КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення

на тему:

ВИКОРИСТАННЯ ГЛИБИННОГО Q-НАВЧАННЯ ДЛЯ РОЗРОБКИ ІГОР

Виконав студент 4-го курсу
Артемій МИСЕЧКО _____
(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Костянтин ЖЕРЕБ _____
(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«28» травня 2021 р.,
протокол № 14
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 44 сторінки, 10 рисунків, 14 використаних джерел, 1 таблиця, 1 додаток.

2D ШУТЕРИ, Q-НАВЧАННЯ, ГЛИБИННЕ Q-НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ

Об’єктом дослідження є алгоритми навчання з підкріпленням з використанням глибинної згорткової нейронної мережі (з рекурентністю та без), які по зображенню гри, надають можливість визначити дію для NPC ворогів. Для побудови середовища для Q-навчання використовувалися бібліотеки, як OpenCV, Tensorflow та Keras на мові програмування Python. Для створення інтерфейсу взаємодії з користувачем було використано бібліотеку PyQT5.

Метою кваліфікаційної роботи є створення допоміжного додатку для розробників ігор, яка по заданим розробником параметрам буде надавати йому нейронну мережу, що буде передбачати яку дію має зробити штучний суперник у грі розробника.

Результати кваліфікаційної роботи: розглянуто сферу 2D ігор, а саме створення NPC для цікавого проходження для гравця; узагальнено 2D ігри шутери з виглядом зверху та збоку до загального шаблону; створено глибинну Q рекурентну нейронну мережу (DQRN) та глибинну Q нейрону мережу (DQN) для керування NPC; створено програмний продукт для взаємодії розробника ігор з алгоритмом навчання нейронних мереж, для отримання ново навченої DQRN чи розширеної DQN.

Сфера застосування: ігрова індустрія на даний момент є однією з популярних у світі серед розробників і зараз створюється дуже багато ігор. Для спрощення розробки, щоб не створювати нові складні алгоритми для NPC, можна застосовувати додаток для отримання глибинної нейронної мережі, яка буде сама реалізовувати поведінку для NPC, і розробнику потрібно тільки буде інтегрувати нейронну мережу у свої гру, що набагато простіше, ніж створювати

свою власну нейронну мережу чи створювати власний алгоритм для керування NPC.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ ІГОР-ШУТЕРІВ 9
1.1 Аналіз 2D ігор шутерів з виглядом зверху та збоку 10
РОЗДІЛ 2. ГЛИБИННЕ Q-НАВЧАННЯ 13
2.1 Процес прийняття рішень Маркова 13
2.2 Q-навчання 14
2.3 Глибинне Q-навчання 15
2.4 Частково спостережуваний процес прийняття рішень Маркова 16
2.5 Глибинне рекурентне Q-навчання та розширена DQN 16
2.6 Аналіз використання глибинного Q-навчання з 2D шутер-іграми з виглядом збоку та зверху 17
РОЗДІЛ 3. ПОБУДОВА СЕРЕДОВИЩА ДЛЯ ГЛИБИННОГО Q-НАВЧАННЯ 19
3.1 Визначення середовища 19
3.2 Визначення персонажів та агентів середовища 20
3.3 Перевід гри розробника у середовище для Q-навчання 20
3.3.1 Місцевість 21
3.3.2 Зброя 22
3.3.3 Персонажі 22
РОЗДІЛ 4. ПОБУДОВА МОДЕЛІ DQRN 24
4.1 Згорткова нейронна мережа 24
4.2 Довга короткострокова пам’ять 24
4.3 Побудова розширеної DQN 25
4.4 Побудова DQRN 26
4.4.1 Згорткова частина DQRN 26
4.4.2 LSTM частина DQRN 26
РОЗДІЛ 5. ЗАСТОСУВАННЯ DQRN ДЛЯ АГЕНТІВ 28

5.1 Множина дій для агентів 28
5.1.1 Множина дій для під-агентів пересування 28
5.1.2 Множина дій для під-агентів контролю 29
5.2 Множина нагород 30
5.2.1 Множина нагород для під-агенту пересування 30
5.2.2 Множина нагород для під-агенту контролю пострілів 31
5.3 Застосування DQRN та розширеної DQN 31
РОЗДІЛ 6. РОЗРОБКА ДОДАТКУ ДЛЯ ВЗАЄМОДІЇ З РОЗРОБНИКОМ ГРИ 33
6.1 Розробка модуля з DQRN 33
6.2 Класи для DQRN 33
6.3 Додаток для взаємодії з модулем DQRN 34
РОЗДІЛ 7. ТЕСТУВАННЯ СЕРЕДОВИЩА 37
ВИСНОВКИ 39
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 41
ДОДАТОК А 43

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

2D – Two-dimensional, Двовимірний;

CNN – Convolutional Neural Network, Згорткова нейронна мережа;

DQN – Deep Q Network, Глибинна Q мережа;

DQRN – Deep Q Recurrent Network, Глибинна Q рекурентна мережа;

fps – Frames per second, фрейми на секунду;

LSTM – Long short-term memory, Довга короткострокова пам’ять;

MDP – Markov Decision process, Процес прийняття рішень Маркова;

NPC – Non-Player Character, Неігровий персонаж;

POMDP – Partially Observable Markov Decision process, Частково спостережуваний процес прийняття рішень Маркова;

RGB – Red green blue, Червоний зелений блакитний;

RNN – Recurrent Neural Network, Рекурентна нейронна мережа;

ВСТУП

Розробники створюють ігри від початку створення перших програм. На даний момент сфера розробки ігор стала однією з прибуткових у світі. Одним з варіантів розробки є створення 2D шутер (з англ. «shooter») ігор. У таких іграх часто треба створювати NPC ворогів, які будуть грати проти справжнього гравця. Серед різних таких ігор більшість таких NPC мають однакові алгоритми взаємодії зі світом та гравцем, але мають обмежені можливості і через це часто не можуть створювати складності для гравця.

Актуальність роботи та її підстави для виконання. На даний момент 2D ігри активно розробляються в світі і задача загального штучного інтелекту для ігор є відкритою. Більшість розробників мають писати власні алгоритми штучного інтелекту для NPC під кожну гру окремо, але загального автоматизованого алгоритму поки не існує.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення допоміжного додатку для розробників ігор, яка по заданим розробником параметрам буде надавати йому нейронну мережу, що буде передбачати яку дію має зробити штучний суперник у грі розробника. Для її побудови були розв’язані такі завдання:

- огляд відомих 2D шутер ігор;
- аналіз зв’язку глибинного Q-навчання з іграми;
- аналіз різних алгоритмів глибинного Q-навчання;
- побудова середовища для глибинного Q-навчання;
- розробка додатку для взаємодії з користувачем;
- тестування додатку та середовища.

Об’єкт, методи й засоби дослідження або розроблення. Об’єктом дослідження є алгоритм навчання з підкріпленням з використанням глибинної згорткової нейронної мережі (з рекурентністю та без), яка по зображенню гри,

надає можливість визначити дію для NPC ворогів. Для побудови середовища для Q-навчання використовувалися бібліотеки, як OpenCV [1], Tensorflow [2] та Keras [3] на мові програмування Python. Для створення інтерфейсу взаємодії з користувачем було використано бібліотеку PyQT5 [4].

Сфера застосування. Будь-який розробник 2D шутер гри з виглядом зверху та збоку зможе використовувати додаток для отримання нейронних мереж, які можна буде використати у власній грі і не прописувати власний алгоритм для NPC ворогів.

РОЗДІЛ 1. АНАЛІЗ ІГОР-ШУТЕРІВ

Шутер – це піджанр екшн-відеоігор (з англ. «action»), що фокусується на перемозі ворогів за допомогою використання зброї, наданої гравцеві. Зазвичай надається зброя далекої дії і її можна використовувати у поєднанні з іншими інструментами, такими як броня для захисту, гранати для додаткового нападу чи інші аксесуари.

Ігри шутери перевіряють просторову орієнтацію гравця, рефлекси та швидкість реагування в ізольованому середовищі. Шутери охоплюють багато піджанрів, які мають спільність фокусування на діях гравця, що вступає в бій зі зброєю проти ворогів NPC, які керуються кодом.

