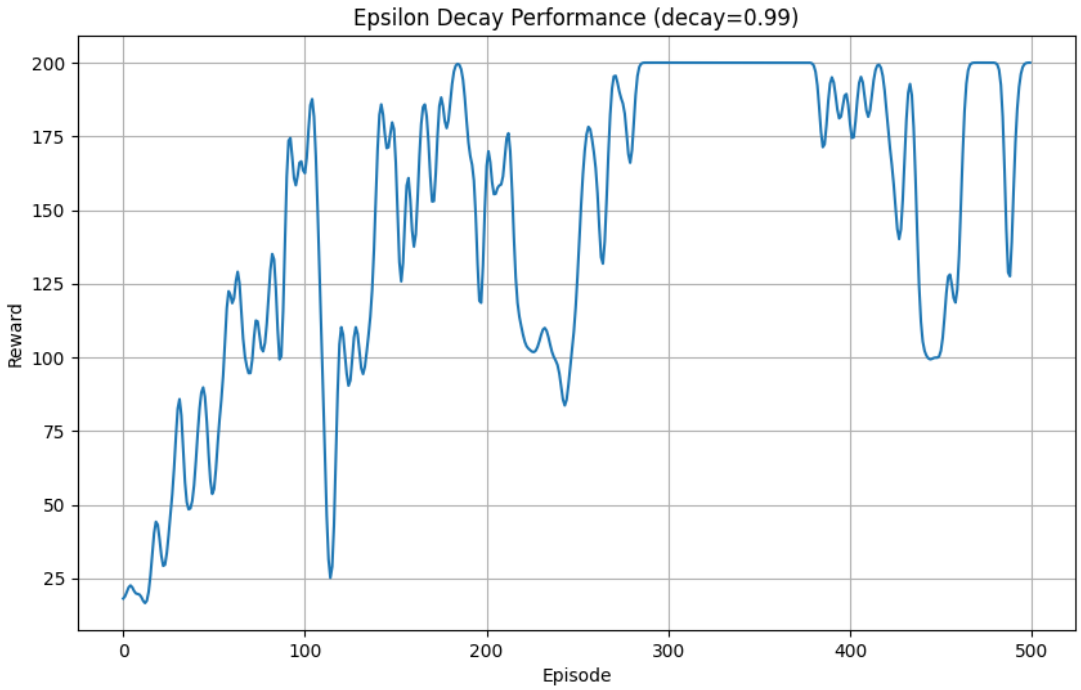

Рисунок 5 – График награды от номера эпизода при epsilon_decay = 0.99

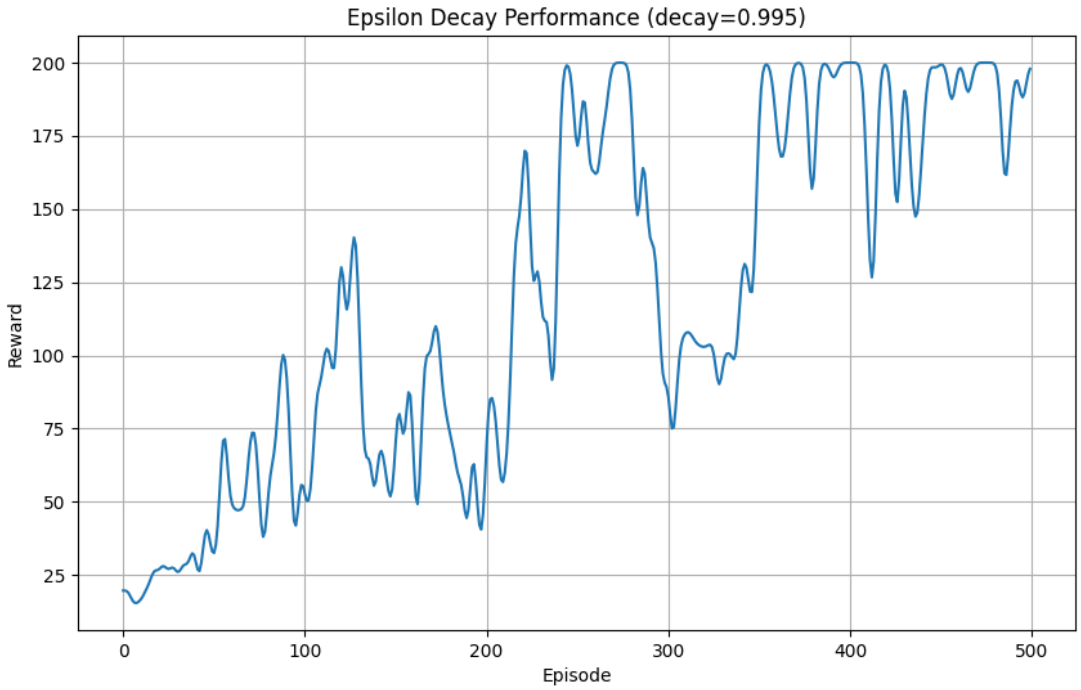

Рисунок 6 – График награды от номера эпизода при epsilon_decay = 0.995

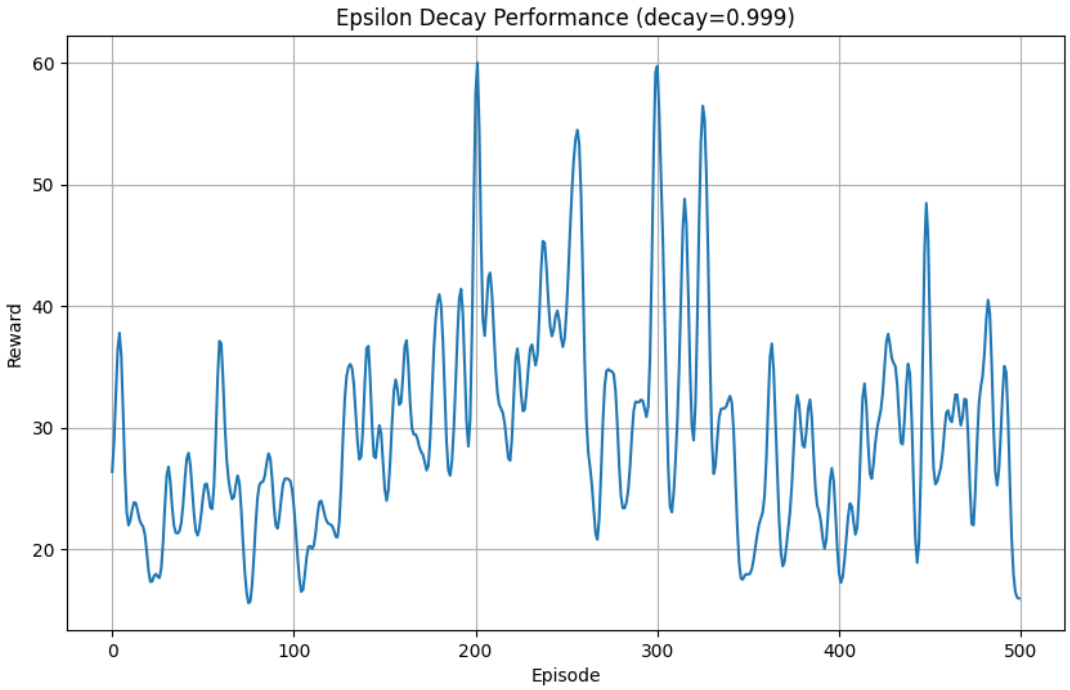

Рисунок 7 – График награды от номера эпизода при epsilon_decay = 0.999

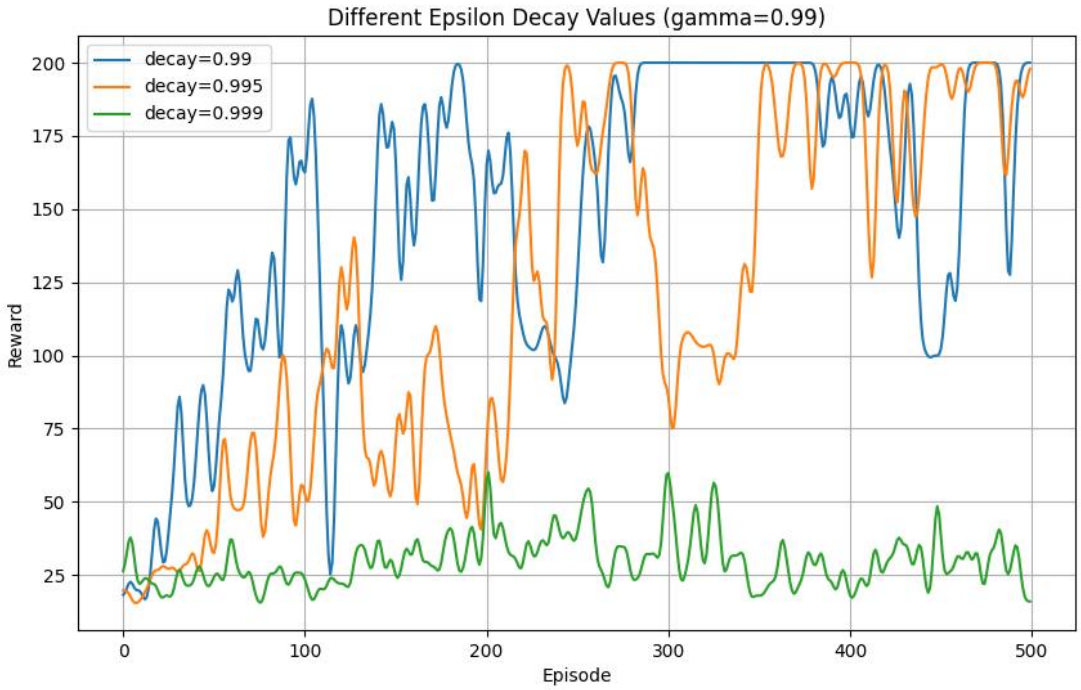

Рисунок 8 – Графики награды от номера эпизода для рассматриваемых в работе epsilon_decay

На основе рисунок 5 – 8 можно сделать следующие выводы:

При высоком значении epsilon_decay (0.999) агент долго обучается и при установленной в работе конфигурации не успевает достичь высоких наград. При низком значении epsilon_decay (0.99) агент обучается быстро, но застревает в локальном оптимуме. При epsilon_decay = 0.995 наблюдается наиболее сбалансированный результат из рассмотренных.

