МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

"ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ" НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ “КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

**Курсова робота**

з дисципліни «Об’єктно-орієнтоване програмування»

***на тему: «Дослідження та порівняння деяких технік навчання з підкріпленням на базі задачі «Лабіринт»»***

РОБОТУ ВИКОНАВ:

студент групи ДА-82 Зайцев К.Ю. Прийняв:

\_ \_ Булах Б.В.

„\_\_\_” \_ 2019

Київ – 2019

# ЗМІСТ

[[ВСТУП 3](#_TOC_250006)](#_Toc28211028)

[[РОЗДІЛ I 5](#_TOC_250006)](#_Toc28211029)

[[1.1. Класифікація лабіринтів 5](#_TOC_250006)](#_Toc28211030)

[[1.2. Алгоритми вирішення лабіринту 7](#_TOC_250006)](#_Toc28211031)

[[1.3. Базові концепції навчання з підкріпленням 9](#_TOC_250006)](#_Toc28211032)

[[1.4. Глибоке навчання з підкріпленням 12](#_TOC_250006)](#_Toc28211033)

[[РОЗДІЛ II 14](#_TOC_250006)](#_Toc28211034)

[[2.1. Розробка програми 14](#_TOC_250006)](#_Toc28211035)

[[РОЗДІЛ III 26](#_TOC_250006)](#_Toc28211036)

[[ВИСНОВКИ 29](#_TOC_250006)](#_Toc28211037)

[[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 31](#_TOC_250006)](#_Toc28211038)

[[ДОДАТКИ 32](#_TOC_250006)](#_Toc28211039)

# ВСТУП

Проблема взаємодії з середовищем, маючи неповні відомості про його структуру є активною сферою досліджень, починаючи з другої половини минулого століття. Прийняття рішень, коли агент, який приймає рішення, знає про різні можливі стани оточення, але не має достатньо інформації, щоб призначити їм імовірнісні оцінки, називається прийняттям рішень у стані невизначеності. Рішення в умовах невизначеності - це існування багатьох невідомих і неможливість визначити, що може статися в майбутньому, щоб отримати бажаний результат.

Базовим середовищем для моделювання таких ситуацій виступає лабіринт. Дослідження поведінки model-free агентів, можливість зробити з дискретного лабіринту неперервний, що міститиме надвелику для комп’ютерних обчислень кількість клітин і т.д. – мотивація використання цього середовища для тестування теоретичних здобутків у навчанні з підкріпленням.

Робот робить крок вперед, потім наштовхується на перешкоду. Наступного разу він робить інший крок і здатний просунутися у лабіринті. Робот багато разів пробує різні комбінації варіантів; Врешті-решт, він засвоює правильні кроки, які потрібно робити, і робить їх на шляху до виходу. Цей процес і називається навчанням з підкріпленням. Він безпосередньо пов'язує дію робота з результатом, при цьому робот не повинен засвоювати складний взаємозв'язок між його дією та результатами. Робот вчиться ходити на основі винагороди (перебування на рівновазі) та покарання (падіння). Цей відгук вважається "підкріпленням" для того, щоб робити чи не робити дію.

Ще один приклад навчання підкріплення можна знайти під час гри в гру Go. Якщо агент відкладе свій білий камінь у деякому місці, а потім потрапить в оточення чорних шматочків і втратить цей простір, його карають за такий крок. Після декількох поразок він уникатиме ходів, що будуть ставити власні камінчики в оточення, тим самим зменшивши ризик їх програти.

Спрощеним визначенням навчання з підкріпленням - є вивчення кращих дій, заснованих на винагороді чи покаранні. У зв’язку з ним існує три основні поняття: стан, дія та винагорода. Середовище описує поточну ситуацію. Для робота, який навчається ходити, стан - це положення його двох ніг. Для програми Go - стан - це позиції всіх камінців на дошці.

Дія - це те, що може зробити агент у кожному положенні. Враховуючи стан або положення двох його ніг, робот може робити кроки на певну відстань. Зазвичай існують обмежені дії у фіксованому діапазоні, які здійснює агент. Наприклад, кроком робота може бути, скажімо, 0,01 метр на 1 метр. Програма Go може зробити хід лише в одній клітинці на всій дошці, розміром 19 x 19.

Коли робот здійснює дію, він отримує винагороду. Тут термін «винагорода» - це абстрактне поняття, яке описує зворотний зв'язок від навколишнього середовища. Нагорода може бути позитивною чи негативною. Коли винагорода є позитивною, вона відповідає нормальному розумінню винагороди. Коли винагорода є негативною, вона відповідає тому, що ми зазвичай називаємо «покаранням».

На прикладі лабіринту ми дослідимо основні положення та практики навчання з підкріпленням та його переваги та недоліки порівняно зі стандартними методами вирішення подібних задач.

За мету для даної роботи було поставлено створення емулятора для моделювання лабіринтів різної складності та агентів, що їх вирішують, імплементацію різноманітних алгоритмів навчання з підкріпленням та їх порівняння між собою та алгоритмами пошуку найкоротшого шляху на графі, яким можна представити лабіринт.

У відповідності з метою роботи поставлені наступні завдання:

* проаналізувати стан проблеми в науковій літературі та Інтернет ресурсах;
* запрограмувати алгоритми з різних сімейств навчання з підкріпленням мовою Python;
* розглянути об’єктно-орієнтований підхід до даної задачі;
* застосувати низку патернів проектування для взаємодії всередині програми;

# РОЗДІЛ I

**ТЕОРЕТИЧНИЙ АНАЛІЗ НАУКОВОЇ ЛІТЕРАТУРИ ТА ВИХІДНІ ПОНЯТТЯ РОБОТИ**

Лабіринт - це сітчастий n-мірний (зазвичай двовимірний) простір довільної форми, як правило, прямокутної [14]. Комірка - це елемент лабіринту. Комірка визначається як обмежений n-мірний простір, як правило, двовимірна область і інтерпретується як позиція агента.

Термін "агент" визначається як сутність, яка здатна рухатися в межах лабіринту. Агент може рухатися в будь-якому довільному напрямку за умови, що простір не зайнятий перешкодою або рух не порушує перешкоду. Таким чином, ми можемо визначити два типи перешкод: перешкоду, яка повністю займає комірку, що називається стінкою, і перешкоду, яка розділяє дві сусідні комірки, називають перегородкою [14].

# Класифікація лабіринтів

У типових проблемах цього типу агент ініціалізується у певній стартовій позиції, з якої він починає шукати вихід. Місце розташування цілі може знаходитися всередині меж лабіринту, наприклад, розташування віртуальної їжі або міфологічного створіння, або може бути точкою виходу з лабіринту. Проблеми лабіринту можуть бути класифіковані відповідно до властивостей лабіринту та властивостей агентів, що його перетинають [14].

