Лабораторна робота № 14. Практичне застосування reinforcement learning у симуляціях

Мета: ознайомитись з принципами налаштування, навчання та оптимізації агентів навчання з підкріпленням через веб-інтерфейс. Отримати практичний досвід налаштування різних гіперпараметрів та розуміння їх впливу на ефективність навчання.

Теоретичні відомості

Параметри середовища

Параметр	Опис	Вплив на навчання
Grid Size	Розмір сітки середовища (наприклад, 4x4).	Визначає доступний простір для агента; впливає на складність навігації та пошук оптимальної стратегії.
Number of Coins	Кількість монет, які агент може зібрати в середовищі.	Чим більше монет, тим більше можливостей для отримання нагород і тренування на основі зібраних даних.
Number of Obstacles	Кількість статичних перешкод на сітці.	Впливає на складність навігації, змушуючи агента ухилятися та стратегічно обирати маршрути.
Dynamic Obstacles	Наявність динамічних перешкод, які змінюють своє положення.	Підвищує складність завдання, стимулюючи агента швидше адаптуватися до змінюваних умов середовища.

Конфігурація нагород та покарань

Параметр	Опис	Вплив на навчання
Coin Collected Reward	Нагорода, яку агент отримує за збирання монети.	Підвищує мотивацію агента шукати монети; впливає на швидкість навчання та оптимізацію траєкторій.
Collision Penalty	Штраф за зіткнення з перешкодою.	Заохочує агента уникати перешкод; зменшує кількість випадкових дій та покращує обережність агента.
Step Penalty	Штраф за кожен крок.	Сприяє ефективності рухів, мотивуючи агента скоротити час досягнення цілей.
Completion Reward	Нагорода за завершення епізоду.	Збільшує мотивацію агента досягти цілі, завершивши епізод; покращує загальну результативність навчання.
Timeout Penalty	Штраф за перевищення ліміту часу епізоду.	Мотивує агента прискорити проходження епізоду, щоб уникнути штрафу, що підвищує ефективність навчання.

Параметри агента

Параметр	Опис	Вплив на навчання
Learning Rate	Швидкість навчання.	Визначає, наскільки швидко агент оновлює свої знання; надто висока або низька швидкість може погіршити навчання.
Batch Size	Розмір вибірки даних, що використовується під час навчання.	Впливає на стабільність оновлень параметрів, оскільки більше вибірок дозволяє обробляти більше інформації за раз.
Gamma (Discount)	Коефіцієнт знижки для майбутніх нагород.	Визначає важливість майбутніх винагород порівняно з поточними, впливаючи на довготривалу стратегію агента.
Initial Epsilon	Початкове значення епсілон для ε-жадібної стратегії.	Впливає на початкову ступінь випадковості в діях агента; поступово зменшується, щоб перейти до кращих рішень.
Epsilon Decay	Співвідношення зменшення епсілон.	Впливає на темп зменшення випадковості в діях агента; стабілізує навчання після достатньої кількості епізодів.
Minimum Epsilon	Мінімальне значення епсілон.	Гарантує певний рівень випадковості, що дозволяє агенту знаходити нові стратегії, навіть на пізніх етапах.
Memory Size	Розмір буфера пам’яті для збереження досвіду.	Збільшує кількість збережених спостережень, що сприяє якіснішому навчанню та запобігає «забуванню».

Параметри тренування

Параметр	Опис	Вплив на навчання
Episodes	Кількість навчальних епізодів.	Визначає тривалість процесу навчання; більше епізодів дозволяє агенту краще адаптуватися до середовища.
Render Every	Інтервал рендерингу епізодів.	Впливає на частоту візуалізації процесу навчання; корисно для моніторингу прогресу агента та налаштування моделі.

Необхідне програмне забезпечення

1. Git 2.47.0+
2. Python 3.8+
3. Node.js 16+
4. npm 8+
5. Веб-браузер з підтримкою сучасного JavaScript

Завдання для всіх варіантів

Завдання 1. Базове налаштування середовища.

1. Встановіть усі залежності, згідно з інструкцією у репозиторії.

2. Запустіть бекенд та фронтенд частини проєкту.

3. Перевірте роботу веб-інтерфейсу:

3.1. Відкрийте веб-інтерфейс у браузері.

3.2. Переконайтесь, що всі елементи та функції інтерфейсу працюють без помилок.

3.3. Зафіксуйте результат: зробіть скріншот робочого інтерфейсу та додайте його до звіту.

Завдання 2. Експерименти з параметрами середовища.

1. Проведіть серію експериментів із різними налаштуваннями середовища:

*1.1. Розмір сітки: випробуйте значення **від 4 до 12**.*

*1.2. Кількість монет: варіюйте **від 1 до 5**.*

*1.3. Кількість перешкод: змінюйте від **0 до 8**.*

*1.4. Динамічні перешкоди: **включіть** і **вимкніть** динамічні перешкоди, щоб порівняти вплив.*

2. Виконайте експерименти з конфігурацією винагород:

*2.1. Coin Collected Reward: змінюйте **від 0 до 5**.*

*2.2. Collision Penalty: випробуйте значення **від 0.5 до 5**.*

*2.3. Step Penalty: використовуйте значення в діапазоні **від 0 до -0.1**.*

*2.4. Completion Reward: задайте **від 0 до 10**.*

*2.5. Timeout Penalty: варіюйте значення **від 0 до 5**.*

3. Заповніть таблицю результатів для кожної конфігурації:

Конфігурація	Середня винагорода	Час навчання	Примітки
…	…	…	…

4. Проаналізуйте результати:

4.1.Визначте вплив кожного параметра на швидкість навчання та продуктивність агента.