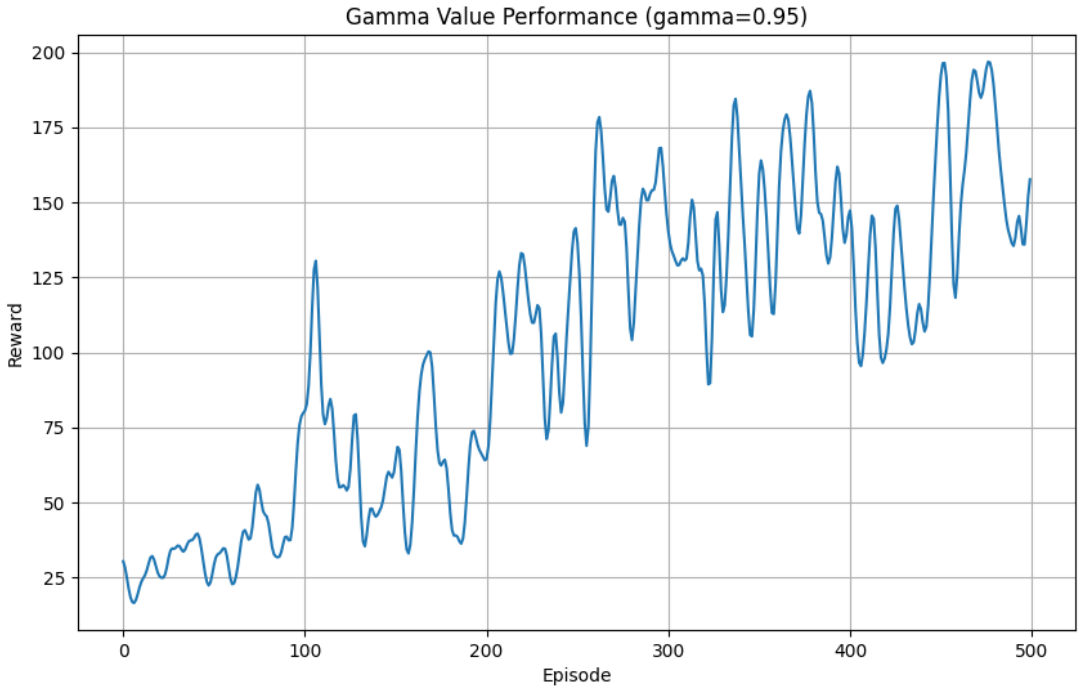

Рисунок 9 – График награды от номера эпизода при gamma = 0.95

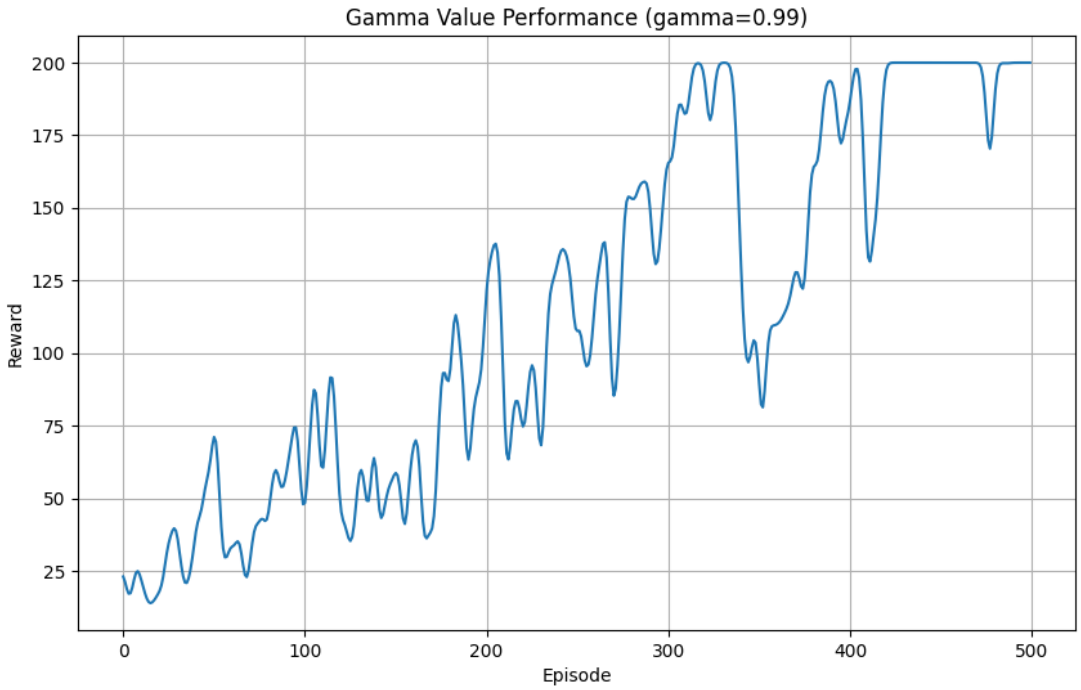

Рисунок 10 – График награды от номера эпизода при gamma = 0.99

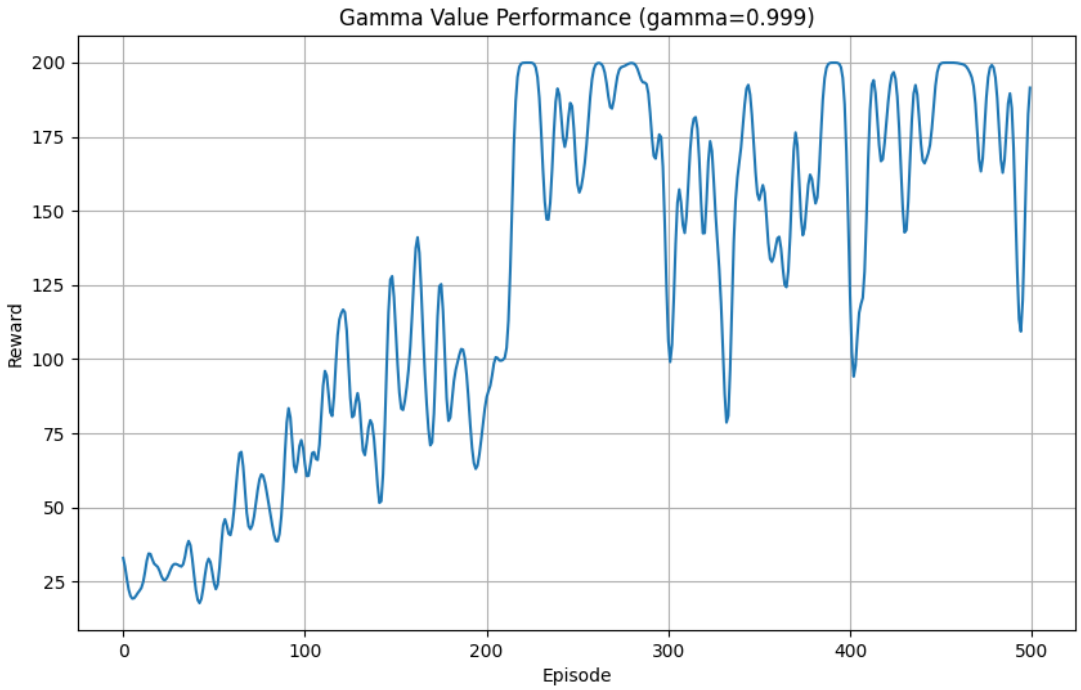

Рисунок 11 – График награды от номера эпизода при gamma = 0.999

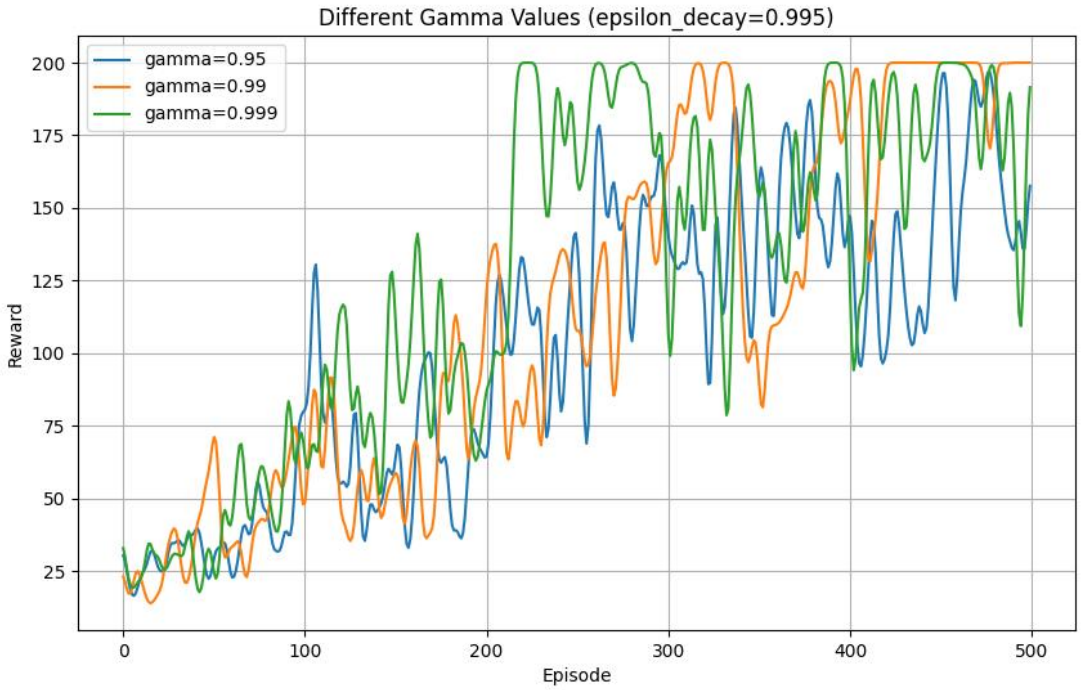

Рисунок 12 – Графики награды от номера эпизода для рассматриваемых в работе gamma

На основе рисунок 9 – 12 можно сделать следующие выводы:

При значении высоком значении gamma (0.999) агент достигает наивысшей награды быстрее всего. При низком значении (0.95) агент не успевает достичь наивысшей награды за 500 эпизодов. При значении gamma = 0.99 агент достигает наивысшей награды наиболее монотонно.