Розмірність лабіринту визначає кількість вимірів, які має лабіринт, наприклад, планарний лабіринт використовує два виміри, а кубічний лабіринт - три [15]. Загалом, лабіринт може використовувати будь-яку кількість вимірів, хоча для людини буде складно уявити більш ніж три. Розмір лабіринту - це кількість комірок, що складають лабіринт. Він зазвичай сильно пов'язаний з довжиною шляху, що з'єднує стартову та цільову позиції, або відстань. Відстань визначається також перешкодами в лабіринті, які можна описати або абсолютною кількістю кроків, або, що значно важливіше, щільністю перешкод, тобто відношенням кількості зайнятих комірок до величини лабіринту. Щільний лабіринт може збільшити відстань, але не обов'язково складність. Зокрема, лабіринтом вважається середовище, в якому можливий лише один звивистий шлях, таким чином обхід лабіринту від початку до кінця може бути тривалим, але не складним.

Перешкоди також надають агентам більше інформації про сенсор у вигляді відмінних форм. Для прикладу, порожній лабіринт не надасть агенту ніякої картографічної інформації, якщо агент не здатний сприймати та обробляти весь простір лабіринту. Складніший випадок, коли агент не в змозі розмежувати між повторюваними патернами у лабіринті, явище, яке називається псевдонімом [14]. Перешкоди також визначають топологію лабіринту [16].

Лабіринт може бути представлений графіком G, що складається з вузлів, що представляють собою комірки, і ребер, кожне з яких являє собою прохідний шлях, що з'єднує пару комірок. У загальному випадку граф може містити підграфи, що не з'єднані між собою, роблячи таким чином частини лабіринту недоступними для агента. Якщо граф з’єднаний, є два можливих випадки для графічного ведення від початкової точки до мети: лише один шлях з'єднує дві точки, або більше одного шляху з'єднують їх. Перший випадок може бути представлений деревом, а другий - просто графом, див. Рис. 1.

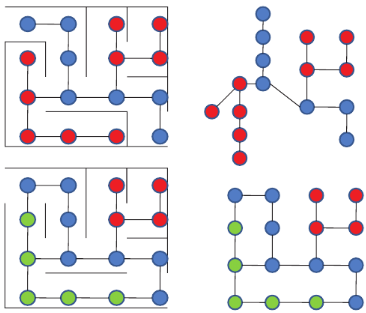


Рисунок 1. Лабіринти представлені деревом (верх) чи графом (низ)

Апріорні знання, які має агент щодо лабіринту, також виступають важливим фактором. Їх об’єм може варіюватися в інтервалі між повною відсутністю та абсолютним знанням. У першому випадку агент повинен використовувати власні сенсорні можливості для вивчення та картографування навколишнього середовища. Після того як картографування зроблено, агент може спланувати шлях, який веде його від початку до кінця. Якщо агент має абсолютне апріорне знання, він може просто спланувати шлях до мети. Однак, апріорні знання є необхідною, але недостатньою умовою успішного планування шляху, оскільки не всі лабіринти є статичними.

Як правило, в лабіринті є три різні типи змін [14]:

1) Зміни структури лабіринту, тобто стіни чи перегородки змінюють власне розташування;

2) Зміни місця розташування цілі;

3) Зміни, викликані рухом інших агентів у лабіринті, у випадку групи з кількома агентами.

# Алгоритми вирішення лабіринту

У більшості випадків задача звучить наступним чином: дано лабіринт у формі бінарної прямокутної матриці, знайти найкоротший шлях в лабіринті з заданих позицій початку та фінішу. Шлях може складатися з клітин, що мають значення 1, і в одну одиницю часу робиться лише один крок. Валідними кроками є: вверх, вниз, вліво, вправо.

**Алгоритм Лі**

Одним з можливих рішень для проблеми лабіринту є рішення, що базується на використання Breadth-First Search. Воно завжди дає оптимальний результат, якщо такий існує, але є повільним та вимагає значних витрат пам’яті. Такі алгоритми мають спільну структуру:

* + - 1. Створити пусту чергу та закинути в неї клітину-початок лабіринту, позначити її як відвідану.
      2. Доки черга не пуста, повторювати:
         1. Забрати з черги передній елемент.
         2. Якщо він є фінішом, повернути дистанцію.
         3. Інакше для кожної з 4 прилежних до даної клітин, перевірити чи є вони вільними та закинути їх у чергу з дистанцією +1 та позначкою, що вони відвідані.
         4. Повернути false, якщо всі клітини відвідані та фініш не знайдено.

Часова складність даного алгоритму O(MN), де M, N – параметри лабіринту.

**Алгоритм Дейкстри**

Алгоритм Дейкстри дозволяє обчислити найкоротший шлях між одним вузлом та кожним іншим вузлом у графі, яким можна представити лабіринт. Вершини – клітини лабіринту, ребра – зв'язок між сусідніми вільними клітинами. Обчислимо найкоротший шлях між вузлом С та іншими вузлами нашого графіка:

1. Позначити початкову вершину та створити масив з відстаней до усіх інших вершин, позначивши всі відстані нескінченністю.
2. Вибрати вершину з найменшою відстанню до неї як поточну.
3. Для кожної сусідньої вершини: додати до поточної відстані відстань до даної вершини. Якщо вона менша за записану в таблиці – оновити таблицю.
4. Позначити поточну вершину як відвідану.
5. Якщо залишились не відвідані вершини – перейти до кроку 2.

Часова складність алгоритму Дейкстри – O(E+VlogV), де Е, V – кількість ребер та вершин відповідно. Суттєвим мінусом алгоритму є відсутність будь-якого «відчуття» вірної траекторії руху, що на практиці часто призводить до серії зайвих проходів по лабіринту.

**Алгоритм А\***

Алгоритм Greedy Best-First-Search працює аналогічно алгоритму Дейкстри, за винятком того, що він має деяку оцінку (що називається евристичною) того, наскільки далеко від цілі знаходиться деяка вершина. Замість вибору вершини, найближчої до початкової, вона вибирає вершину, найближчу до мети. Жадібний Best-First-Search не гарантовано знайде найкоротший шлях. Однак він працює набагато швидше, ніж алгоритм Дейкстри, тому що він використовує евристичну функцію для того, щоб знайти шлях до фінішу дуже швидко.

А\* алгоритм об’єднує інформацію, яку використовує Алгоритм Дейкстри (надаючи перевагу вершинам, близьким до вихідної точки), та інформацію, яку використовує Greedy Best-First-Search (надаючи перевагу вершинам, близьким до мети). g(n) являє собою довжину шляху від вихідної точки до будь-якої вершини n, а h(n) являє собою евристичну оцінюючу довжину від вершини n до фінішу. На наведених діаграмах жовтий (h) являє собою вершини, далекі від мети, а teal (g) являє собою вершини, далекі від вихідної точки. A \* врівноважує два, рухаючись від початкової точки до мети. В основному циклі алгоритм на кожному кроці обирає деяку вершину n, яка має найменшу відстань f(n) = g (n) + h (n). Алгоритм має наступний вигляд:

* + 1. Обрати клітину у списку з доступними клітинами з найменшим f(n).
    2. Додати обрану клітину до списку вершин, що не розглядаються.
    3. Для кожної сусідньої клітини Т:
       1. Якщо Т у списку вершин, що не розглядаються – ігнорувати.
       2. Якщо Т не у списку доступних клітин – додати Т та обчислити f(T).
       3. Якщо Т у списку доступних клітин – перевірити поточний шлях до поточної вершини та порівняти його зі шляхом у Т + від Т до поточної вершини. Якщо відстань менша – оновити f(n) та f усіх батьківських вершин.