4.2.Додайте графік або діаграму для ілюстрації впливу змінних.

Завдання 3. Оптимізація гіперпараметрів агента.

1. Підберіть оптимальні значення для наступних гіперпараметрів агента:

*1.1. Learning Rate: значення **від 0.0001 до 0.01**.*

*1.2. Batch Size: значення **від 16 до 256**.*

*1.3. Gamma (Discount Factor): значення **від 0.8 до 1**.*

*1.4. Initial Epsilon: початкове значення ε, варіюйте **від 0 до 1**.*

*1.5. Epsilon Decay: швидкість зменшення ε **від 0.9 до 1**.*

*1.6. Minimum Epsilon: встановіть мінімальне значення **від 0.01 до 0.3**.*

*1.7. Memory Size: кількість збережених станів **від 1,000 до 50,000**.*

2. Порівняйте результати з базовими налаштуваннями:

Параметр	Базове значення	Оптимізоване значення	Результат
…	…	…	…

3. Обґрунтуйте вибір кожного параметра:

3.1.Опишіть, як зміни впливають на ефективність навчання та результати.

Завдання 4. Аналіз процесу навчання

*1. Проведіть навчання з різною кількістю епізодів: **100**, **500**, **1000**.*

2. Побудуйте графіки, що відображають процес навчання:

2.1. Графік винагород за епізод.

2.2. Графік втрат під час навчання.

2.3. Середня тривалість епізоду.

3. Проаналізуйте момент збіжності навчання:

3.1. Визначте, скільки епізодів потрібно для стабільного навчання.

3.2. Додайте висновки про мінімальну кількість епізодів для ефективного навчання.

Завдання 5. Додаткове завдання.

1. Розширення агента:

1.1. Модифікуйте `agent.py`*, додавши нову стратегію дослідження.*

1.2. Оцініть, наскільки нова стратегія покращує результати порівняно з базовими.

2. Зміни у середовищі:

2.1. Внесіть зміни до `environment.py`*, додавши нові режими гри.*

2.2. Оцініть, як ці зміни впливають на процес навчання.

3. Вдосконалення моделі:

3.1. Покращіть архітектуру мережі у `model.py`*, додавши нові шари або функції активації.*

3.2. Оцініть, як це впливає на швидкість і точність навчання.

** За необхідності внесіть додаткові зміни в програмний код*

Базова структура роботи

Завдання 1. Базове налаштування середовища.