4. Были рассмотрены разные значения начальной epsilon при архитектуре «medium», gamma = 0.99, epsilon_decay = 0.995 (рис. 13 – 16).

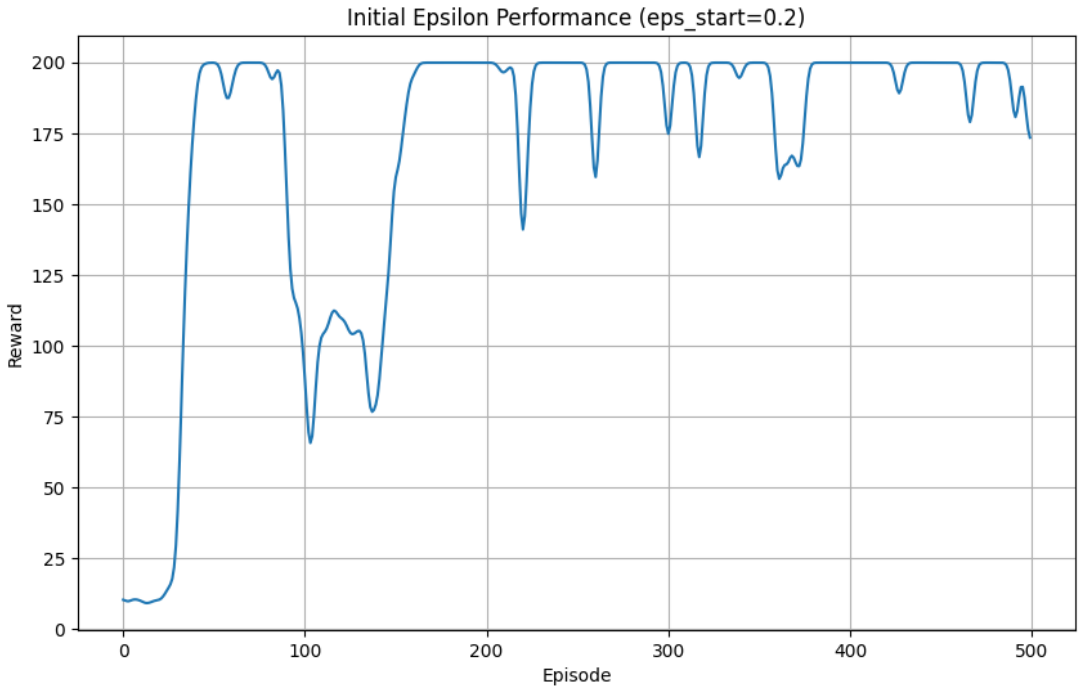

Рисунок 13 – График награды от номера эпизода при eps_start = 0.2

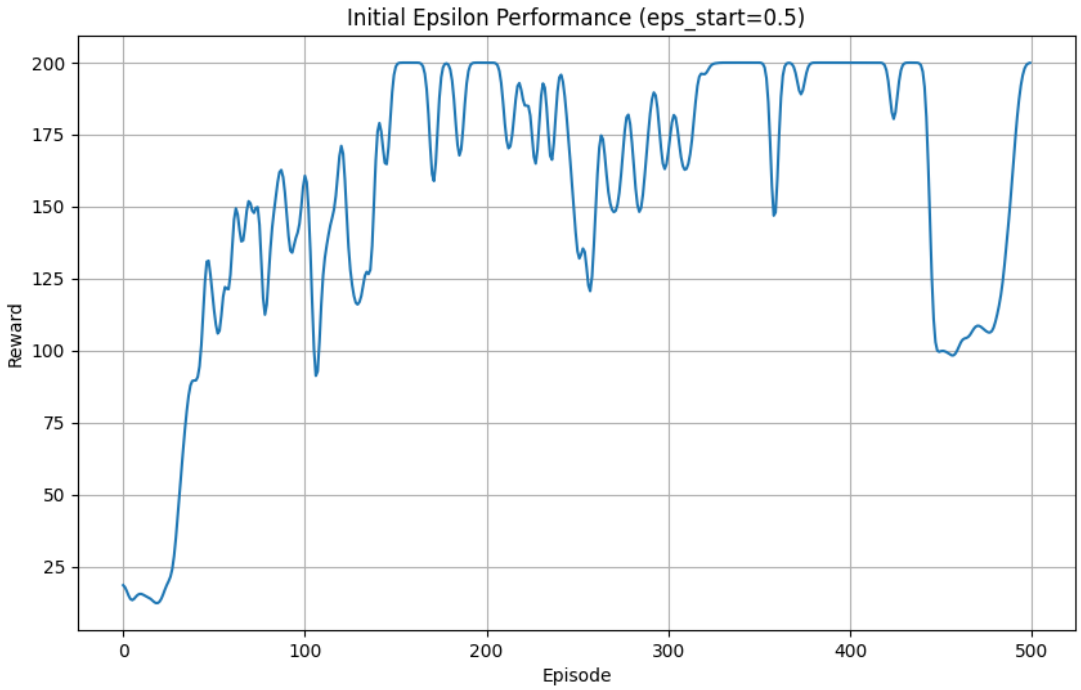

Рисунок 14 – График награды от номера эпизода при eps_start = 0.5

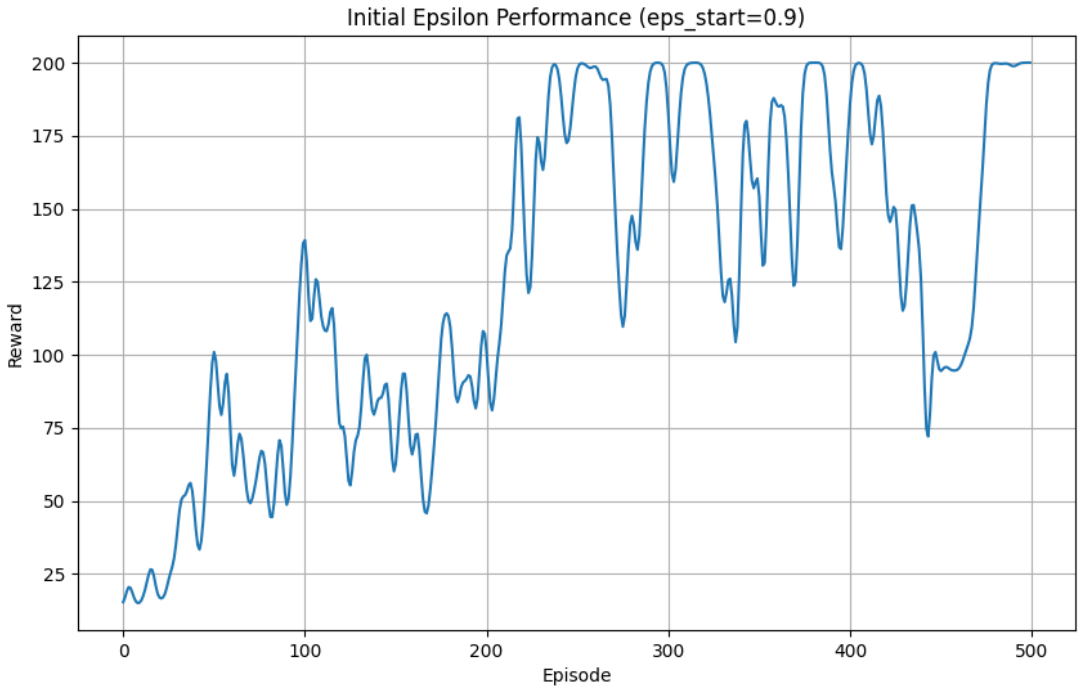

Рисунок 15 – График награды от номера эпизода при eps_start = 0.9