Часова складність алгоритму – О(Е). Основним його недоліком є використання пам’яті для зберігання додаткових масивів, що рівне O(bd), де b – середня кількість дочірніх елементів по усім клітинам, d – глибина оптимального шляху.

# Базові концепції навчання з підкріпленням

Навчання з підкріпленням визначається як метод машинного навчання, який вивчає, як програмні агенти мають діяти у деякому середовищі. Згідно зі Стівеном Хуангом, основні терміни визначаються наступним чином:

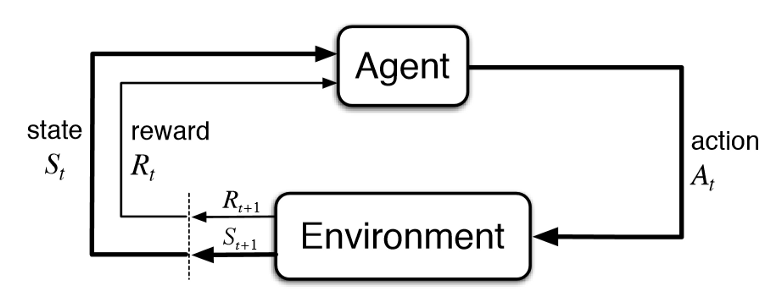
* **Агент**: це деяка сутність, яка певним чином діє в середовищі, щоб отримувати певну винагороду.
* **Середовище** (e): сценарій, у якому знаходиться агент.
* **Винагорода** (R): міра того, на скільки вдалою чи невдалою була остання дія агента у середовищі .
* **Стан** (s): стан посилається на поточну ситуацію, яку повертає середовище.
* **Політика** (π): Це стратегія, яка застосовується агентом для вирішення наступної дії на основі поточного стану.
* **Цінність** (V): очікувана винагорода від дій агента згідно з поточною політикою, починаючи з даного стану.
* **Функція цінності**: вона визначає вартість держави, яка є загальною сумою винагороди. Це агент, якого слід очікувати, починаючи з цього стану.
* **Модель середовища**: це імітує поведінку навколишнього середовища агента. Це допомагає зробити висновки, а також визначити, як поводитиметься середовище.
* **Model-based method**: це метод вирішення задач навчання з підкріпленням на основі знань про середовище.
* **Q-значення** (або значення дії) (Q): значення Q досить подібне до цінності. Єдина відмінність між ними - це те, що він приймає додатковий параметр – поточну дію.

Рис. 1: Базова схема роботи навчання з підкріпленням

**Як працює RL?**

Базове навчання з підкріпленням включає такі кроки (рис. 1):

1. моніторинг середовища;
2. вирішення як діяти, використовуючи певну стратегію;
3. відповідна дія;
4. отримання винагороди чи штрафу;
5. навчання на досвіді та вдосконалення початкової стратегії;
6. ітерувати до тих пір, поки не буде знайдена оптимальна стратегія;

Існує 3 основних підходи, коли мова йде про реалізацію алгоритму RL:

* На основі цінності - в методі навчання з підкріпленням на основі

цінності агент намагається максимально використовувати значення V(s). Основний фокус - пошук оптимального значення V\*.

* На основі політики - в методі навчання, що базується на політиці, цільовою є така політика, щоб дія, яка виконується в кожному стані, була оптимальною для отримання максимальної винагороди в майбутньому. Основна увага приділяється пошуку оптимальної політики.
* На основі моделі - у цьому типі навчання з підкріпленням експертом створюється модель для кожного середовища, і агент вчиться діяти в цьому конкретному середовищі.

**Markov Decision Process**

Процесс прийняття рішень Маркова (Markov Decision Process) – формальний опис взаємодії між середовищем та агентом, що певним чином в ньому діє. Даний фреймворк вирізняє залежність не тільки між станом середовища та зчиненою дією, а й існування *історії* станів. MDP можна зобразити набором з (S, A, P, R, ):

S – набір станів

A – набір дій

P (s, a, s/) – вірогідність, що дія а у стані s в в час t приведе у стан s/.

R (s, a, s/) – винагорода після переходу у стан s/

discounted factor, використовується для формулювання проблеми з врахуванням майбутніх винагород.

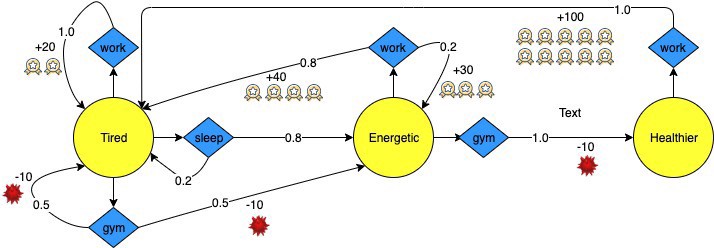


Рис. 2: Приклад проблеми, що описується як MDP-процес

**Рівняння Беллмана**

Для визначення state-value функції V(s) існує рівняння у вигляді:

Це рівняння називається рівнянням Беллмана і означає, що цінність у певного стану визначається як очікувана винагорода при слідуванні поточній політиці до кінця епізоду. Значення визначає на скільки важливими для агента будуть майбутні винагороди і лежить на проміжку між 0 та 1. Іншими словами, цінність певного епізоду можна визначити за допомогою епізодів після нього, що відкриває двері для ітераційних підходів.

# Глибоке навчання з підкріпленням

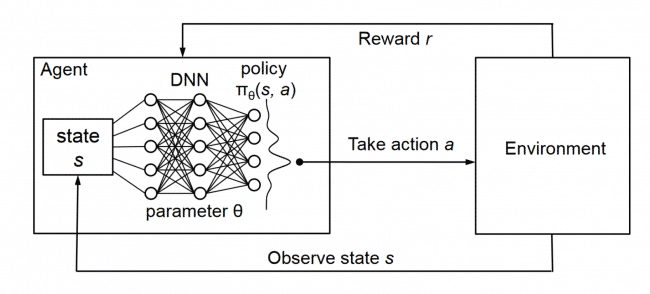
Одним з основних недоліків запропонованого фреймворку вирішення проблем є те, що нові проблеми доведеться вирішувати заново. Агент не вчиться загальним правилам роботи середовища, у якому він знаходиться – він вчиться шукати найбільш оптимальний вихід в даній конкретній ситуації. Це мотивувало прихід до навчання з підкріпленням нейронних мереж, що представили можливість не тільки вчитися узагальнювати отриманий досвід і застосовувати його у нових середовищах, а й відкрили такі напрями як автономне водіння і т.п.

Рис. 3: Приклад реформатованого під використання нейронних мереж процесу Маркова

Виходом такої мережі є очікувана цінність стану, базуючись на усіх можливих діях. Тренування полягає у налаштуванні параметрів для кожної дії так, щоб максимізувати очікуване Q-value.