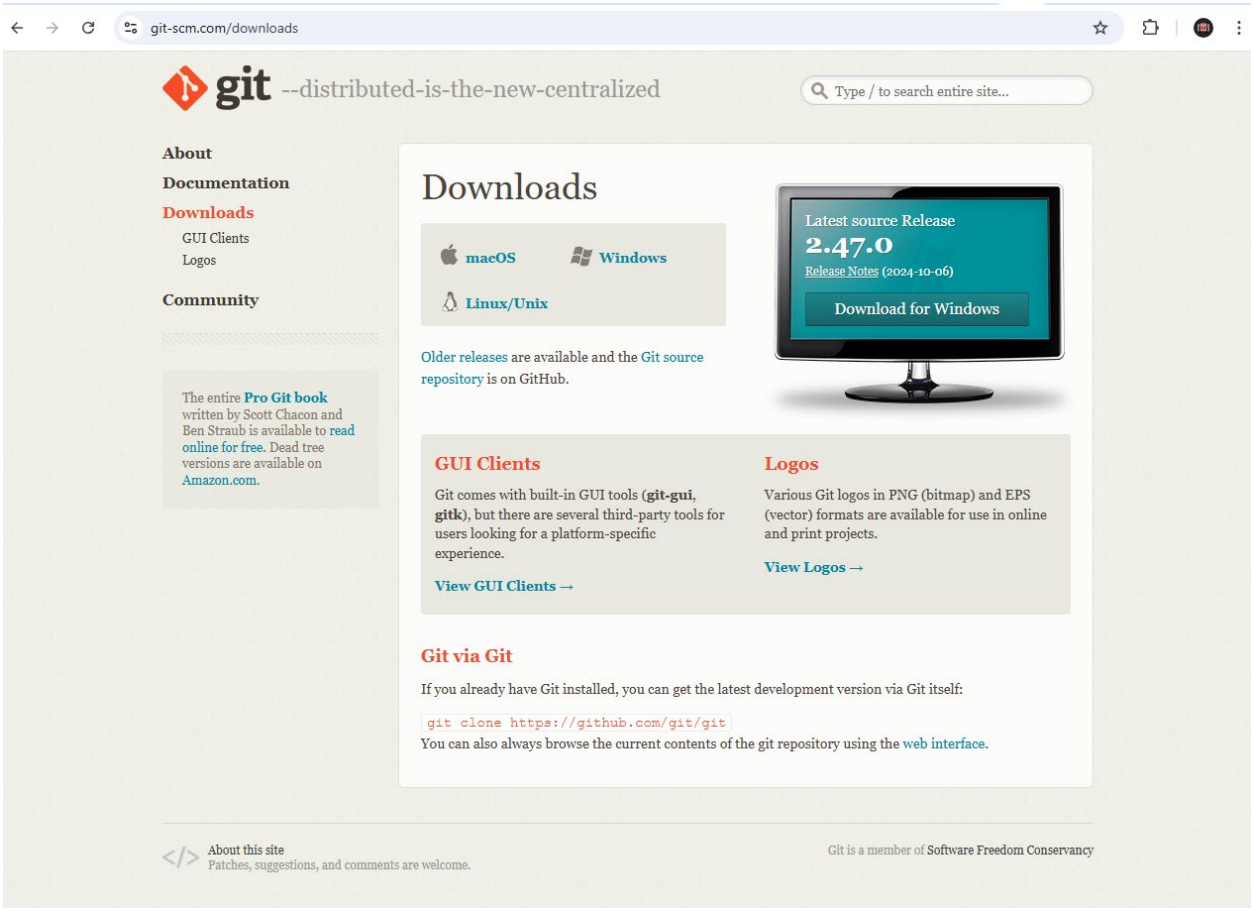

Рисунок 1.1. Встановлення Git

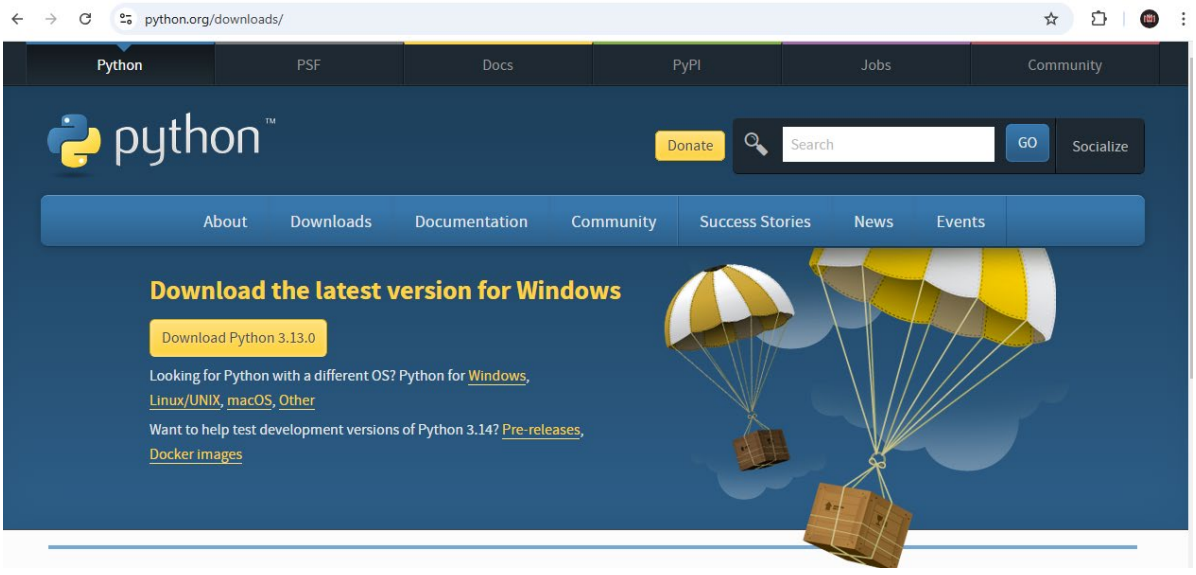

Рисунок 1.2. Встановлення Python

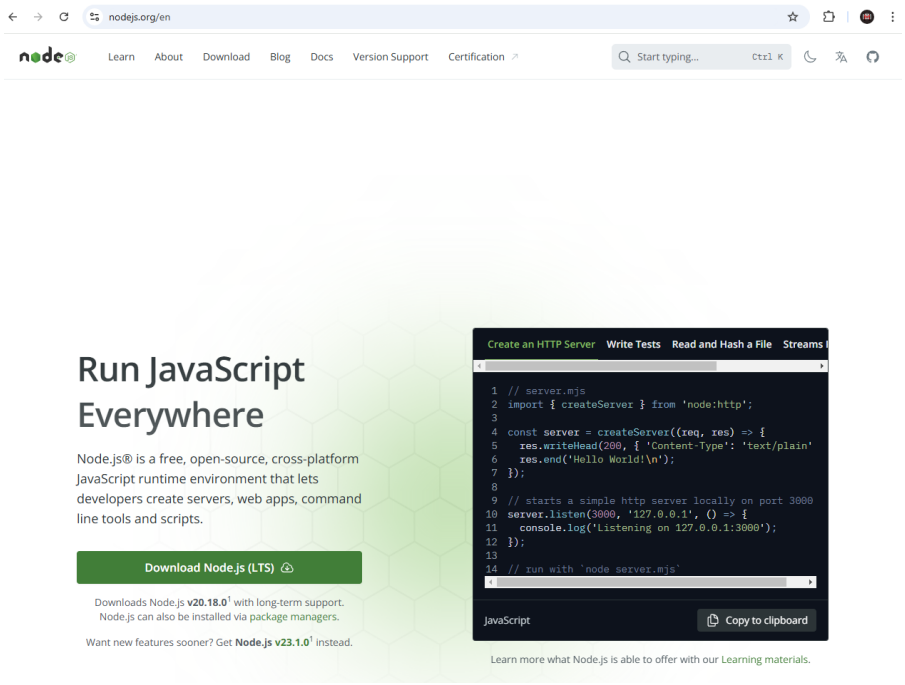

Рисунок 1.3. Встановлення Node.js

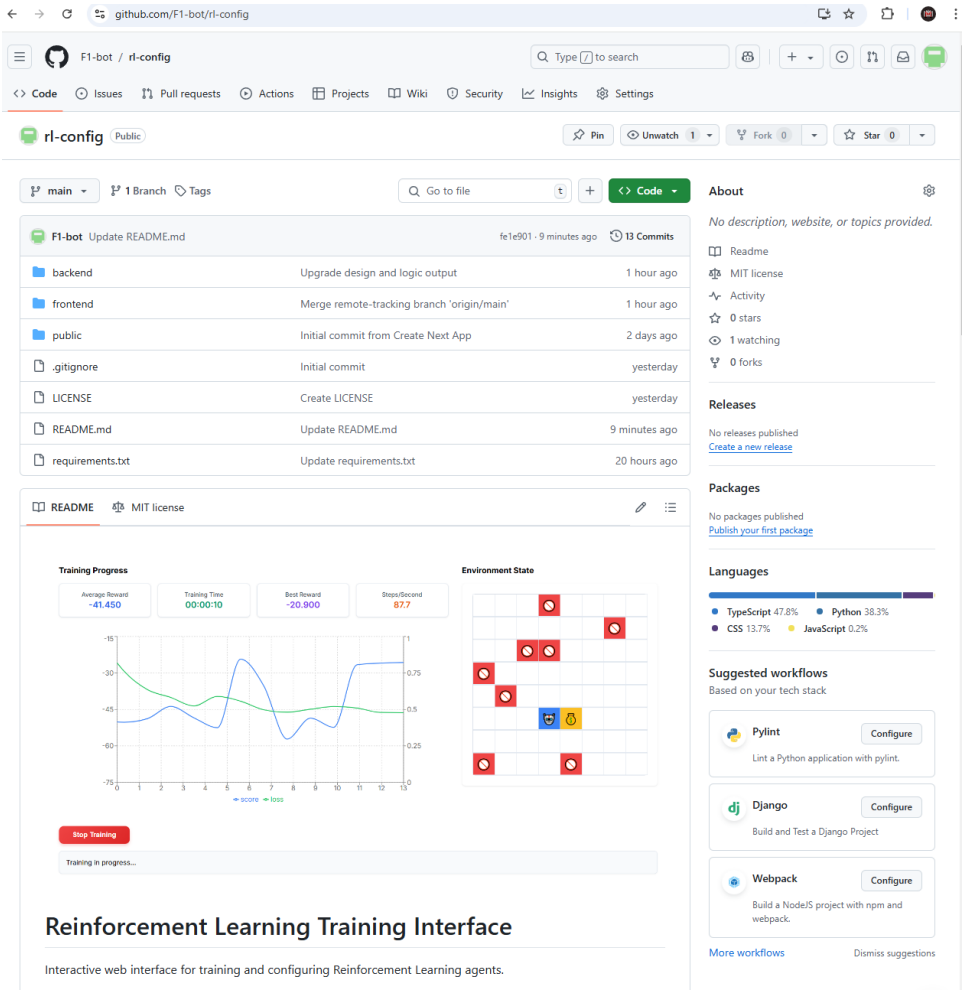

Рисунок 1.4. Вигляд репозиторію rl-config

Клонування репозиторію:

```
git clone https://github.com/F1-bot/rl-config.git
cd rl-config
```

```
C:\Users\Marano>cd C:\Users\Marano\Pictures\test

C:\Users\Marano\Pictures\test>git clone https://github.com/F1-bot/rl-config.git
Cloning into 'rl-config'...
remote: Enumerating objects: 123, done.
remote: Counting objects: 100% (123/123), done.
remote: Compressing objects: 100% (99/99), done.
remote: Total 123 (delta 33), reused 90 (delta 15), pack-reused 0 (from 0)
Receiving objects: 100% (123/123), 249.43 KiB | 1.63 MiB/s, done.
Resolving deltas: 100% (33/33), done.

C:\Users\Marano\Pictures\test>cd rl-config

C:\Users\Marano\Pictures\test\rl-config>
C:\Users\Marano\Pictures\test\rl-config>
```

Рисунок 1.5. Клонування репозиторію

Налаштування віртуального середовища Python:

```
python -m venv venv
```

```
venv\Scripts\activate
```

```
C:\Users\Marano\Pictures\test\rl-config>
C:\Users\Marano\Pictures\test\rl-config>python -m venv venv

C:\Users\Marano\Pictures\test\rl-config>venv\Scripts\activate

(venv) C:\Users\Marano\Pictures\test\rl-config>
```