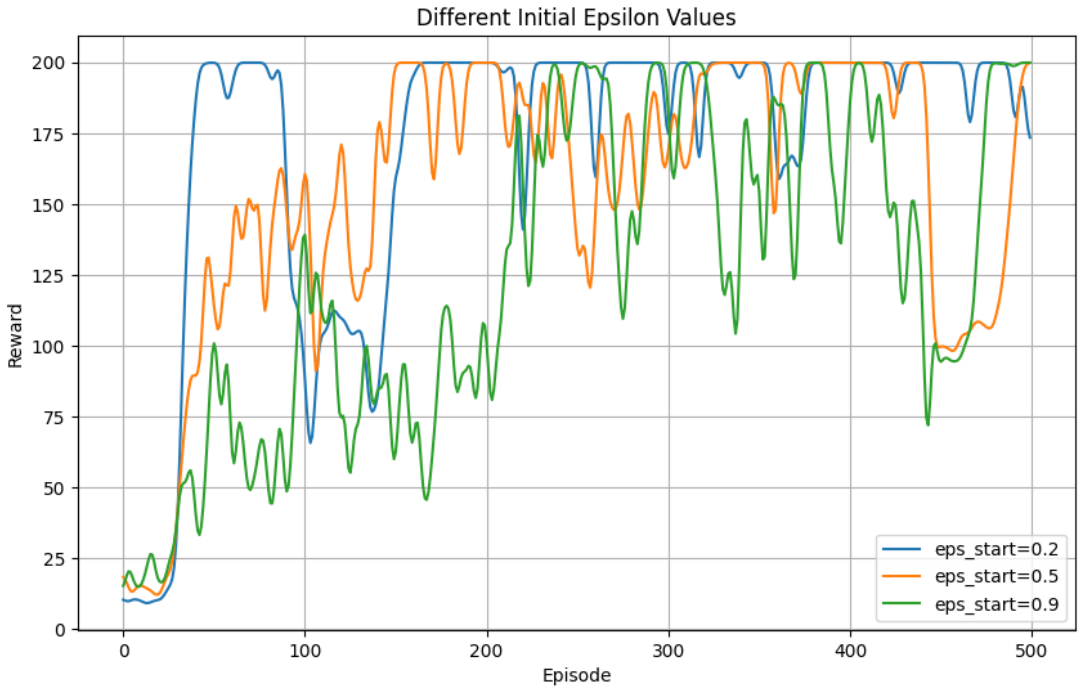

Рисунок 16 – Графики награды от номера эпизода для рассматриваемых в работе eps_start

На основе рисунок 13 – 16 можно сделать следующие выводы:

При низком значении eps_start (0.2) агент наиболее быстро достигает наивысшей награды, но часто застревает в локальных оптимумах. При высоком значении eps_start (0.9) агент наиболее медленно достигает наивысшей награды. То есть изначальное значение epsilon обратно пропорционально скорости обучения.

Выводы.