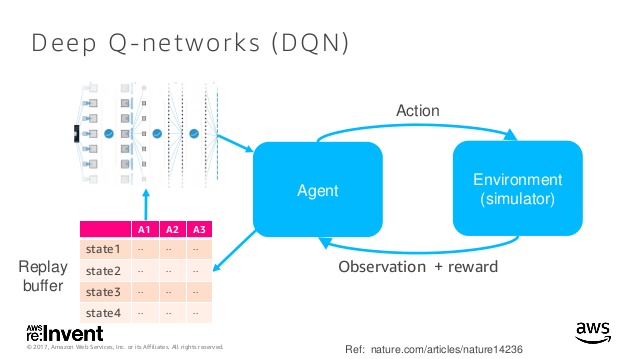
На жаль, сигнал від успішного вирішення задачі зменшується з кожним шаром такої мережі, а тренування сходиться до найкращих параметрів не гарантовно. Це мотивувало розробку таких архітектур як Dueling Networks, DQN with Experience replay, та інші. В даній роботі імплементовано DQN, що використовує дані не всього епізоду, а деякої вибірки з останніх станів та дій.

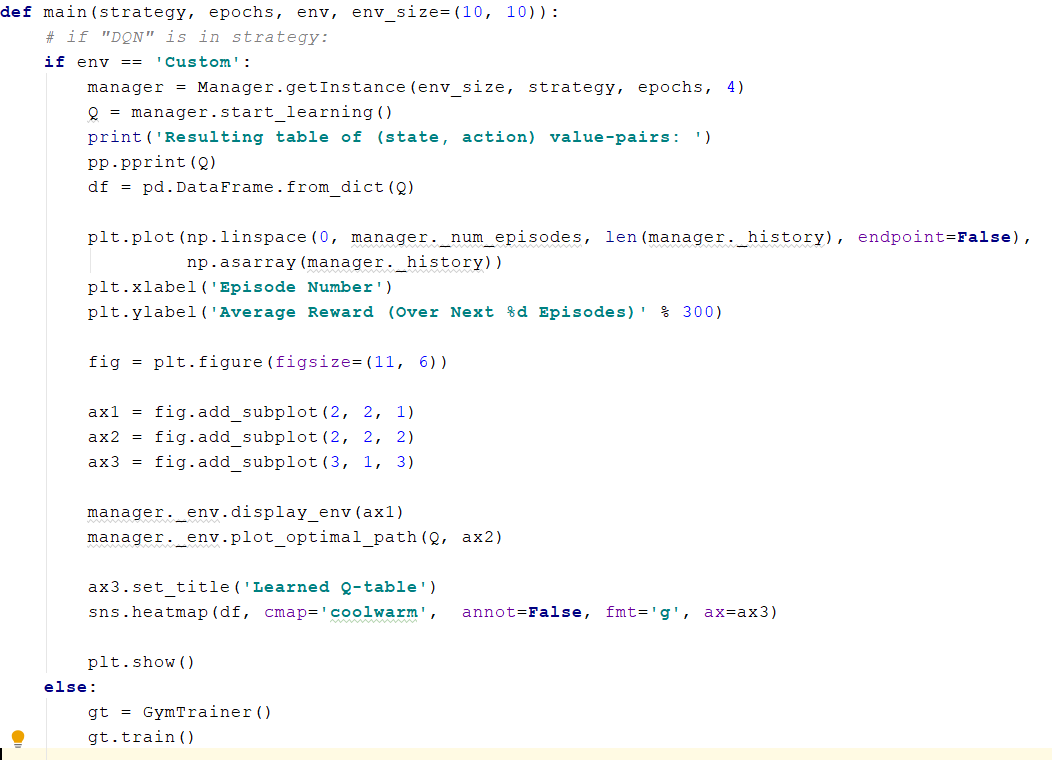
Рис. 4: Deep Q-Network with Experience Replay

# РОЗДІЛ II

**ПОБУДОВА І РЕАЛІЗАЦІЯ ПРОЕКТУ**

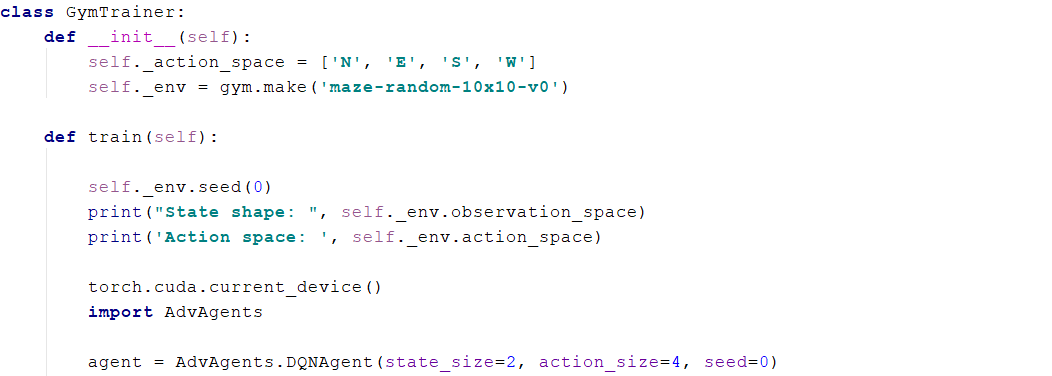
# Розробка програми

Для моделювання взаємодії між агентом та середовищем було створено декілька модулів, що містять в собі основні компоненти для навчання з підкріпленням: агенти, політики навчання та середовища.

Основним модулем є менеджер, що збирає в єдину модель задані користувачем типи агентів, середовищ та політик:

*env* – тип середовища, приймає значення ‘Custom’/’OpenAI Gym’

Модуль Trainers відповідає за створення підкласів-менеджерів у відповідному середовищі – лабіринті, ініціалізованому вручну чи у Gym:



*GymTrainer* – клас для керування агентами у Gym, *DQNAgent* – клас у модулі з агентами, що використовують DRL, *action\_space –* кількість можливих дій у середовищі*, observation\_space –* параметри лабіринту.

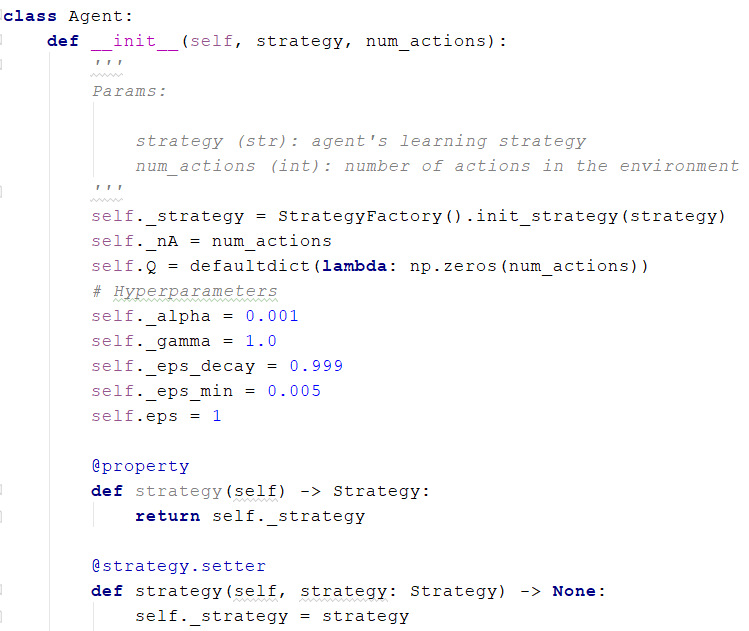
Для зберігання класів агентів відокремлено модулі SimpleAgents та AdvAgents для агентів, що реалізують Монте-Карло, Temporal-difference та DRL методи відповідно:

Рисунок 1. Модуль SimpleAgents

*\_strategy* – стратегія, яку реалізує агент, *Q* – таблиця (state, value) пар (метод часових різниць), *alpha, gamma –*  гіперпараметри навчання.