Рисунок 1.6. Створення віртуального середовища Python

Встановлення необхідних залежностей для Python:

```
pip install -r requirements.txt
```

```
(venv) C:\Users\Marano\Pictures\test\rl-config>pip install -r requirements.txt
Collecting annotated-types==0.7.0
  Using cached annotated_types-0.7.0-py3-none-any.whl (13 kB)
Collecting anyio==4.6.2.post1
  Using cached anyio-4.6.2.post1-py3-none-any.whl (90 kB)
Collecting click==8.1.7
  Using cached click-8.1.7-py3-none-any.whl (97 kB)
Collecting colorama==0.4.6
```

Рисунок 1.7. Завантаження залежностей для Python

Примітка! В межах `requirements.txt` версія torch надана для роботи з CPU. Якщо Ви бажаєте використовувати GPU скористайтесь цієї інформацією задля коректної інсталяції: youtube.com/watch?v=M60_J-jjtn0&ab_channel=AleksandarHaberPhD

Старт роботи сервера fastapi:

```
cd backend
python server.py
```

```
(venv) C:\Users\Marano\Pictures\test\rl-config\backend>python server.py
←[32mINFO←[0m:     Started server process [←[36m40152←[0m]
←[32mINFO←[0m:     Waiting for application startup.
←[32mINFO←[0m:     Application startup complete.
←[32mINFO←[0m:     Uvicorn running on ←[1mhttp://127.0.0.1:8001←[0m (Press CTRL+C to quit)
```

Рисунок 1.8. Запуск сервера fastapi

Встановлення необхідних залежностей для Node.js (у другому терміналі):

```
cd frontend
npm install
```

```
C:\Users\Marano\Pictures\test\rl-config\frontend>npm install
npm WARN deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache i
f you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerfu
l.
npm WARN deprecated @humanwhocodes/config-array@0.13.0: Use @eslint/config-array instead
npm WARN deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm WARN deprecated glob@7.1.7: Glob versions prior to v9 are no longer supported
npm WARN deprecated @humanwhocodes/object-schema@2.0.3: Use @eslint/object-schema instead
npm WARN deprecated eslint@8.57.1: This version is no longer supported. Please see https://eslint.org/version-support fo
r other options.

added 434 packages, and audited 435 packages in 24s

139 packages are looking for funding
```

Рисунок 1.9. Завантаження залежностей для Node.js

Запуск веб-інтерфейсу Next.js (у другому терміналі):

```
npm run dev
```

```
C:\Users\Marano\Pictures\test\rl-config\frontend>npm run dev

> rl-config@0.1.0 dev
> next dev

 Port 3000 is in use, trying 3001 instead.
  ▲ Next.js 13.5.6
  - Local:        http://localhost:3001

 Ready in 8.2s
o Compiling /page ...
 Compiled /page in 16.9s (1544 modules)
 Compiled /favicon.ico/route in 2.5s (1542 modules)
```