Реализован алгоритм DQN для среды CartPole-v1. Были проведены исследования при различных архитектурах и гиперпараметрах. Также была выявлена связь между начальным значением epsilon и скоростью обучения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ
MAIN.PY

```
import random
import gymnasium as gym
import torch
import numpy as np
import matplotlib.pyplot as plt
import os
from collections import deque
from torch import nn
from torch import optim
from tqdm import tqdm
from scipy.ndimage import gaussian_filter1d

os.makedirs("plots", exist_ok=True)

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

    def push(self, state, action, reward, next_state, done):
        self.buffer.append((state, action, reward, next_state,
done))

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        state, action, reward, next_state, done = zip(*batch)
        return (
            torch.tensor(np.array(state), dtype=torch.float32),
            torch.tensor(action, dtype=torch.int64),
            torch.tensor(reward, dtype=torch.float32),
            torch.tensor(np.array(next_state),
dtype=torch.float32),
            torch.tensor(done, dtype=torch.float32)
        )

    def __len__(self):
```

```
        return len(self.buffer)

class QNetworkSmall(nn.Module):
    def __init__(self, obs_size, n_actions):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(obs_size, 64),
            nn.ReLU(),
            nn.Linear(64, n_actions)
        )

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

class QNetworkMedium(nn.Module):
    def __init__(self, obs_size, n_actions):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(obs_size, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, n_actions)
        )

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

class QNetworkLarge(nn.Module):
    def __init__(self, obs_size, n_actions):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(obs_size, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
```

```
            nn.Linear(64, n_actions)
        )

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

class DQNAgent:
    def __init__(self, obs_size, n_actions, network_type='medium', gamma=0.99, epsilon_decay=0.995, epsilon_start=1.0):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        if network_type == 'small':
            self.q_net = QNetworkSmall(obs_size, n_actions).to(self.device)
            self.target_net = QNetworkSmall(obs_size, n_actions).to(self.device)
        elif network_type == 'large':
            self.q_net = QNetworkLarge(obs_size, n_actions).to(self.device)
            self.target_net = QNetworkLarge(obs_size, n_actions).to(self.device)
        else:
            self.q_net = QNetworkMedium(obs_size, n_actions).to(self.device)
            self.target_net = QNetworkMedium(obs_size, n_actions).to(self.device)

        self.target_net.load_state_dict(self.q_net.state_dict())
        self.optimizer = optim.Adam(self.q_net.parameters(), lr=1e-3)
        self.gamma = gamma
        self.batch_size = 128
        self.epsilon = epsilon_start
        self.epsilon_decay = epsilon_decay
        self.epsilon_min = 0.01
        self.replay_buffer = ReplayBuffer(10000)
        self.network_type = network_type
```

```
        def select_action(self, state):
            if random.random() < self.epsilon:
                return random.randint(0, 1)
            else:
                with torch.no_grad():
                    state_tensor = torch.tensor(state,
dtype=torch.float32, device=self.device)
                    q_values = self.q_net(state_tensor)
                    return torch.argmax(q_values).item()

        def train(self):
            if len(self.replay_buffer) < self.batch_size:
                return

            state, action, reward, next_state, done =
self.replay_buffer.sample(self.batch_size)
            state = state.to(self.device)
            action = action.to(self.device)
            reward = reward.to(self.device)
            next_state = next_state.to(self.device)
            done = done.to(self.device)

            q_values = self.q_net(state).gather(1,
action.unsqueeze(1)).squeeze(1)
            with torch.no_grad():
                next_q_values = self.target_net(next_state).max(1)[0]
                target_q_values = reward + self.gamma * next_q_values *
(1 - done)

            loss = nn.MSELoss()(q_values, target_q_values)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

        def update_target(self):
            self.target_net.load_state_dict(self.q_net.state_dict())
```

```
    def smooth(y, sigma=2):
        return gaussian_filter1d(y, sigma=sigma)

    def plot_individual_results(results, title, xlabel, ylabel,
filename, smooth_curve=True):
        plt.figure(figsize=(10, 6))
        for name, rewards in results.items():