З усіх видів шутерів можна відокремити п’ять основних:

- перестріляй їх усіх (з англ. «Shoot’em up») – жанр відеоігор, в якому гравець може рухатися вверх, вниз, вправо, вліво по усьому екрану, зазвичай стріляючи по NPC ворогам тільки в гору;
- галерея стрільби (з англ. «Shooting gallery») – жанр відеоігор, в якому гравець має цілитися у цілі, які рухаються, на стаціонарному екрані;
- лазерний шутер (з англ. «Light gun shooter») – жанр відеоігор, в якому використовується пристрій з лазером для комп’ютерів або ігровий контролер;
- шутер від першої особи – жанр відеоігор, який характеризується видом на екрані, який симулює точку зору ігрового персонажу, та можливістю пересування по світу в будь-яких напрямках;
- шутер від третьої особи – жанр відеоігор, який характеризується видом камери від третьої особи, що гравець може бачити повністю свого ігрового персонажа та навколишнє середовище, та можливістю пересування по світу в будь-яких напрямках.

1.1 Аналіз 2D ігор шутерів з виглядом зверху та збоку

З поданих вище п’яти видів ігор-шутерів, можна виділити підвиди як 2D шутери з виглядом зверху та збоку.

2D шутер з виглядом зверху (збоку) – піджанр відеоігор, який характеризується видом камери зверху (збоку) від ігрового персонажа, в якому гравець може рухатися вверх, вниз, вправо, вліво (для збоку – вправо, вліво та може стрибати вверх чи падати вниз) по усьому екрану. Основною метою гравця зазвичай є набрання більше очок за певну кількість вбитих NPC ворогів, за виконання завдань та/чи можливості дійти до певного пункту призначення.

Прикладом гри-шутера з виглядом зверху може бути The Binding of Isaac [5] (див. рис. 1.1). В цій грі гравець має контролювати персонажа Ісака чи іншого персонажа для проходження підземель в підвалі Ісака. У кожного персонажа варіюється кількість здоров’я, швидкість, пошкоджень, які може наносити персонаж. В кожному поверсі підземелля гравець має боротися проти монстрів у кожній кімнаті, щоб просуватися по ньому. Боротьба зазвичай проходить далекобійними атаками. В процесі проходження гравець може покращувати свого ігрового персонажа, щоб можна було легше проходити наступні рівні.

Рисунок 1.1 – Знімок екрану ігрового процесу The Binding Of Isaac, який показує атакуючого Ісака (по центу) та двох ворогів (зверху справа та зліва)

Прикладом гри-шутера з виглядом збоку може бути Noita [6] (див. рис. 1.2). В цій грі гравець контролює персонажа чарівника, який може створювати та чаклувати заклинання для того, щоб перемогти ворогів, названих в честь фінських міфічних істот. Цей світ є відкритим і гравець повинен боротися з ворогами, зазвичай на відстані, щоб вижити у світі гри.

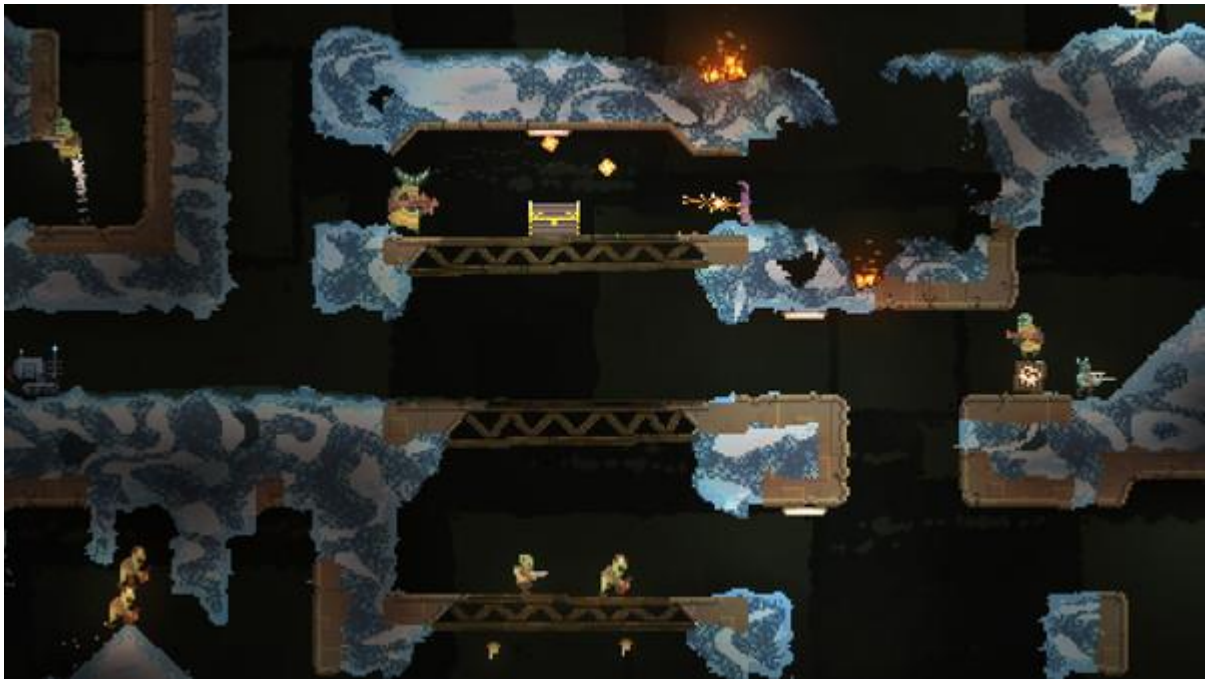


Рисунок 1.2 – Знімок екрану ігрового процесу Noita, який показує чаклуна зі зброєю (по центу справа від скрині) та десять ворогів навколо на карті

Як можна помітити, в таких іграх часто наявні ворожі NPC, з якими треба боротися і це правило розповсюджується на більшість ігор такого типу. Під час написання 2D гри-шутера з виглядом зверху чи збоку розробнику постає задача написання поведінки для ворожих NPC. Для елементарної поведінки ворожого NPC, наприклад, можуть застосовувати дерево ухвалення рішень або щось подібне до цього, але це не створює великої складності для гри гравця. Для складної поведінки NPC розробнику ігор треба буде звертатися до написання складних алгоритмів або алгоритмів машинного навчання, що є важкою роботою та забирає багато часу. Цей процес розробки можна автоматизувати і розробник зможе використовувати готові нейронні мережі для свої гри. Для цього треба виділити спільні частини 2D шутер ігор з видом збоку та зверху:

а) орієнтація в просторі – для вигляду зверху ми можемо рухатися вверх, вниз, вправо, вліво, для вигляду збоку, з’являється гравітація і пересування вверх та вниз, замінюються на стрибки та падіння;

б) місцевість – є зони, в які не можливо потрапити, які важко проходити, які мають негативний вплив на персонажа та NPC (можуть загинути);

в) постріли – можна поділити на 2 основні типи:

1) снаряди летять по прямій;

2) снаряди летять по дузі.

По поданим узагальненням можна створити систему, яка буде створювати по заданим параметрам агентів для керування NPC. Розробник зможе інтегрувати нейронні мережі у власний проект і зможе використовувати їх для керування персонажами.

Для використання алгоритму глибинного Q-навчання для агенту потрібно визначити чи мають ігри такого типу основу у вигляді процесу прийняття рішень Маркова.

РОЗДІЛ 2. ГЛИБИННЕ Q-НАВЧАННЯ

2.1 Процес прийняття рішень Маркова

Процес прийняття рішень Маркова (з англ. «Markov Decision Process») – дискретний від часу стохастичний процес керування, що надає можливість знайди оптимальний розв'язок для системи [7].

MDP складається з:

– множини станів середовища *S*, що визначається, як скінченна множина $S = \{s^1, ..., s^N\}$ розміру *N*, де кожен стан є унікальною характеристикою, що описує всі важливі деталі змодельованої проблеми;

– множини дій у середовищі *A*, що визначається, як скінченна множина $A = \{a^1, ..., a^K\}$ розміру *K*. Дії можуть використовуватися для контролю станів системи. Множину дій можна застосувати на визначений стан $s \in S$, що позначається як $A(s)$, де $A(s) \subseteq A$. Не в кожній системі можна застосувати кожну дію на кожний стан, але в загальному вважаємо, що $A(s) = A$ для будь-якого $s \in S$;

– функції переходів – під час застосування дії $a \in A$ для стану $s \in S$, система робить перехід з s в новий стан $s' \in S$. Функція переходу T визначається як $T : S \times A \times S \to [0,1]$, що є вірогідністю закінчити в стані s' після виконання певної дії a в стані *s*, що визначається як $T(s,a,s')$. При цих складових система вважається марковою, якщо результат поточної виконаної дії не залежить від результатів минулих дій та відвіданих станів, тобто:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1} \mid s_t, a_t) = T(s_t, a_t, s_{t+1});$$

– функції нагород – визначає нагороди за находження у станах, або виконання певної дії в станах. Функція нагород для стану позначається як $R : S \to \Re$, що визначає кількість отримання нагород у стані. Також є ще функції нагород як $R : S \times A \to \Re$ та $R : S \times A \times S \to \Re$. Перша надає

кількість нагород за дію, а друга за певний перехід.