Рисунок 1.10. Запуск веб-інтерфейсу Next.js

Перехід до локального сайту:

```
http://localhost:3000
```

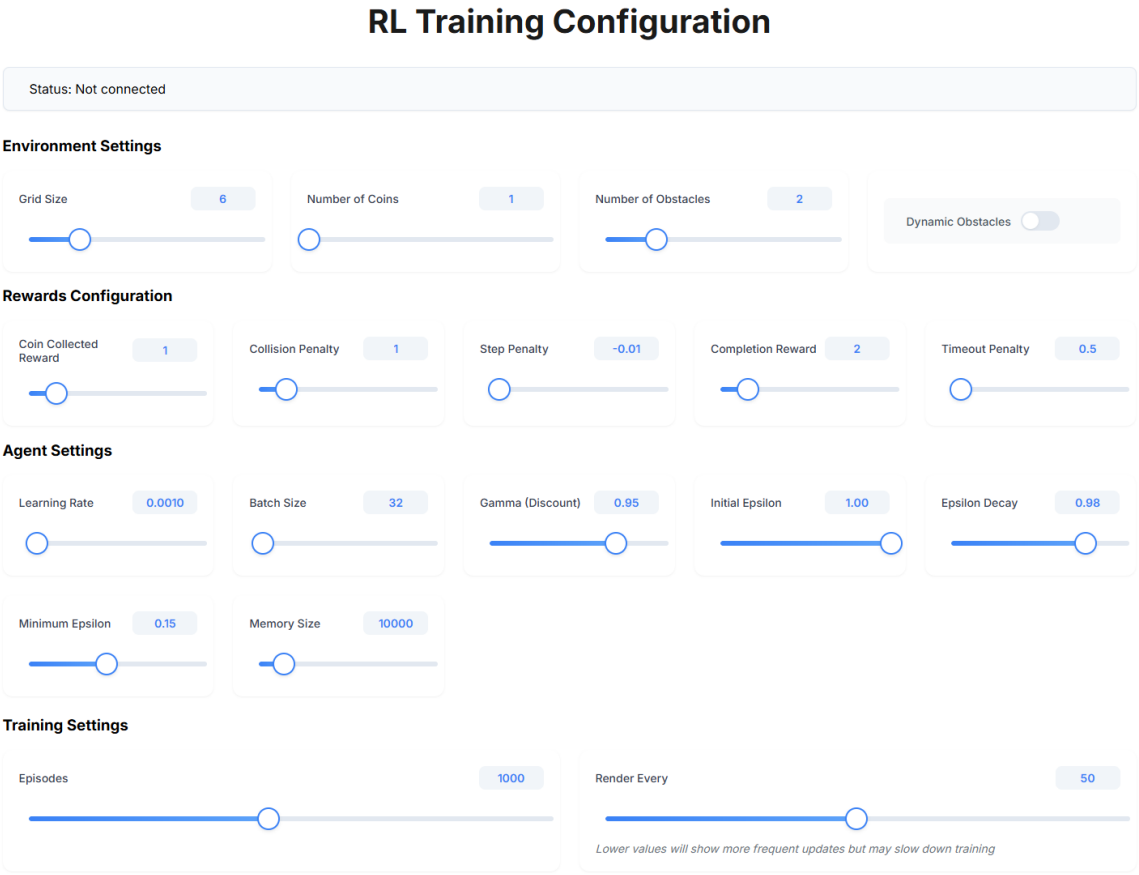

Рисунок 1.11. Вигляд сайту (частина 1)

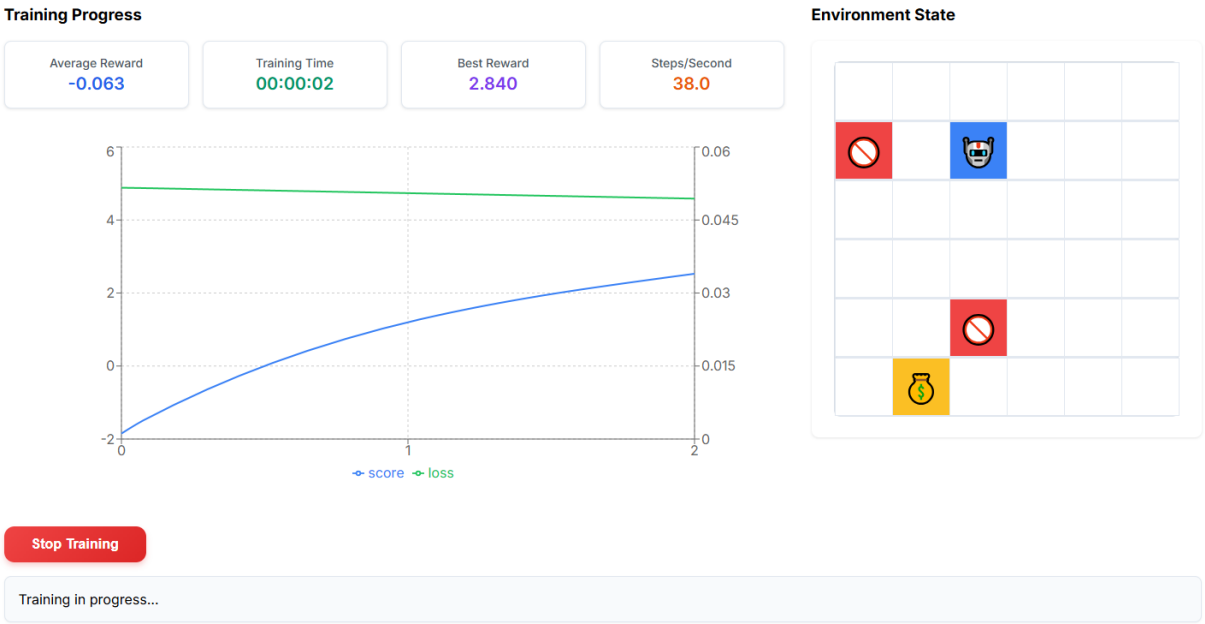

Рисунок 1.12. Вигляд сайту (частина 2)

Завдання 2. Експерименти з параметрами середовища.

Конфігурація		Середня винагорода	Час навчання	Примітки
Environment Settings Grid size: 6 Number of Coins: 1 Number of Obstacles: 2 Dynamic Obstacles: false	**Rewards Configuration** Coin Collected Reward: 2 Collision Penalty: 0.5 Step Penalty: -0.01 Completion Reward: 3 Timeout Penalty: 1	1.560	00:00:27	Best Reward: 5.000 Steps/Second: 85.9
Agent Settings | Default |	**Training Settings** Episodes: 100			
Environment Settings Grid size: 6 Number of Coins: 3 Number of Obstacles: 3 Dynamic Obstacles: true	**Rewards Configuration** Coin Collected Reward: 2 Collision Penalty: 0.5 Step Penalty: -0.01 Completion Reward: 3 Timeout Penalty: 1	2.816	00:00:36	Best Reward: 8.990 Steps/Second: 83.3
Agent Settings | Default |	**Training Settings** Episodes: 100			
Environment Settings Grid size: 10 Number of Coins: 3 Number of Obstacles: 3 Dynamic Obstacles: true	**Rewards Configuration** Coin Collected Reward: 2 Collision Penalty: 0.5 Step Penalty: -0.01 Completion Reward: 3 Timeout Penalty: 1	0.174	00:03:12	Best Reward: 8.860 Steps/Second: 48.7
Agent Settings | Default |	**Training Settings** Episodes: 100			
…	**…**	…	…	…
…	**…**			

Висновки: …

Завдання 3. Оптимізація гіперпараметрів агента.