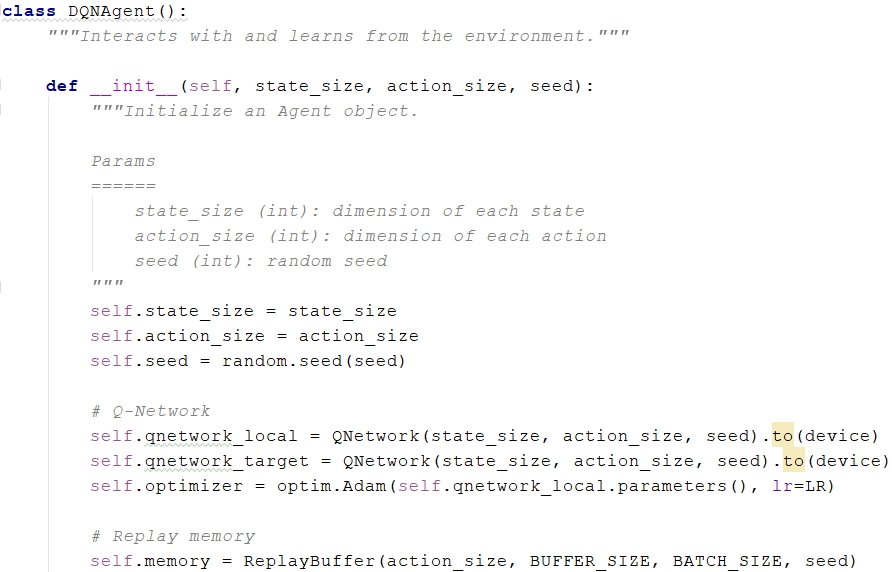
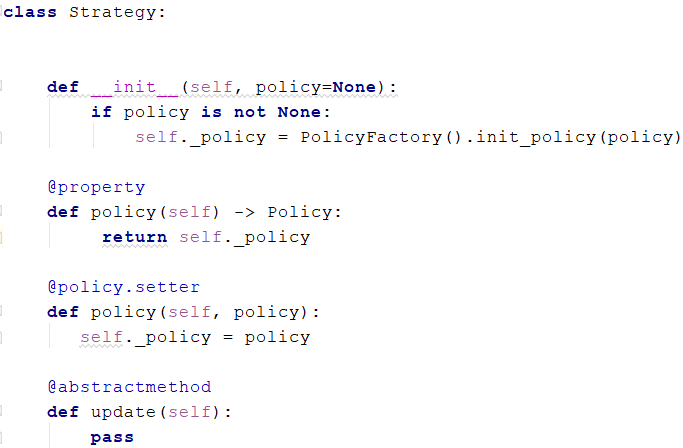
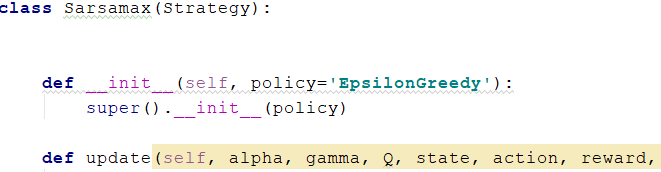


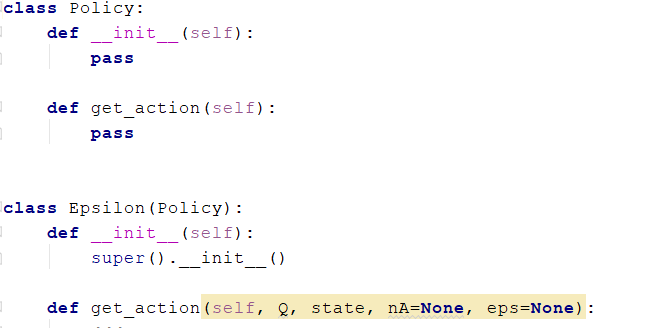
Рисунок 2. Модуль AdvAgents

*seed* – фіксоване рандомне число для репродукції результатів, *qnetwork* – нейронна мережа, виходом з якої є доцільність тієї чи іншої дії у певному стані.

Стратегії визначаються у модулі Strategies:

*policy* – політика вибору дій для реалізації стратегії, *update* –

поліморфна функція, яку кожна стратегія перевизначає залежно від структур даних, з якими працює.

Політики визначені у модулі Policies:

*get\_action* – метод для вибору дії, базуючись на принципах, визначених певною політикою.