Отже можна навести, ще одне визначення процесу прийняття рішень Маркова: це кортеж $\langle S, A, T, R, \Omega, O \rangle$, де S – скінченна множина станів, A – скінченна множина дій, T – функція переходів, що визначена як $T : S \times A \times S \to [0,1]$, та R – функція нагород, що визначена як $R : S \times A \times S \to \Re$.

По заданому MDP, можна визначити функцію лінії поведінки, яка надає для будь-якого стану дію для виконання. Детермінована функція лінії поведінки визначається як $\pi : S \to A$. Стохастична функція лінії поведінки визначається як $\pi : S \times A \to [0,1]$, де для будь-якого стану $s \in S$ має виконуватися, що $\pi(s, a) \geq 0$ та $\sum_{a \in A} \pi(s, a) = 1$.

2.2 Q-навчання

Q-навчання – форма безмодельного навчання з підкріпленням (з англ. «reinforcement learning»), може розглядатися як метод асинхронного динамічного програмування [8]. Надає обчислювальних агентів з можливістю навчання, які ведуть себе оптимально на MDP, отримуючи досвід від результатів виконаних дій, без потреби побудови карт доменів прийняття рішень.

Вважаємо, що обчислювальний агент рухається по дискретному скінченному світу, обираючи кожного разу одну дію на кожному кроці часу. Світ є контрольованим MDP з агентом, як контролером. На кожному кроці n агент може зафіксувати стан $s_n \in S$ та обрати дію $a_n \in A$, після того отримує вірогідну оцінку r_n, у якої математичне сподівання $M_{s_n}(a_n)$ залежить лише від стану та дії (через властивості функції переходу MDP), і стан світу змінюється вірогідно на $s'_n \in S$.

Тоді за [8] отримуємо, що в Q-навчанні досвід агенту складається з послідовності виразних етапів. На кожному етапі n, агент:

– спостерігає поточний стан $s_n \in S$;

– обирає та виконує дію $a_n \in A$;

– спостерігає наступний стан $p_n \in S$;

– одразу отримує нагороду r_n;

– та вирівнює значення Q_{n-1} за допомогою фактору $0 < \alpha_n < 1$:

$$Q_n(s,a) = \begin{cases} (1-\alpha_n)\, Q_{n-1}(s,a) + \alpha_n [r_n + \gamma\, V_{n-1}(p_n)] & | \quad x = x_n \wedge a = a_n \\ Q_{n-1}(s,a) \end{cases}$$

де γ – фактор вирівнювання минулого значення нагороди $0 < \gamma < 1$, та

$$V_{n-1}(p) \equiv \max_b \{Q_{n-1}(p_n, b)\}$$

що визначається, як найкраще, що може зробити агент в стані p_n.

До першого етапу Q-навчання усі значення $Q_0(s,a)$ вважаються випадково визначеними. Такий опис алгоритму визначає, що $Q_n(s,a)$ є таблицею, а інші варіанти за [9] можуть не сходитися. Також за теоремою [8] при певних умовах було визначено, що $Q_n(s,a) \to Q_n^*(s,a)$, при $n \to \infty$, де $Q_n^*(s,a)$ оптимальний розв'язок системи.

2.3 Глибинне Q-навчання

Глибинне Q-навчання (з англ. «Deep Q-learning») – Q-навчання, яке в основі використовує згорткові нейронні мережі замість використання таблиць [10]. Також застосовує техніку повторного досвіду, що для кожного агенту зберігає на кожному кроці досвід у вигляді значення $e_t = (s_t, a_t, r_t, s_{t+1})$ в датасет (з англ. «dataset») $D = e_1, ..., e_N$ об'єднаний у величезну кількість епізодів в пам'ять повторення. Під час головного алгоритму, визначаються оновлення для одного Q значення, або за кожен міні-батч (з англ. «mini-batch») для вибірки з множини D, обраний випадковим чином з усієї вибірки. Після виконання повторного досвіду агент обирає дію використовуючи θ-жадібну функцію лінії поведінки (з вірогідністю $1-\theta$ обирається дія з найбільшим коефіцієнтом Q для стану s, інакше обирається випадкова можлива дія для стану s). Оскільки

використання довільної довжини кількість епізодів може бути складним для нейронної мережі, то Q-функція працює на обмежену кількість кроків.

2.4 Частково спостережуваний процес прийняття рішень Маркова

Частково спостережуваний процес прийняття рішень Маркова (з англ. «Partially Observable Markov Decision process») є узагальненням MDP, моделює агента процесу прийняття рішень, у якому визначається, що динаміка системи визначена MDP, але агент не може спостерігати безпосередньо поточний стан системи [11]. Агенту надається можливість отримати стохастичний розподіл різних спостережень наданих для поточного стану та MDP. Тобто у даному випадку до основного кортежу MDP $\langle S, A, T, R \rangle$ додається ще множина спостережень станів середовища Ω та стохастичний розподіл O. Агент отримує спостереження для поточного стану s, $o \in \Omega$, яке генерується з системи, що має в основі $o \sim O(s)$.

Звичайне Q-навчання не має точних методів отримання визначеного стану для POMDP і ефективне у випадку, якщо спостереження є відображенням станів системи. У загальному випадку вважається, що $Q(o, a) \neq Q(s, a)$.

2.5 Глибинне рекурентне Q-навчання та розширена DQN

Глибинне рекурентне Q-навчання – глибинне Q-навчання з використанням рекурентних нейронних мереж [12]. Таким чином, як спостереження для поточного стану, можна надавати певну кількість станів від часу до поточного. Як частину нейронної мережі, можна використовувати архітектуру довгої короткострокової пам’яті (з англ. «Long short-term memory») при цьому інших змін у алгоритмі глибинного Q-навчання не потрібно (детальніше структура DQRN подана у підрозділі 4.4).

Іншим варіантом реалізації алгоритмів навчання для POMDP може бути

розширена архітектура DQN без використання рекурентних мереж. Тобто вона працює аналогічно до DQN визначеної, у підрозділі 2.3, але на вході замість одного стану будуть подаватися N, які відповідають певному спостереженню. Таким чином потрібно лише збільшити лише вхідну розмірність (детальніше структура подана у підрозділі 4.3).

2.6 Аналіз використання глибинного Q-навчання з 2D шутер-іграми з виглядом збоку та зверху

В 2D шутер-ігри з виглядом збоку і зверху можна віднести до POMDP за таких причин:

– середовище: гра – це середовище для гравця та NPC, з визначеною на перед, або випадково згенерованою місцевістю для взаємодії з ним;

– наявність станів – кожен фрейм гри можна визначити, як окремий стан s середовища. Фрейми обмежені за розміром екрану для гравця та розміром площі видимості для NPC. Оскільки кількість типів місцевостей в іграх обмежена, обмежено озброєння гравця та NPC, то кількість різних фреймів з позицією гравця, NPC, варіацій місцевості та варіацій озброєнь персонажів теж скінченна, тобто кількість станів скінченна кількість;

– спостереженнями у даному випадку вважається визначена кількість вибраних станів за певною умовою (детальніше розділ 5), які йдуть послідовно за часом;

– наявність дій – гравець та NPC можуть рухатися в усіх напрямках з визначеною наперед швидкістю по місцевості, тобто для ігор з виглядом зверху: вправо, вліво, вверх, вниз, вправо-вверх, вправо-вниз, вліво-вверх, вліво-вниз; для ігор з виглядом збоку: вправо, вліво, стрибок вверх, падіння вниз (за наявності, куди можна падати), стрибок вверх-вправо, падіння вправо-вниз, стрибок вліво-вверх, падіння вліво-вниз. Також гравець та NPC можуть одночасно стріляти зі зброї на 360°,

оскільки кількість точок на місцевості скінченна кількість (кількість пікселів на фреймі), то кількість можливих пострілів обмежена, тобто маємо, що кількість дій в середовищі скінченна кількість;

– функція переходів між станами реалізується фізикою світу, і кожна дія веде до переходу у якийсь існуючий стан у системі;
– функція нагород – основна мета гри-шутера це вбити суперника, тобто потрапити зі зброї у суперника, тому можна визначити, що функція, нагород дає позитивне значення, якщо у поточному стані куля, яку вистрілили зі зброї, знаходилась на позиції суперника, інакше від'ємне, щоб змушувати майбутнього агента діяти (детальніше підрозділ 5.2).