            if smooth_curve:
                plt.plot(smooth(rewards), label=name)
            else:
                plt.plot(rewards, label=name)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title)
        plt.legend()
        plt.grid()
        plt.savefig(f"plots/{filename}.png")
        plt.close()

    def plot_separate_results(results, title_prefix, xlabel, ylabel,
smooth_curve=True):
        for name, rewards in results.items():
            plt.figure(figsize=(10, 6))
            if smooth_curve:
                plt.plot(smooth(rewards))
            else:
                plt.plot(rewards)
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
            plt.title(f"{title_prefix} ({name})")
            plt.grid()
            plt.savefig(f"plots/{title_prefix.lower().replace('      ',
'_')}_{name}.png")
            plt.close()

    def train_agent(params):
        env = gym.make("CartPole-v1")
        agent = DQNAgent(**params)
```

```
        rewards = []

        for episode in tqdm(range(500), desc=f"Training {params.get('network_type', 'medium')}"):
            state, _ = env.reset()
            episode_reward = 0
            done = False

            for _ in range(200):
                action = agent.select_action(state)
                next_state, reward, done, _, _ = env.step(action)
                agent.replay_buffer.push(state, action, reward, next_state, done)
                state = next_state
                episode_reward += reward
                agent.train()

                if done: break

            agent.update_target()
            agent.epsilon = max(agent.epsilon * agent.epsilon_decay, agent.epsilon_min)
            rewards.append(episode_reward)

        env.close()
        return rewards

    # 1. Эксперименты с архитектурами
    print("Running architecture experiments...")
    architectures = ['small', 'medium', 'large']
    arch_results = {}

    for arch in architectures:
        rewards = train_agent({
            'obs_size': 4,
            'n_actions': 2,
            'network_type': arch,
            'gamma': 0.99,
```

```
        'epsilon_decay': 0.995
    })
    arch_results[arch] = rewards

# Сохраняем общий график архитектур
plot_individual_results(
    arch_results,
    "Comparison of Different Network Architectures",
    "Episode",
    "Reward",
    "architectures_comparison"
)

# Сохраняем отдельные графики для каждой архитектуры
plot_separate_results(
    arch_results,
    "Network Architecture Performance",
    "Episode",
    "Reward"
)

# 2. Эксперименты с гиперпараметрами
print("\nRunning hyperparameter experiments...")

# 2.1. Разные epsilon_decay при gamma=0.99
decay_values = [0.99, 0.995, 0.999]
decay_results = {}

for decay in decay_values:
    rewards = train_agent({
        'obs_size': 4,
        'n_actions': 2,
        'gamma': 0.99,
        'epsilon_decay': decay
    })
    decay_results[f"decay={decay}"] = rewards

# Сохраняем общий график для epsilon_decay
```

```
plot_individual_results(
    decay_results,
    "Different Epsilon Decay Values (gamma=0.99)",
    "Episode",
    "Reward",
    "epsilon_decay_comparison"
)

# Сохраняем отдельные графики для каждого epsilon_decay
plot_separate_results(
    decay_results,
    "Epsilon Decay Performance",
    "Episode",
    "Reward"
)

# 2.2. Разные gamma при epsilon_decay=0.995
gamma_values = [0.95, 0.99, 0.999]
gamma_results = {}

for gamma in gamma_values:
    rewards = train_agent({
        'obs_size': 4,
        'n_actions': 2,
        'gamma': gamma,
        'epsilon_decay': 0.995
    })
    gamma_results[f"gamma={gamma}"] = rewards

# Сохраняем общий график для gamma
plot_individual_results(
    gamma_results,
    "Different Gamma Values (epsilon_decay=0.995)",
    "Episode",
    "Reward",
    "gamma_comparison"
)
```

```
# Сохраняем отдельные графики для каждого gamma
plot_separate_results(
    gamma_results,
    "Gamma Value Performance",
    "Episode",
    "Reward"
)

# 3. Эксперименты с начальным epsilon (5 значений)
print("\nRunning epsilon start experiments...")
epsilon_starts = [0.2, 0.5, 1.0, 1.5, 2.0]
epsilon_results = {}

for eps in epsilon_starts:
    rewards = train_agent({
        'obs_size': 4,
        'n_actions': 2,
        'gamma': 0.99,
        'epsilon_decay': 0.995,
        'epsilon_start': eps
    })
    epsilon_results[f"eps_start={eps}"] = rewards

# Сохраняем общий график для epsilon_start
plot_individual_results(
    epsilon_results,
    "Different Initial Epsilon Values",
    "Episode",
    "Reward",
    "epsilon_start_comparison"
)

# Сохраняем отдельные графики для каждого epsilon_start
plot_separate_results(
    epsilon_results,
    "Initial Epsilon Performance",
    "Episode",
    "Reward"
```

```
    )

print("\nAll experiments completed! Results saved to 'plots' folder.")
```