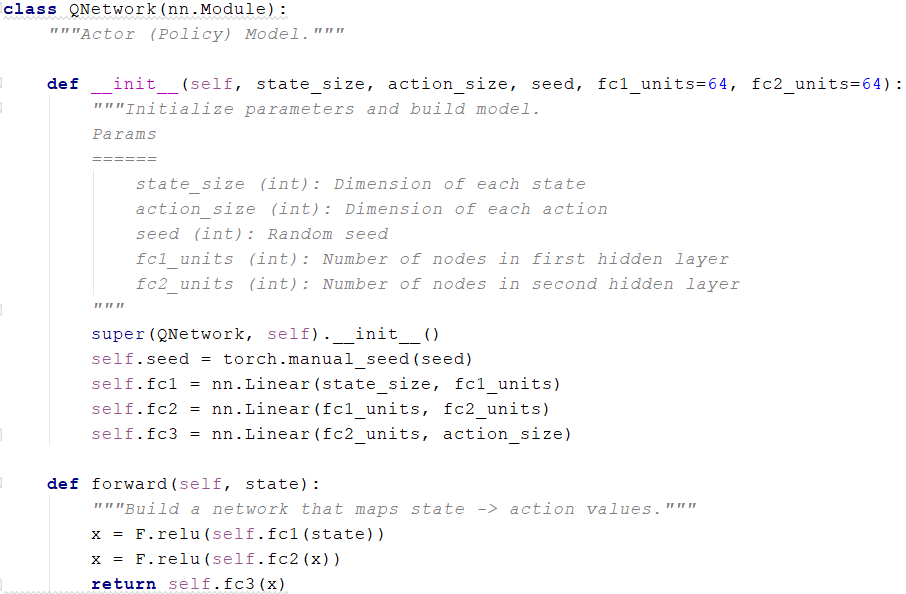
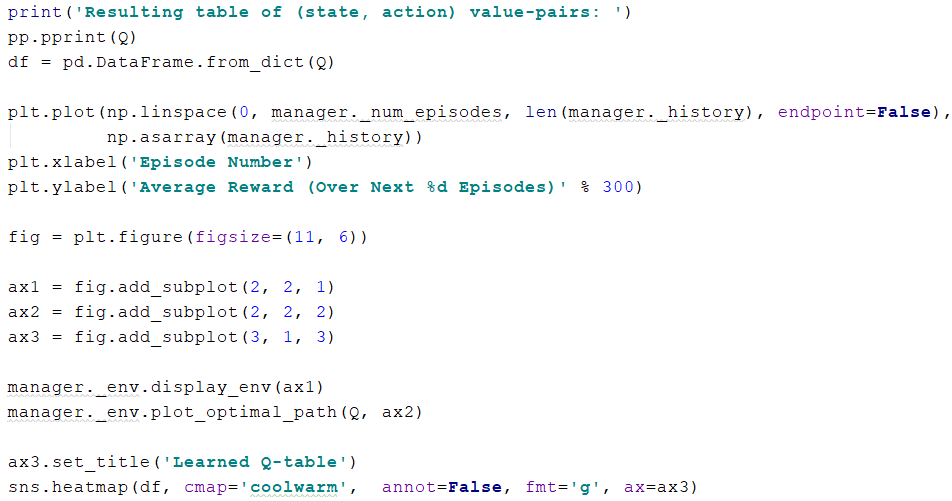
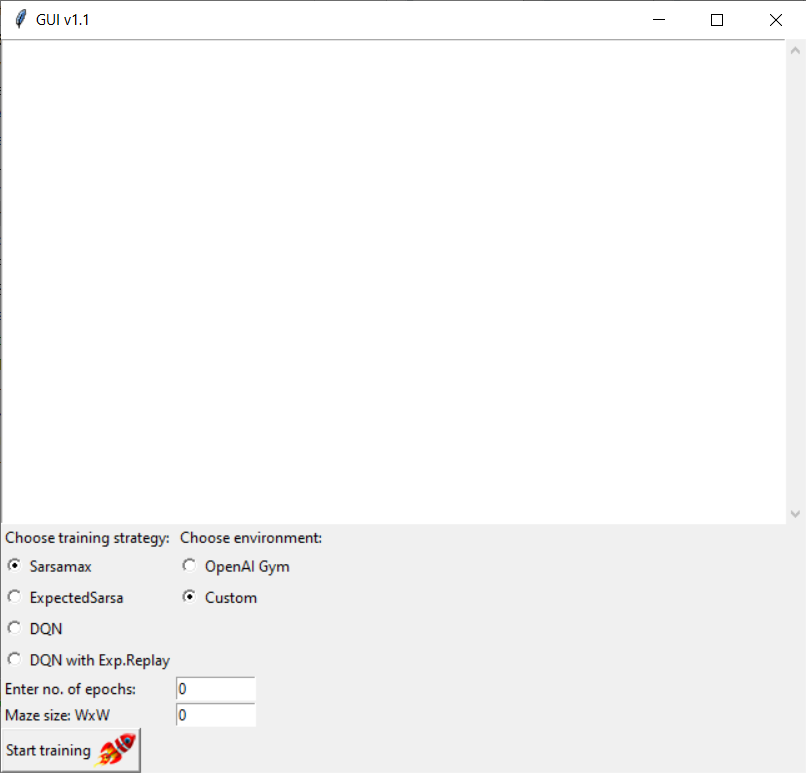
Для реалізації глибокого навчання з підкріпленням було створено модуль model.py, де за допомогою фреймворку Pytorch створено нейронну мережу з декількома fully-connected шарами. Функцією активації обрано RectifiedLinearUnit, оскільки вона диференційовна на всьому проміжку та не так швидко призводить до затухання градієнтів при зворотньому поширенню похибки, як функції на кшталт Sigmoid, Tanh і їм подібні.

Рисунок 3. Архітектура мережі для задачі «Лабіринт»

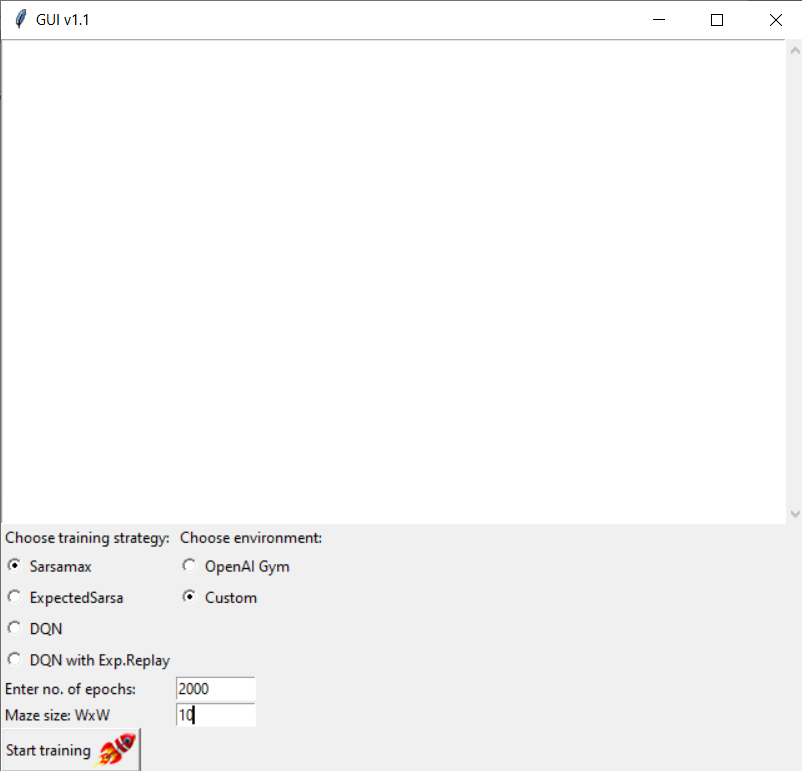
В даній роботі основний фокус приділявся алгоритму Q-learning, що на практиці працює кращі за такі методи як Sarsa та Expected Sarsa. Результати роботи алгоритму вивчалися за допомогою будування словника state, value) пар та візуалізації найкращих ходів для певних положень засобами Matplotlib.