Оскільки визначено кортеж, який складається з множини станів, множини дій, функції переходів та функції нагород, спостережень, то системи 2D шутер ігор з виглядом зверху та збоку є POMDP. Тоді можна застосувати алгоритм глибинного Q рекурентного навчання для агентів, які будуть керувати NPC.

Оскільки система шукає оптимальну функцію лінії поведінки для перемоги над суперником, то результуючу нейронну мережу, можна буде застосувати у ігровій системі, яка буде грати проти людини на її ж рівні, якщо не перевершувати.

РОЗДІЛ 3. ПОБУДОВА СЕРЕДОВИЩА ДЛЯ ГЛИБИННОГО Q-НАВЧАННЯ

3.1 Визначення середовища

Середовище буде надаватися у вигляді піксельного зображення на вхід алгоритму агента глибинного Q-навчання, який у відповідь буде надавати дію, яку має виконати NPC чи гравець. Вважаємо, що піксельне зображення RGB формату та розміру 42x42 (для інших розмірів див. пункт 3.3.1). Кожний колір на зображені несе тільки потрібну інформацію для глибинного навчання (див. рис. 3.1), а саме:

– блакитний колір (в форматі RGB: (0, 175, 255)) – поле з гравцем;
– червоний колір (в форматі RGB: (255, 0, 0)) – поле з ворожим NPC;
– чорний колір (в форматі RGB: (0, 0, 0)) – місцевість є вільною для пересування персонажів;
– білий колір (в форматі RGB: (255, 255, 255)) – місцевість є непроходимою для персонажів;
– помаранчевий колір (в форматі RGB: (255, 69, 0)) – місцевість, яка негативно впливає на персонажа (персонаж помирає/програє, якщо її торкається);
– салатовий колір (в форматі RGB: (153, 255, 153)) – місцевість крізь яку можуть пролітати тільки снаряди зброї;
– жовтий колір (в форматі RGB: (255, 255, 0)) – колір снарядів гравця;
– рожевий колір (в форматі RGB: (255, 0, 255)) – колір снарядів ворожого NPC.

Подальше використання кольорів стосуватиметься тільки цієї класифікації.

Рисунок 3.1 – Приклад місцевості з персонажами та кулями

3.2 Визначення персонажів та агентів середовища

У середовищі обов’язково повинен бути присутній один гравець та один ворожий NPC, інакше, якщо вони будуть поодинці, то не буде ворога для агенту і глибинний навчання у такому випадку ніяких властивостей не визначить. Вважаємо, що гравцем та ворожим NPC керують два різних агенти за алгоритмом Q-навчання. Агенти будуть змагатися один з одним, для кожного з них визначена однакова система нагород. Навчання виконується не в режимі реального часу, тобто по заданих параметрах будуть симулюватися різні ситуації, в яких випадковим чином агентам надаються початкові позиції, і в залежності від поточного стану агент буде виконувати дії за правилами алгоритму Q-навчання.

3.3 Перевід гри розробника у середовище для Q-навчання

Усі подані параметри у пунктах 3.3.1-3.3.3 цього розділу розробник зможе

надати за допомогою розробленого додатку (детальніше розділ 6).

3.3.1 Місцевість

При розробці 2D ігор-шутерів розробник створює місцевості для рівнів. Всі рівні можна звести до піксельного представлення (картинка, у якої кожен піксель несе тільки потрібну інформацію), щоб при навчанні нейронної мережі не було надлишкової інформації. Тобто, щоб розробник зміг навчити на власній місцевості нейронну мережу йому достатньо перевести її у місцевість з пікселів за такими умовами (всі подальші кольори розписані детальніше у підрозділі 3.1):

– вважається, що гравець та ворожий NPC займають рівно 1 піксель на карті (задавати на власній мапі не потрібно), тобто усі подальші співвідношення розмірів до персонажів у справжній грі мають зберігатися до розмірів персонажів у піксельній місцевості;
– місцевість вільного пересування має позначатися чорним кольором;
– місцевість де можливо просунутися має позначатися білим кольором;
– за наявності місцевості, що негативно впливає на персонажів (смерть), позначати її помаранчевим кольором;
– за наявності місцевості, через яку можуть пролітати снаряди зброї, позначати її салатовим кольором.
– за наявності інших місцевостей з власними ефектами, наближаємо їх, до вже існуючих (за подальшої розробки додатку список повинен буде розширюватися для більш унікальних випадків).

Мінімальні розміри піксельної місцевості мають бути 42x42. Така розмірність була обрана оскільки у даному випадку персонажам не знадобиться довго шукати один одного, оскільки такий пошук буде сповільнює роботу алгоритму. Також дуже завантажені мапи на величезну кількість об’єктів впливають на продуктивність алгоритму, що потрібно буде більше часу для навчання, а при такій розмірності багато об’єктів не розташувати.

У випадку, якщо місцевість менше, ніж 42x42, то заповнюємо її до

потрібного розміру білим кольором, що є обмежуючим для пересування і не буде впливати на роботу алгоритму. У випадку, якщо піксельна місцевість більше ніж 42x42, достатньо порозбивати на менші місцевості по 42x42 випадковим чином, але бажано, щоб не було відокремлених територій, щоб персонажі могли дістатися один одного.

Оскільки маємо 3 канали кольорів (червоний, зелений та синій), то під час роботи додатку піксельна місцевість буде зберігатися у вигляді матриці 42x42x3, яка буде подаватися на вхід алгоритму Q-навчання.

3.3.2 Зброя

У 2D шутер іграх присутні різні типи зброї. Можна розділити зброю на 2 типи:

– снаряд після вистрілу летить по прямій;

– снаряд після вистрілу летить по дузі.

Кожна зброя має такі властивості:

– час перезарядки зброї – як часто можна робити постріл;

– швидкість снарядів – для розробника треба буде перевести швидкість у відповідну швидкість (розмірність пікселі на секунду) по піксельній карті.

3.3.3 Персонажі

Персонажами гри у даному випадку є гравець та ворожий NPC, якими, як було сказано у підрозділі 3.2, керуватимуть агенти глибинної мережі. Для персонажів важливі такі параметри:

– швидкість пересування – для розробника треба буде перевести швидкість у відповідну швидкість (розмірність пікселі на секунду) по піксельній карті;

– висота стрибка у випадку гри збоку – для розробника треба буде

перевести довжину у відповідну піксельну довжину у піксельній карті.

– зброя, якою володіє персонаж (особливості зброї визначені у пункті 3.3.1 цього розділу).

РОЗДІЛ 4. ПОБУДОВА МОДЕЛІ DQRN

4.1 Згорткова нейронна мережа

Згорткова нейрона мережа (з англ. «convolutional neural network») – це спеціальний вид нейронної мережі для обробки даних, яка в основі має математичне поняття згортки. Основна суть полягає, що замість загальної операції множення на матрицю використовується принаймні в одному шарі згортка вигляду [13, с. 282-284]:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m,n)K(i-m, j-n),$$

де $S(i, j)$ – результат згортки на позиції i, j ,

I – вхідна матриця,

J – ядро згортки,

m, n – розмірність ядра згортки.

4.2 Довга короткострокова пам’ять

Довга короткострокова пам’ять – це одна з ефективних моделей послідовностей вентильних рекурентних нейронних мереж (з англ. «recurrent neural network») [14, с. 344-347]. RNN мають проблему в обчисленнях, що при методі зворотного поширення помилки (з англ. «backpropagation») градієнти можуть перетворюватися на нуль чи прямувати до нескінченності через те, що в обчисленнях використовуються числа з фіксованою точністю. У свою чергу у LSTM уникає цих проблем завдяки тому, що не дає можливості градієнтам перетворитися на нуль або прямувати до нескінченності, за допомогою чотирьох вентилів керування пам’яттю. LSTM має в собі блоки з витоком (петлі), які мають особливість накопичувати інформацію протягом довгого часу, але з можливістю затирання забування застарілих властивостей даних.