Параметр	Базове значення	Оптимізоване значення	Результат			
Learning Rate	0.0001	0.0010	Average Reward	5.870	1.796	-69.39%
			Training Time	00:00:25	00:00:38	+34%
			Best Reward	8.970	8.980	+0.11%
			Steps/Second	83.8	84.8	+1.19%
Batch Size	32	96	Average Reward	1.796	1.109	-38.25%
			Training Time	00:00:38	00:00:39	+2.63%
			Best Reward	8.980	8.930	-0.56%
			Steps/Second	84.8	86.3	+1.77%
Memory Size	10000	50000	Average Reward	1.109	0.668	-39.76%
			Training Time	00:00:39	00:00:41	+5.13%
			Best Reward	8.930	8.970	-0.45%
			Steps/Second	86.3	84.1	-2.55%
…	…	…	Average Reward	…	…	…
			Training Time	…	…	…
			Best Reward	…	…	…
			Steps/Second	…	…	…

Висновки: …

Завдання 4. Аналіз процесу навчання.

Рисунок 1.13. Проведення навчання зі 100 епізодами

Рисунок 1.14. Проведення навчання з 500 епізодами

Рисунок 1.15. Проведення навчання з 1000 епізодами

Висновки: ...

Завдання 5. Додаткове завдання.

agent.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
from collections import deque
import random
from torch.cuda.amp import autocast, GradScaler
from model import SimpleNet
from utils import DEVICE, Experience


class SimplifiedAgent:
    def __init__(self, state_shape, n_actions, learning_rate=1e-3):
        self.state_shape = state_shape  # Повинно бути (7, size, size)
        self.n_actions = n_actions

        # Гіперпараметри
        self.batch_size = 32
        self.gamma = 0.95
        self.epsilon = 1.0
        self.epsilon_decay = 0.98
        self.epsilon_min = 0.15
        self.learning_rate = learning_rate

        # Мережа
        self.policy_net = SimpleNet(state_shape, n_actions).to(DEVICE)
        self.target_net = SimpleNet(state_shape, n_actions).to(DEVICE)
        self.target_net.load_state_dict(self.policy_net.state_dict())

        self.optimizer = optim.Adam(self.policy_net.parameters(),
lr=learning_rate)
        self.memory = deque(maxlen=10000)

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def get_action(self, state):
        if random.random() < self.epsilon:
            return random.randrange(self.n_actions)

        with torch.no_grad():
            state = torch.FloatTensor(state).unsqueeze(0).to(DEVICE)
            q_values = self.policy_net(state)
            return q_values.argmax(1).item()
```

```
    def train(self):
        if len(self.memory) < self.batch_size:
            return None

        batch = random.sample(self.memory, self.batch_size)
        states, actions, rewards, next_states, dones = zip(*batch)

        states = torch.FloatTensor(np.array(states)).to(DEVICE)
        actions = torch.LongTensor(actions).to(DEVICE)
        rewards = torch.FloatTensor(rewards).to(DEVICE)
        next_states = torch.FloatTensor(np.array(next_states)).to(DEVICE)
        dones = torch.FloatTensor(dones).to(DEVICE)

        current_q_values = self.policy_net(states).gather(1,
actions.unsqueeze(1))
        next_q_values = self.target_net(next_states).max(1)[0].detach()
        expected_q_values = rewards + (1 - dones) * self.gamma *
next_q_values

        loss = F.smooth_l1_loss(current_q_values.squeeze(),
expected_q_values)

        self.optimizer.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(self.policy_net.parameters(), 1.0)
        self.optimizer.step()

        # Оновлюємо target network кожні 10 батчів
        if random.random() < 0.1:
            self.target_net.load_state_dict(self.policy_net.state_dict())

        return loss.item()

    def update_epsilon(self):
        self.epsilon = max(self.epsilon_min, self.epsilon *
self.epsilon_decay)
```

environment.py

```
import numpy as np
import random
from typing import List, Tuple, Dict

class SimplifiedGameEnv:
    def __init__(self, size: int = 6, n_coins: int = 1, n_obstacles: int =
2,
                 dynamic_obstacles: bool = False, rewards: dict = None):
        self.size = size
```

```
        self.n_coins = n_coins
        self.n_obstacles = n_obstacles
        self.dynamic_obstacles = dynamic_obstacles
        self.rewards = rewards or {
            'coinCollected': 1,
            'collision': -1,
            'step': -0.01,
            'completion': 2,
            'timeout': -0.5
        }
        self.action_space_n = 8
        self.observation_space_shape = (7, size, size)
        self.max_steps = size * size
        self.steps = 0
        self.grid = np.zeros((7, size, size))
        self.score = 0
        self.reset()

    def reset(self):
        self.steps = 0
        self.score = 0
        self.grid.fill(0)

        # Розміщення агента в центрі
        self.agent_pos = [self.size // 2, self.size // 2]

        # Розміщення монет та перешкод
        self.coins = []
        self.obstacles = []
        self._place_objects(self.coins, self.n_coins)
        self._place_objects(self.obstacles, self.n_obstacles)

        self.update_grid()
        return self.get_state()

    def _place_objects(self, objects: List, count: int) -> None:
        # Створюємо список всіх можливих позицій
        available_positions = []
        for i in range(self.size):
            for j in range(self.size):
                if ([i, j] != self.agent_pos and
                        [i, j] not in self.coins and
                        [i, j] not in self.obstacles):
                    available_positions.append([i, j])