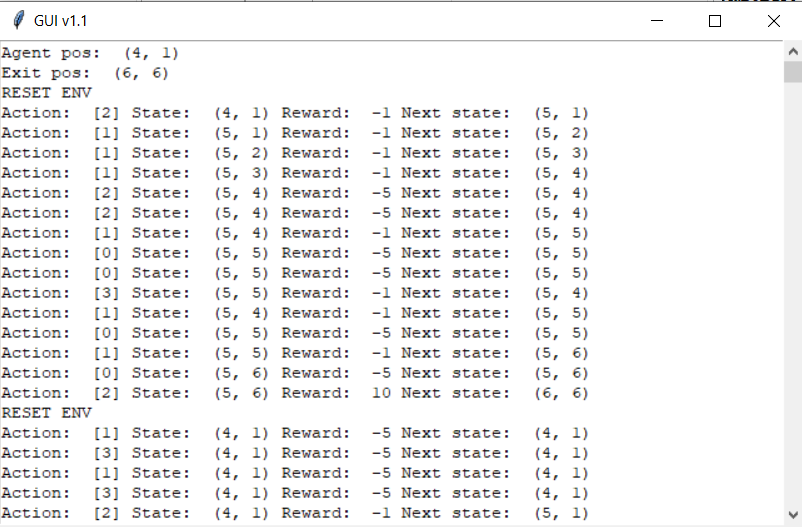
# РОЗДІЛ III

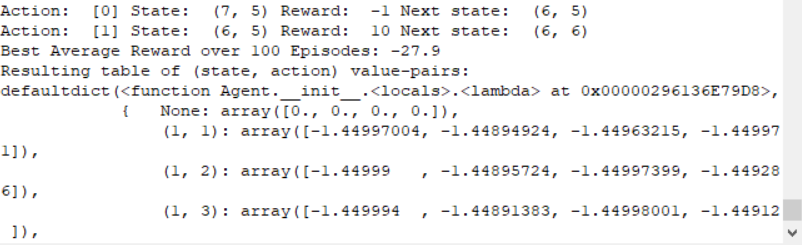
**ІНТЕРФЕЙС КОРИСТУВАЧА**

Програму розроблено для роботи у графічному інтерфейсі, що створено за допомогою python-бібліотеки Tkinter. Для його запуску необхідно ввести у консолі команду *python tkinter\_main.py, після чого* користувач потрапляє у наступне вікно:

Наступні кроки здійснюються по введенні користувачем запропонованої інформації: кількості епох для тренування, параметрів лабіринту, стратегій навчання та середовища, де тренування відбувається. Для початку тренування, користувач має заповнити всі поля форми ненульовими цілими значеннями:



Після натиснення кнопки *Start training* у фоновому режимі запускається модуль Manager.py з параметрами, взятими з полів форми. Результати виконання програми розміщуються у текстовому полі. Їх можна відслідкувати до кінця , використовуючи скроллбар.



Результати представлені у вигляді графіку навчання, що означає покращення орієнтації агента у лабіринті з часом:

та кінцевої політики дій:

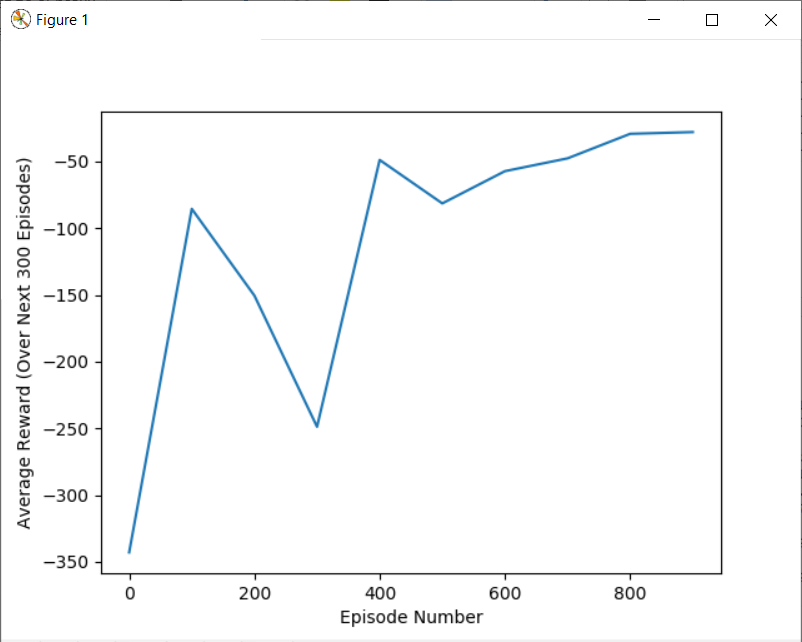


Рисунок 4. Вісь х – кількість пройдених епізодів (початок-фініш), вісь у – середня винагорода за епізод

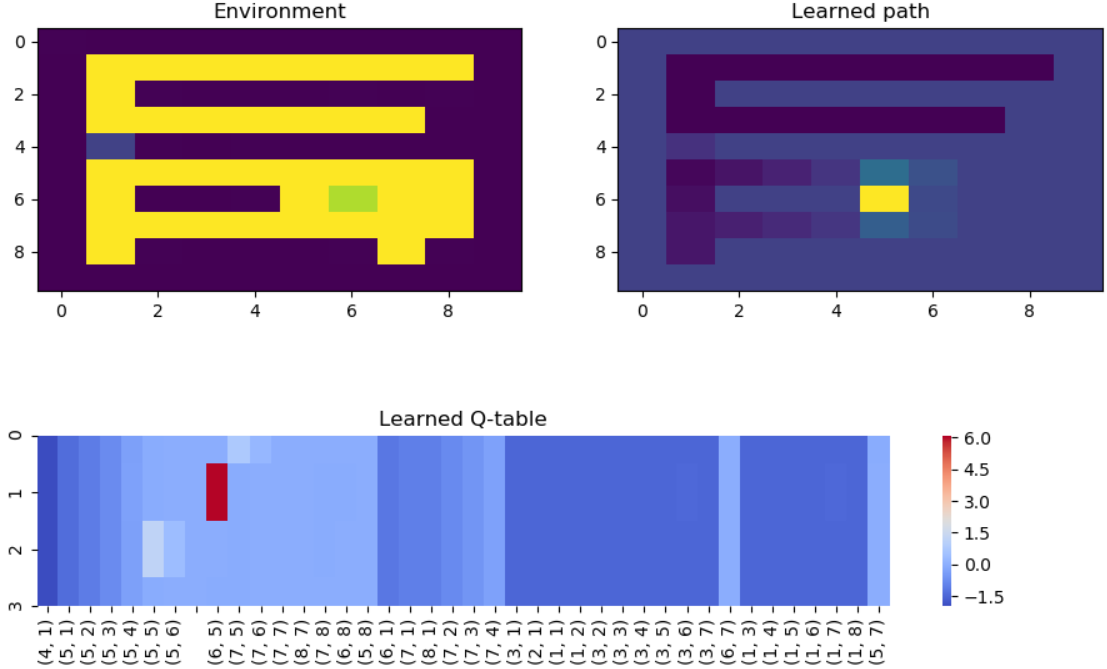


Рисунок 5. Створений лабіринт та найкращий шлях (право) від початку (темна позначка) до фінішу (зелена позначка).