4.3 Побудова розширеної DQN

У розширену DQN подається спостереження, яке складається з N послідовних від часу станів, які є піксельним зображеннями розміру 42x42x3 зі значеннями від 0 до 255. Перед поданням у DQN будуть дані потрібно нормалізувати. Тобто розмірність кожного елементу при поданні у DQN це Nx42x42x3.

У свою чергу нейронна мережа спочатку перетворює дані у розмірність 42x42x(3*N), та подає їх у 2 послідовних згорткових шари. Після останнього згорткового шару дані перетворюються у вектор, які переходять в останній шар мережі, який надає на виході K_Q ваги, серед яких можна буде за алгоритмом Q-навчання визначити максимальну вагу та виконати відповідну дію для поточного стану (див. рис. 4.1).

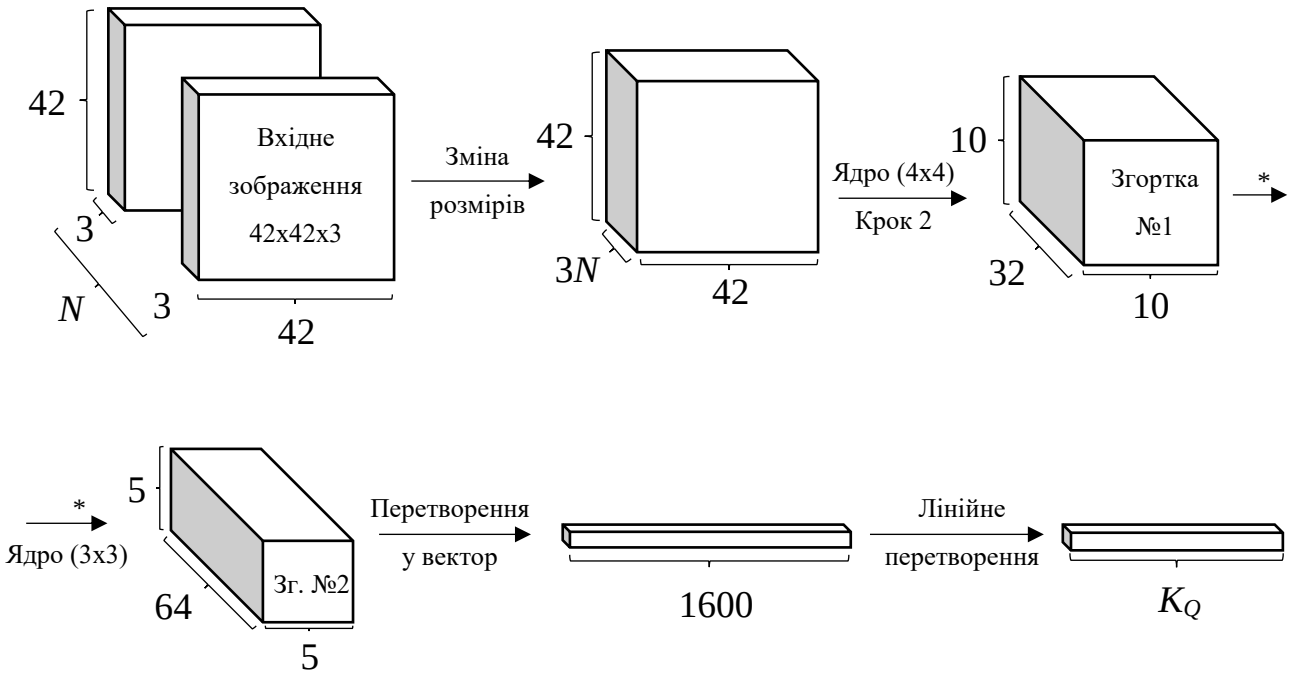


Рисунок 4.1 – Схема розширеної DQN (числа вказують розмірності, скорочення: Зг. – згортка)

4.4 Побудова DQRN

Модель DQRN можна поділити на 2 частини. Перша частина відповідає за використання згорткових нейронних мереж для отримання стиснутих властивостей з вхідного піксельного зображення гри. Друга частина це безпосередньо LSTM, на вхід якої будуть подаватися стиснуті зображення з першої частини, в залежності від часу, і на виході будуть отримані для поточного спостереження Q-значення для агента.

4.4.1 Згорткова частина DQRN

Оскільки всі поточні стани, це піксельне зображення розміру 42x42x3, зі значеннями від 0 до 255, то на вхід згорткової мережі буде подаватися матриця такого ж розміру, але з нормалізацією даних. Сама нейронна мережа складається з 3х згорткових шарів. Кожен згортковий шар має ядро розміру 3x3, після кожного з них застосовується операція пулінгу 2x2 (див. рис. 4.2). Останній шар надається на вхід LSTM.

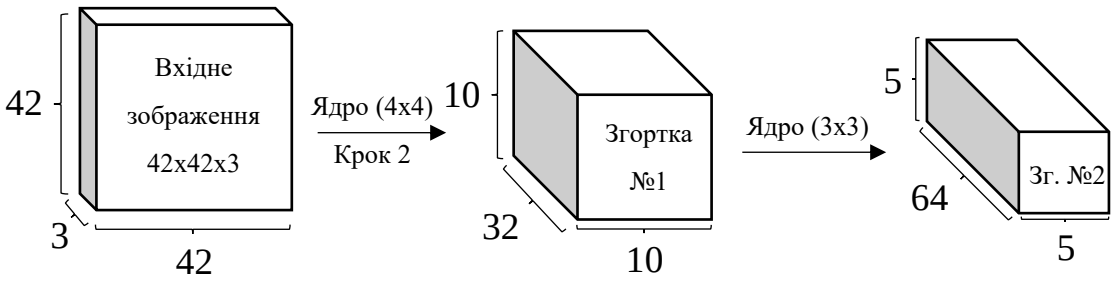


Рисунок 4.2 – Схема згорткової нейронної мережі. (числа вказують розмірності, скорочення: Зг. – згортка)

4.4.2 LSTM частина DQRN

Для поданих вихідних зображень зі згорткової частини буде застосовано підвид LSTM, як згорткова LSTM, у якій у якості клітини будуть використані

згорткові клітини. На вхід моделі подається спостереження від часу з N станами. Таким чином остаточна частина DQRN складається шару LSTM. На виході з нього отримується матриці, які об'єднуються в один вектор. На нього застосовується лінійна функція і на виході отримуємо вектор розміру на кількість вихідних Q-значень (див. рис. 4.3).

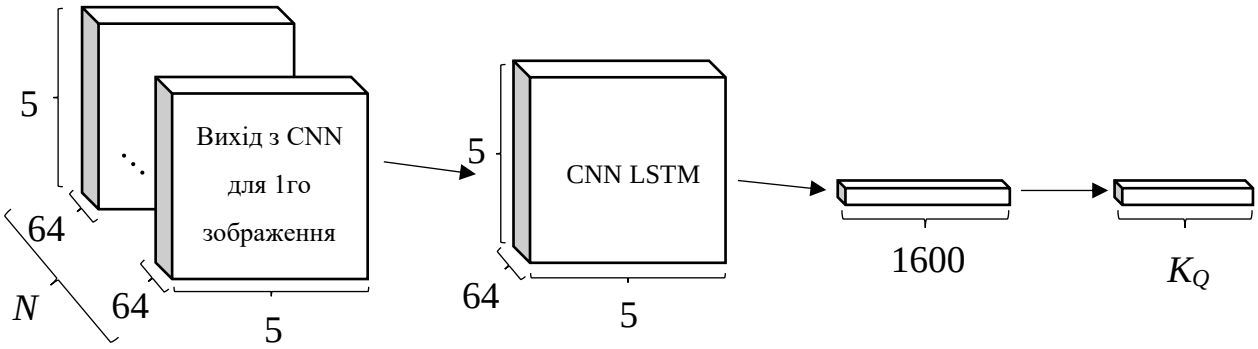


Рисунок 4.3 – Схема LSTM. (числа вказують розмірності, N – кількість станів в одному спостережені, K_Q – кількість можливих Q-значень на виході)

Таким чином на вхід нейронної мережі подається N послідовних за часом піксельних зображень, кожне з яких обробляється за допомогою CNN, та передається на вхід до LSTM. У свою чергу, LSTM надаватиме на виході K_Q ваги, серед яких можна буде за алгоритмом Q-навчання визначити максимальну вагу та виконати відповідну дію для поточного стану.