        # Вибираємо випадкові позиції
        n_positions = min(count, len(available_positions))
        if n_positions > 0:
```

```
            selected_positions = random.sample(available_positions,
n_positions)
            objects.extend(selected_positions)

    def update_grid(self) -> None:
        self.grid.fill(0)

        # Базові канали
        self.grid[0, self.agent_pos[0], self.agent_pos[1]] = 1
        for coin in self.coins:
            self.grid[1, coin[0], coin[1]] = 1
        for obs in self.obstacles:
            self.grid[2, obs[0], obs[1]] = 1

        # Distance map для монет
        if self.coins:
            for i in range(self.size):
                for j in range(self.size):
                    min_dist = min(abs(i - coin[0]) + abs(j - coin[1])
                                   for coin in self.coins)
                    self.grid[3, i, j] = 1 - min_dist / (2 * self.size)

        # Вільний простір
        self.grid[4] = 1 - (self.grid[0] + self.grid[1] + self.grid[2])

        # Додаємо напрямки до найближчої монети
        if self.coins:
            closest_coin = min(self.coins,
                               key=lambda c: abs(c[0] - self.agent_pos[0]) +
                                             abs(c[1] - self.agent_pos[1]))
            dx = closest_coin[0] - self.agent_pos[0]
            dy = closest_coin[1] - self.agent_pos[1]
            dist = max(abs(dx), abs(dy), 1)
            self.grid[5].fill(dx / dist)
            self.grid[6].fill(dy / dist)

    def get_state(self) -> np.ndarray:
        return self.grid.copy()

    def step(self, action: int) -> Tuple[np.ndarray, float, bool, bool,
Dict]:
        self.steps += 1
        old_pos = self.agent_pos.copy()

        # Рух агента
        moves = [(-1, 0), (-1, 1), (0, 1), (1, 1),
                 (1, 0), (1, -1), (0, -1), (-1, -1)]
        dx, dy = moves[action]
```

```
        self.agent_pos[0] = np.clip(self.agent_pos[0] + dx, 0, self.size -
1)
        self.agent_pos[1] = np.clip(self.agent_pos[1] + dy, 0, self.size -
1)

        # Перевірка колізій та винагород
        if self.agent_pos in self.obstacles:
            self.agent_pos = old_pos
            reward = self.rewards['collision']
            done = False
        else:
            reward = self.rewards['step']
            done = False

            # Збір монет
            if self.agent_pos in self.coins:
                self.coins.remove(self.agent_pos)
                reward = self.rewards['coinCollected']
                self.score += 1
                if not self.coins:
                    reward += self.rewards['completion']
                    done = True

            # Перевірка ліміту кроків
            if self.steps >= self.max_steps:
                reward += self.rewards['timeout']
                done = True

        # Динамічні перешкоди
        if self.dynamic_obstacles and not done and random.random() < 0.1:
            self._update_dynamic_obstacles()

        self.update_grid()
        info = {
            "score": self.score,
            "steps": self.steps,
            "coins_left": len(self.coins)
        }

        return self.get_state(), reward, done, False, info

    def _update_dynamic_obstacles(self):
        # Рухаємо кожну перешкоду з певною ймовірністю
        for i, obs in enumerate(self.obstacles):
            if random.random() < 0.3:  # 30% шанс руху
                possible_moves = [
                    (dx, dy) for dx, dy in [(0, 1), (1, 0), (0, -1), (-1,
0)]
```

```
                    if 0 <= obs[0] + dx < self.size and 0 <= obs[1] + dy <
self.size
                ]
                if possible_moves:
                    dx, dy = random.choice(possible_moves)
                    new_pos = [obs[0] + dx, obs[1] + dy]
                    # Перевіряємо, чи нова позиція не зайнята
                    if (new_pos != self.agent_pos and
                        new_pos not in self.coins and
                        new_pos not in self.obstacles):
                        self.obstacles[i] = new_pos

    def render(self) -> None:
        grid = [[' ' for _ in range(self.size)] for _ in range(self.size)]

        for obs in self.obstacles:
            grid[obs[0]][obs[1]] = 'X'
        for coin in self.coins:
            grid[coin[0]][coin[1]] = 'O'
        grid[self.agent_pos[0]][self.agent_pos[1]] = 'A'

        print("\n".join([''.join(row) for row in grid]))
        print(f"Score: {self.score}, Steps: {self.steps}")
```

model.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from typing import Tuple

class SimpleNet(nn.Module):
    def __init__(self, input_shape, n_actions):
        super(SimpleNet, self).__init__()

        input_channels = input_shape[0]  # Отримуємо кількість вхідних
каналів

        self.conv = nn.Sequential(
            nn.Conv2d(input_channels, 32, kernel_size=3, stride=1,
padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU()
        )
```

```
        # Обчислюємо розмір виходу згорткових шарів
        conv_out_size = self._get_conv_out(input_shape)

        self.fc = nn.Sequential(
            nn.Linear(conv_out_size, 512),
            nn.ReLU(),
            nn.Linear(512, n_actions)
        )

    def _get_conv_out(self, shape):
        o = self.conv(torch.zeros(1, *shape))
        return int(np.prod(o.shape))

    def forward(self, x):
        conv_out = self.conv(x)
        return self.fc(conv_out.view(conv_out.size(0), -1))
```

Висновки: …

Welcome to Github Repository

github.com/F1-bot/rl-config