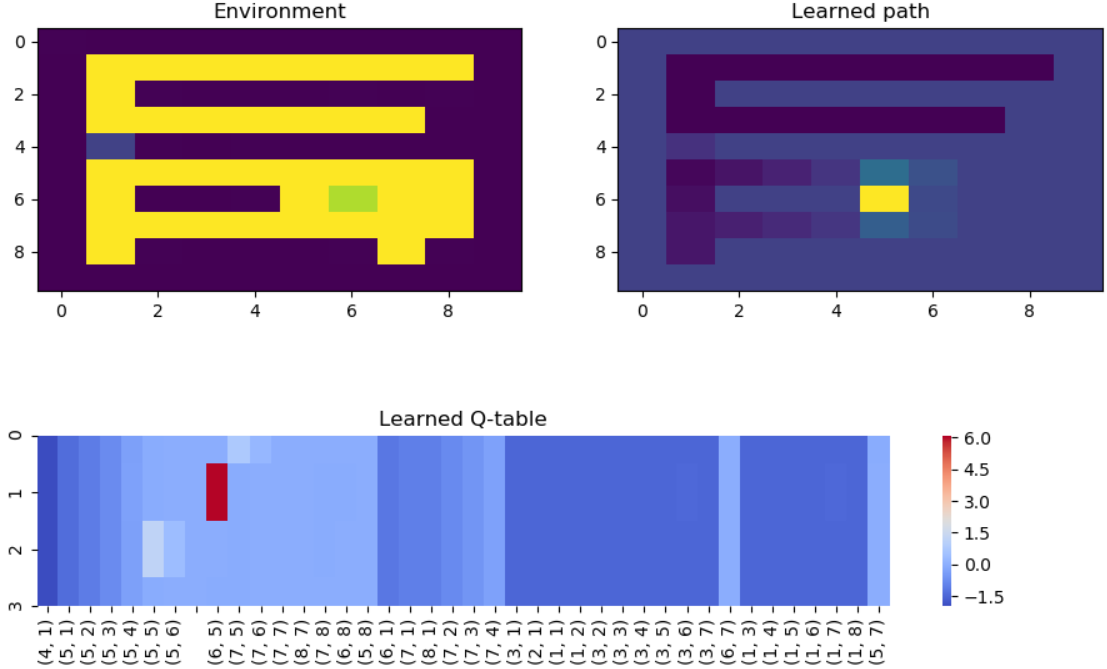
Закриття всіх вікон приведе до початкового інтерфейсу, де експеримент можна продовжити, задавши нові параметри лабіринту та кількість епох:

Рисунок 6. Таблиця пар значень Положення, дія, де позитивний результат від тієї чи іншої дії відображається кольором від темно-зеленого (погана дія) до червоного (найкраща у положенні)

# ВИСНОВКИ

У ході виконання лабораторної роботи було досліджено гілку машинного навчання – навчання з підкріпленням та застосовано її для вирішення задачі «Лабіринт» шляхом створення низки програмних агентів. Застосовуючи методи, що не використовують модель середовища для процесу навчання, отримано бажаний результат у формі найкращих дій для певних положень у лабіринті.

Паралельно дослідивши класичні методи для вирішення даної задачі, можна стверджувати, що методи навчання з підкріпленням мають суттєвий недолік – часову складність, яка в більшості випадків є експоненційною та залежить від розмірів середовища та кількості наявних дій. У свою чергу стандартні алгоритми пошуку найкоротшого шляху на графах, що з успіхом застосовуються у випадку лабіринту, дають час, що лінійно залежить від кількості ребер та вершин.

Це дає змогу зробити висновок щодо незастосовності навчання з підкріпленням у сферах, де рішення може бути знайдено відомими методами аналітичної геометрії, тим паче у разі лінійної, квадратичної, кубічної часової складності. Реальними сферами застосування даної техніки виступають ті, де складність аналізу ситуацій, що з’являються, перевищує можливості людини і потребує автоматизованої роботи штучного агента чи деякій їх кількості протягом великої кількості часу.

У роботі розібрано загальну структуру та ключові компоненти алгоритмів пошуку найкоротшого шляху в лабіринті, досліджено застосування навчання з підкріпленням для даної задачі. Звідки можна зробити наступні висновки:

* навчання з підкріпленням не має застосовуватись до задач з дискретним середовищем та обмеженими діями;
* використання матриць суміжності є неефективним підходом для пошуку шляхів на графах ;
* методи динамічного програмування можна застосовувати для пошуку найкоротшого шляху;
* алгоритми пошуку на графах є найефективнішими для вирішення задачі «Лабіринт»;
* більш прості методи навчання з підкріпленням дають кращий результат швидше за будь-які інші;
* Temporal-Difference методи дають результат за меншу кількість часу порівняно з Monte-Carlo методами у задачах, де винагорода не досягається у межах одного епізоду;
* Q-learning є найбільш оптимальним порівняно з Sarsa та Expected Sarsa за рахунок використання жадібної політики прийняття рішень;
* ефективність вирішення задач тим чи іншим алгоритмом має перевірятися на прикладі конкретної задачі. Універсально гарного методу не існує;
* для вирішення малорозмірних задач не варто використовувати складні з точки зору обчислень методи та шукати рішення у аналітиці;

У процесі проектування програмного забезпечення мовою Python було розроблено низку функцій для управління процесом навчання програмного агента. Використано низку паттернів програмування, а саме: Одинак, Стратегія, Команда та Фабрика. Використано засоби графічної бібліотеки Tkinter для створення графічного інтерфейсу користувача. В розробленому інтерфейсі можна налаштовувати низку параметрів, що регулюють стратегію, середовище та тривалість тренування агента.

Результатом роботи є програма, яку можна використовувати у цілях дослідження алгоритмів навчання з підкріпленням різної складності та масштабу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* + 1. Bin Ning, Xuewei Li, “A comprehensive information system for railway networks,” Department of Control Engineering, August 2015, pp.1-8.
    2. M. Hafez Fahmy, H.Hemeda, “Computer applications in railway operation,” Alexandria Engineering Journal, Vol. 55, Issue 2, June 2016, Pages 1573- 1580.
    3. Petersen, C.D., “An Introduction to Computer Assisted Train Dispatch, ” Transportation Science, Vol. 3, No. 1, May 1996, pp. 85-91.
    4. Petersen, E.R. and Taylor, A.J., “A Structural Model for Rail Line Simulation and Optimization,” Transportation Science, Vol. 16, No. 2, May 1982, pp. 192-205.
    5. Petersen, E.R. and Taylor, A.J., “Line Block Prevention in Rail Line Dispatch and Simulation Models,” INFOR, Vol. 21, No. 1, February 1983, pp. 46-51.

ІНТЕРНЕТ РЕСУРСИ

* + 1. Автоматизовані системи диспетчерського контролю.<http://khartep.com/en/article/view/id/14/>
    2. Алгоритми керування потоком потягів станції.<https://link.springer.com/chapter/10.1007/978-3-319-72153-8_12>.
    3. Диспетчер залізниці. <https://en.wikipedia.org/wiki/Train_dispatcher>
    4. Програмне забезпечення диспетчерської системи залізничної станції.<https://www.sintef.no/en/software/train-dispatcher/>
    5. Системи управління залізничними мережами Європи. [https://www.mobility.siemens.com/mobility/global/en/interurban- mobility/rail-solutions/rail-automation/train-control-system/european-train- protection-system/Pages/european-train-control-system.aspx](https://www.mobility.siemens.com/mobility/global/en/interurban-mobility/rail-solutions/rail-automation/train-control-system/european-train-protection-system/Pages/european-train-control-system.aspx)

# ДОДАТКИ

*Додаток А. Структурна схема програми*

Головне меню



Режим диспетчера

Вихід

Режим пасажира

Додати потяг у базу

Інформація про посадку

Корегувати параметри потяга

Інформація про стан потягів

Видалити потяг з бази

Надрукувати базу

Повернення у головне меню

Управління потоком потягів

Реєстрація надзвичайних випадків

Повернення у головне меню