РОЗДІЛ 5. ЗАСТОСУВАННЯ DQRN ДЛЯ АГЕНТІВ

Вважаємо, що швидкість оновлення фреймів у середовищі 60fps. Оскільки людина може реагувати та взаємодіяти у середовищі приблизно кожні 80-100 мс, то вважаємо, що кожен агент може взаємодіяти кожні 5 фреймів з середовищем. На інші фрейми впливатиме лише фізика середовища.

5.1 Множина дій для агентів

Як було вказано раніше, у середовищі у нас наявні два персонажі, якими керують відповідно два агенти. У кожного агента наявні два типу дій: керування пересуванням персонажу та керування пострілами персонажу; які в свою чергу не є залежними одними від одного. Таким чином кожного агента можна розділити на два під-агенти відповідно.

5.1.1 Множина дій для під-агентів пересування

Для ігор з виглядом зверху та збоку відрізняється множина дій для пересування. У випадку гри збоку на персонажів і на певні типи куль діє сила тяжіння, таким чином у цьому випадку середовище після дії персонажа застосовує фізику сили тяжіння. Також середовище при кожному фреймі гри пересуває за вектором швидкості персонажів (якщо такий наявний) та кулі (якщо вони наявні у середовищі).

Для під-агентів керуванням пересування персонажів для гри зверху визначена така множина з шести дій:

– стояти на місці;

– пересування вліво чи пересування вправо;

– стрибок вліво, вертикально вверх, стрибок вправо.

Стрибок для персонажу можливий у випадку, якщо він стоїть на білому блоці, інакше замість стрибка персонаж буде пересуватися вправо чи вліво, чи

вертикально, але швидкість по вертикалі не буде додаватися для подолання фізики тяжіння. Також у випадку пересування вліво чи вправо, і не наявності блоку на якому можна стояти, персонаж почне падати під дією сили тяжіння.

Вектор напряму вліво та вправо, відповідає заданій швидкості розробником, вектор стрибку визначається за його висотою та напрямом.

Для під-агентів керуванням пересування персонажів для гри зверху визначена така множина з дев’яти дій:

– стояти на місці;

– пересування вліво, вправо, вверх чи вниз;

– пересування вліво-вверх, вправо-вверх, вліво-вниз, вправо-вниз.

Вектор напряму у цих випадках завжди однаковий.

Таким чином, кожні п’ять фреймів під-агент пересування зможе обирати напрямок руху.

5.1.2 Множина дій для під-агентів контролю

На визначення пострілу персонажу впливають фактори, як час перезарядки між пострілами та кут відносно осі Ox під яким стріляють зі зброї.

Множина дій для під-агенту контролю стрільби складається з 17 елементів. Вважаємо, що один елемент множини, це вибір не стріляти, інші 16 – це 360 градусів поділено на 16 рівномірних кутів, які будуть відраховуватися від осі Ox за годинниковою стрілкою і це будуть напрями куди потрібно вистрелити.

Оскільки визначено час перезарядки для пострілів у середовищі, то маємо, що під-агент при певних спостереженнях не матиме можливості вистрілити. Тому вважаємо, що під-агенту надається спостереження тоді і тільки тоді, коли він має можливість вистрілити, при цьому йому надається вікно історії за останні N_s станів, де N_s – має бути більше ніж за кількість пропущених фреймів між перезарядками, які теоретично міг би отримати під-агент (отримує він кожен п’ятий).

5.2 Множина нагород

Для роботи алгоритму Q-навчання потрібно визначити множину нагород та штрафів за кожен стан у системі. Нагороди та штрафи для кожного типу під-агенту різні. Всі нагороди та штрафи рахуватимуться в цілих числах. За нагороду під-агент отримує позитивне значення за поточну ситуацію, за штраф – негативне. Вважаємо, що якщо поточний стан визначає програш одного з персонажів, гра автоматично завершується і починається нова. Подана множина нагород визначалася більше на логічних міркуваннях. За потреби покращення множини бажано провести тестування з різними значеннями та обрати саме ту множину на якій більш результати навчання будуть найкращі.

5.2.1 Множина нагород для під-агенту пересування

Система нагород при пересуванні можна визначити таким чином:

– за те, що під-агент не рухається – штраф 3 бали, для спонукання, щоб рухався;
– за пересування – штраф 1 бал, для якісного пересування.
– якщо після пересування, персонаж торкнувся помаранчевої зони – штраф 350 балів, оскільки це миттєве закінчення гри, у якому виграш отримує суперник. Суперник не отримує за це нагороди, вважаємо, що буде штраф такий, як був за звичайне пересування чи не пересування, відповідно до його дій;
– якщо після пересування, позиція персонажу гравця, співпадає з позицією персонажу NPC, вважаємо, що гравець автоматично програє (зазвичай в іграх гравець отримує величезну кількість шкоди), тобто таким чином для персонажа гравця буде штраф 300 балів, а персонаж NPC отримує нагороду 30 балів;
– якщо після пересування, персонаж пересікся з кулею ворога, то ворог отримує нагороду 60 балів, а персонаж, який програв, отримує штраф 250

балів.

– у випадку нічиї від пострілів, під-агент отримує штраф 200 балів за загибель від пострілу та 250 балів за загибель від дотику до помаранчевої місцевості та 100 балів у іншому випадку;

5.2.2 Множина нагород для під-агенту контролю пострілів

Система нагород при пересуванні можна визначити таким чином:

– за те, що під-агент не стріляє – штраф 3 бали, для спонукання, щоб стріляв;

– за виконаний постріл – штраф 1 бал, для контролю якісного пострілу;

– якщо персонаж загинув не від пострілу, то під-агент отримує штраф 100 балів, при цьому суперник, нагороди не отримує, вважаємо, що отримує штраф, як за виконаний постріл (у випадку, якщо сам не загинув);

– якщо персонаж загинув від пострілу, то під-агент отримує штраф 250 балів, при цьому суперник, отримує нагороду 60 балів.

– у випадку нічиї від пострілів, під-агент отримує штраф 200 балів за загибель від пострілу та 50 у іншому випадку;

5.3 Застосування DQRN та розширеної DQN

Оскільки кожен під-агент має власну лінію поведінку, то для кожного з них можна визначити управління за допомогою DQRN чи розширеної DQN. Тобто маємо 4 нейронні мережі для кожного під-агенту, які навчатимуться паралельно між собою. Також у цьому випадку застосовується принцип генеративних нейронних мереж, які змагаються між собою.

Навчання виконується за фіксовану кількість епізодів (на практиці було взято 10000 для DQRN та розширеної DQN), де кожен епізод – це нова гра. На кожну гру виділяється максимум 200 кроків дій, для кожного агенту. У випадку, якщо гра не завершилась за 200 кроків, вважаємо, що це нічия, і кожен під-агент

отримує власний штраф за нічию, визначену у минулому підрозділі.

Для навчання визначено параметр $\theta = 1$, який протягом навчання буде спрямує до значення 0.05, і потім зменшуватися не буде. Цей параметр визначає вірогідність випадкового кроку для під-агенту. Оскільки з самого початку для під-агентів середовище невідоме, то для пришвидшення вивчання з самого початку будь-який рух є випадковим. Після кожного кроку у епізодах це значення зменшується і коли досягне 0.05, у 95% під-агенти будуть ходити відносно кроків власної DQRN чи розширеної DQN. Цей параметр не перетворюється в нуль задля того, щоб у під-агентів лишалась можливість періодично вивчати середовище.

Кожен під-агент зберігає власну чергу історії, яка обмежена за довжиною (до 50000 останніх спостережень), тобто кожен під-агент зберігає лише найновіші останні спостереження, які використовуються для навчання. Після того, як в історії накопичується 2000 спостережень, то DQRN чи розширена DQN навчається міні-батчами по 64 випадкових спостереження після кожного кроку середовища.

РОЗДІЛ 6. РОЗРОБКА ДОДАТКУ ДЛЯ ВЗАЄМОДІЇ З РОЗРОБНИКОМ ГРИ

6.1 Розробка модуля з DQRN

Для реалізації алгоритму глибинного *Q*-навчання було використано широковідому бібліотеку Tensorflow 2.5 від компанії Google, яка надає можливість швидко створювати нейроні мережі. В результаті розробки було створено незалежний модуль для навчання агентів для параметричного середовища та піксельного зображення карти.

6.2 Класи для DQRN

Клас Unit – зберігає інформацію про персонажа, таку як швидкість пересування, висота стрибка (за наявності), тип зброї якою володіє, її перезарядка та швидкість куль. Реалізовує механіку пересування по піксельній мапі – є вибір пересування за вектором направлення за фізичним світом та є можливість змінити напрям пересування, за допомогою множини дій. Контролює перезарядку зброї та смерть персонажу.

Клас Bullet – зберігає інформацію про кулю, а саме її вектор швидкості, посилання на ворога і на власника. Контролює перетин траєкторій між фреймами між собою та ворогом, рухається за визначеною для неї траєкторією. У випадку зіткнення з ворогом, визначає супернику тип смерті.

Клас Environment – зберігає інформацію про середовище, а саме мапу, персонажів та кулі. Реалізовує механіку виклику фізики для персонажів та для куль. Контролює закінчення гри, програш персонажів. Визначає нагороди та надає поточний стан після кожного кроку середовища. Перевіряє місце знаходження персонажів, для визначення, чи дотикаються вони помаранчевих територій. Відповідає за демонстрацію симуляції – для цього використовується бібліотека OpenCV.

Клас DQRNAgent – зберігає дві моделі нейронних мереж: для контролю пострілів та для контролю пересування. Реалізовуються черги для зберігання історії спостережень епізодів для пострілів та для пересування окремо. Реалізований механізм тренування моделей.

6.3 Додаток для взаємодії з модулем DQRN

Для можливості швидкого використання для навчання власних агентів для власних ігор, було створено додаток з використанням бібліотеки PyQt5 на мові програмування Python.

Qt – це набір крос-платформних бібліотек мови C++, які реалізовують високорівневі API для доступу до багатьох аспектів сучасних настільних та мобільних систем [15]. PyQt5 – це повний набір прив’язок Python для Qt версії 5.

При запуску головної програми, розробнику доступне лише одне меню (див. рис. 6.1). У меню присутня таблиця для заповнення. Умовні позначення на рисунку:

1) Визначає чи діє гравітація на персонажа;
2) Швидкість персонажа в пікселях на секунду;
3) Швидкість кулі в пікселях на секунду;
4) Час перезарядки зброї у мілі-секундах;
5) Визначення типу пострілів для персонажів, можна обирати лише один тип;
6) Додаткові властивості по типу місцевості, за їх наявності;
7) Кнопки для завантаження та зберігання моделі чи даних таблиці, також запуск навчання, при поданих параметрах у таблиці.

Кнопка «Show model work» дозволяє перевірити, як навчена нейронна мережа для таблиці з обраними параметрами.

Кнопка «Learn model» при коректних заданих параметрах, запускає алгоритм глибинного Q-навчання, описаний у розділі 6. Під час навчання розробник отримує інформацію про прогрес етапу навчання у мітці «Status».

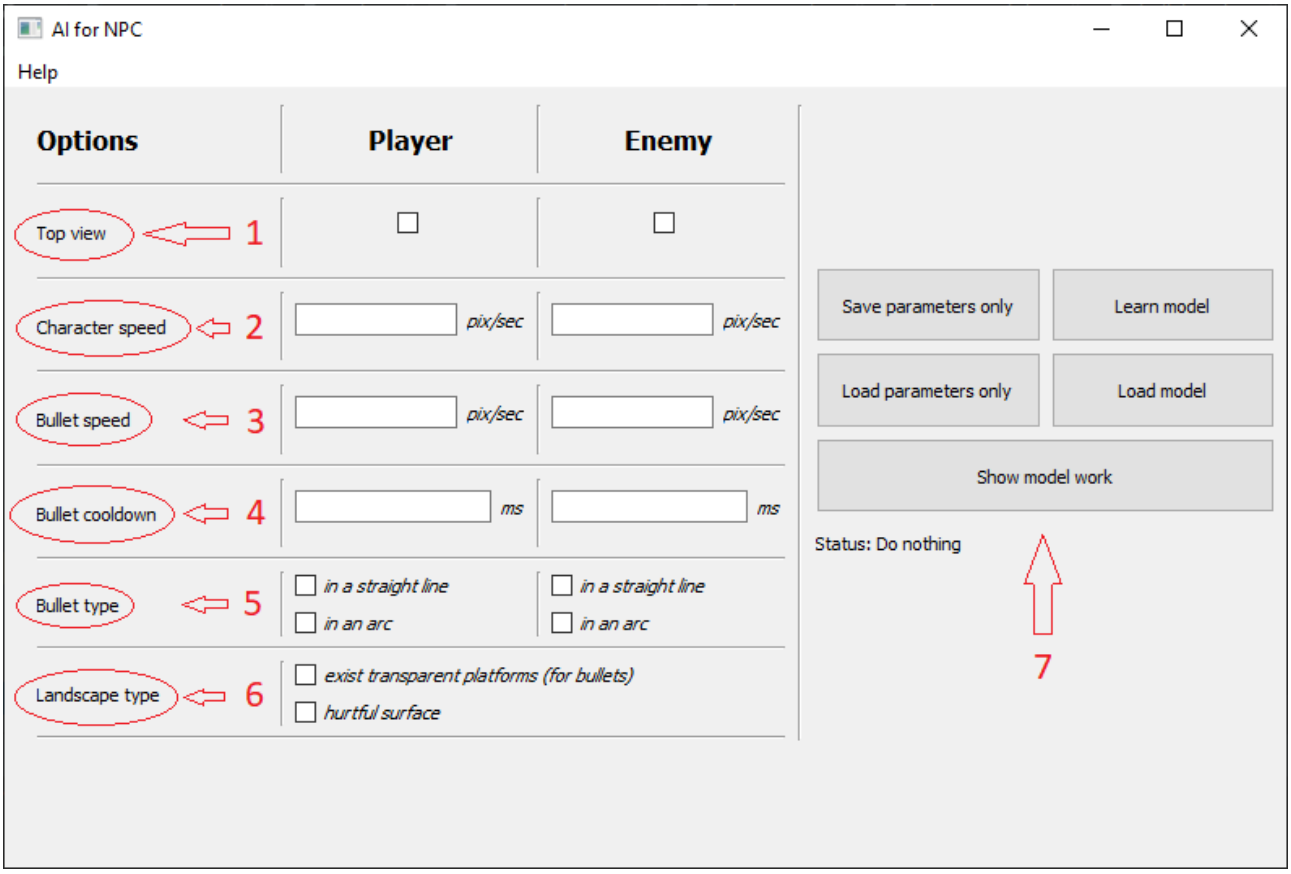


Рисунок 6.1 – Головне меню програми

Нейронні мережі зберігається в загальному форматі H5, який є загально-вживаним. Для того щоб використати у власному проекті нейронні мережі, розробнику потрібно:

– завантажити ваги нейронної мережі з файлу;
– використовувати алгоритм переводу зображення гри у піксельне зображення;
– надавати нейронним мережам кожен п’ятий фрейм (у випадку якщо у розробника швидкість оновлення середовища 60fps, інакше по співвідношенню);
– для визначення наступного кроку, надавати декілька послідовних отриманих фреймів (ця кількість вказується розробнику після завершення навчання);
– з результату нейронної мережі визначає індекс у якого найбільше Q-значення і виконує дію відповідно до індексу.

Таким чином розробник може навчати і використовувати стільки NPC, скільки потрібно для проекту гри.

РОЗДІЛ 7. ТЕСТУВАННЯ СЕРЕДОВИЩА

Для тестування середовища, було створено власні мапи (див. рис. 7.1), а для пришвидшеного процесу навчання було використано систему тензорного блоку обробки. Тензорний блок обробки (з англ. «tensor processing unit») – це інтегральна схема представлена компанією Google специфічного застосування призначена для швидкісних розрахунків спеціально для машинного навчання нейронних мереж.

а) б)

Рисунок 7.1 – Мапи для тестування: а – складний рівень;
б – простий рівень мапи

Як було подано в розділі 4, було використано два типи алгоритмів: DQRN та розширена DQN. Після навчання було отримано по 2 агенти від кожної нейронної мережі. Графіки навчання агентів DQRN та розширеної DQN дивитися детальніше у додатку А рис. А.1 та рис. А.2 відповідно.

Для перевірки ефективності було обрано порівнювати агенти, які керуються нейронними мережами та які керуються випадковим чином, між собою. Агенти грали між собою на тестових мапах (результат подано у табл. 7.1).

	Випадковий алгоритм			DQRN			Розширена DQN		
Результати гри	W	D	L	W	D	L	W	D	L
Випадковий алгоритм	X			16%	4%	80%	17%	20%	59%
DQRN	80%	4%	16%	X			54%	28%	18%
Розширена DQN	59%	20%	17%	18%	28%	54%	X		

Таблиця 7.1 – Результат алгоритмів в рядку проти алгоритмів у стовпчику за 100 ігор. Умовні позначення: W – перемога; D – нічия; L – програш; X – гра алгоритму проти самого себе.

Як результат було отримано, що алгоритм DQRN має кращу ефективність ніж розширена DQN. По графікам навчання з додатку А видно, що чим довше навчається агент у системі, тим краще він прогнозує свій наступний хід.

Подальше тестування полягає, щоб агенти нейронної мережі могли грати з людьми.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено додаток, який надає можливість створювати нейронні мережі по заданим параметрам, які можна застосовувати в 2D іграх, а саме 2D шутерах з виглядом збоку та зверху.

Під час виконання кваліфікаційної роботи було проаналізовано проблему створення NPC для існуючих 2D ігор шутерів; було узагальнено категорію 2D ігор шутерів з виглядом зверху та збоку в єдиний шаблон, завдяки чому можна створювати алгоритми управління NPC для цього шаблону, які будуть коректні для ігор з ідентичними характеристиками шаблону.

Також було показано, що такий тип ігор є POMDP тому для них можна застосовувати алгоритм глибинного Q рекурентного навчання або змінений алгоритм глибинного Q навчання. Управління персонажів у грі було розділено на 2 основні частини, для кожної з яких визначається різна структура DQRN чи DQN та система нагород.

Середовище для навчання нейронних мереж було створено з застосуванням мови програмування Python. Для створення самої DQRN та розширеної DQN було використано бібліотеку TensorFlow та Keras. В основі навчання береться принцип генеративних змагальних мереж, що два різних агенти нейронних мереж намагаються обіграти один одного. Як результат було отримано, що обидва алгоритми мають кращу ефективність у порівнянні з випадковим алгоритмом пересування агенту. Також алгоритм DQRN має кращу ефективність ніж розширений DQN. Таким чином алгоритм DQRN цілком можна використовувати в іграх.

Для взаємодії з нейронними мережами було створено додаток з використанням PyQt5, який дозволяє, швидко вводити потрібні параметри, завантажувати, зберігати та навчати нейронні мережі для власних ігор.

По поданим результатам роботи, було отримано універсальну систему для навчання нейронних мереж для використання їх у 2D шутер іграх з виглядом

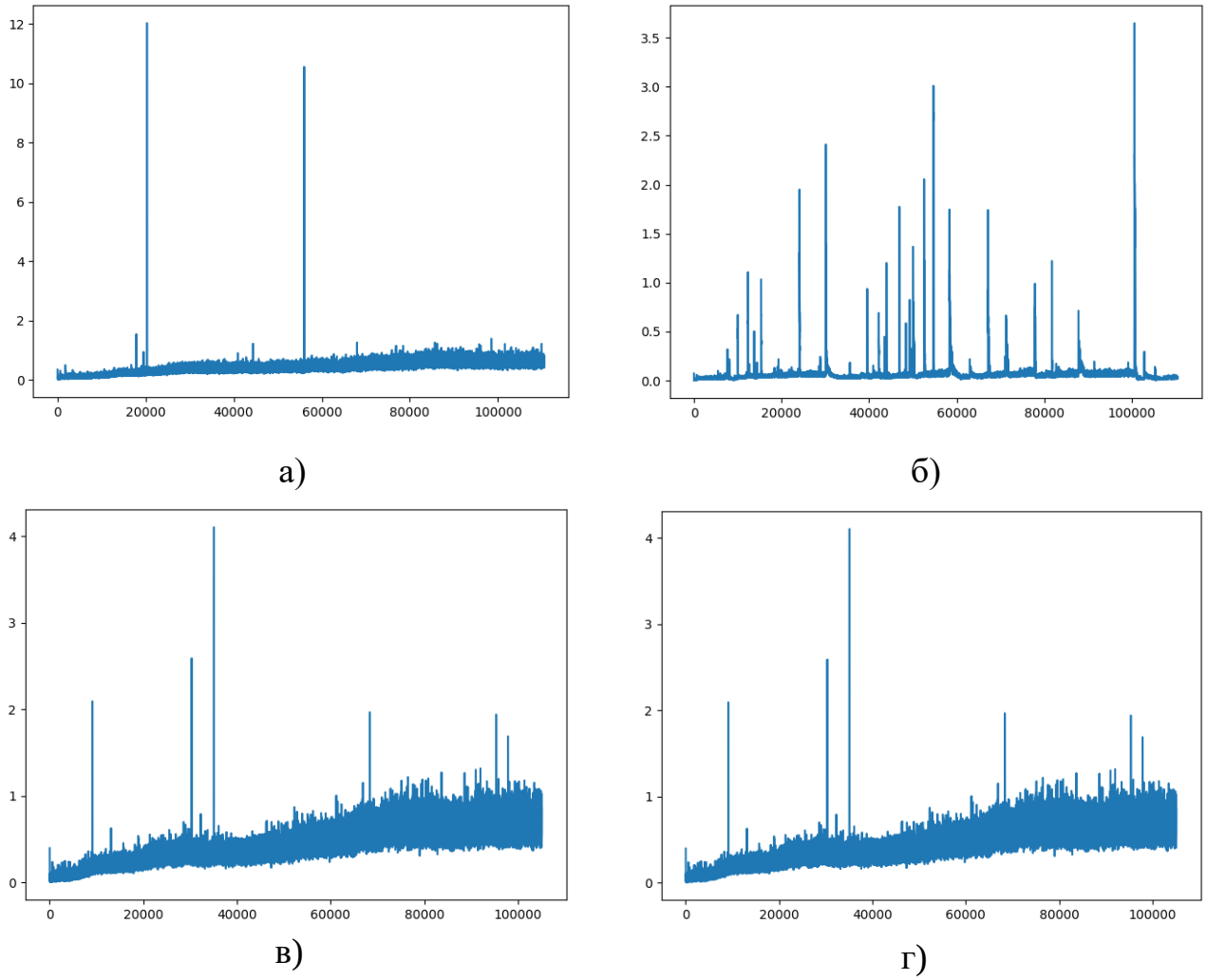
зверху та збоку, було створено прототип додатку для розробників, який буде полегшувати створення подібних ігор, і їм потрібно буде витрачати менше часу на реалізацію створення складних алгоритмів для керування NPC.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenCV [Електронний ресурс], – Режим доступу до ресурсу: https://pypi.org/project/opencv-python/.
2. Tensorflow [Електронний ресурс], – Режим доступу до ресурсу: https://www.tensorflow.org/.
3. Keras [Електронний ресурс], – Режим доступу до ресурсу: https://keras.io/;
4. PyQt5 [Електронний ресурс], – Режим доступу до ресурсу: https://pypi.org/project/PyQt5/.
5. The Binding of Isaac [Електронний ресурс], – Режим доступу до ресурсу: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/
6. Noita [Електронний ресурс], – Режим доступу до ресурсу: https://noitagame.com/.
7. Van Otterlo M. Reinforcement learning and markov decision processes / Van Otterlo M., Wiering M. // Reinforcement learning. – Springer, Berlin, Heidelberg, 2012. – P. 3-42.
8. Watkins C. J. C. H. Q-learning / Watkins C. J. C. H., Dayan P. // Machine learning. – 1992. – Vol. 8. – №. 3-4. – P. 279-292.
9. Watkins C. J. C. H. Learning from Delayed Rewards. Cambridge, UK, King's College : дис. – Dissertation, 1989.
10. Mnih V. et al. Playing atari with deep reinforcement learning // arXiv preprint arXiv:1312.5602. – 2013.
11. Chong E. K. P. Partially observable Markov decision process approximations for adaptive sensing / Chong E. K. P., Kreucher C. M., Hero A. O. // Discrete Event Dynamic Systems. – 2009. – Vol. 19. – №. 3. – P. 377-422.
12. Hausknecht M. Deep recurrent q-learning for partially observable mdps / Hausknecht M., Stone P. // arXiv preprint arXiv:1507.06527. – 2015.
13. Я. Гудфелоу, Глубокое обучение / Я. Гудфелоу, И. Бенджио, А. Курвилль – Litres, 2018.

14. Qt [Електронний ресурс], – Режим доступу до ресурсу: https://www.qt.io/.

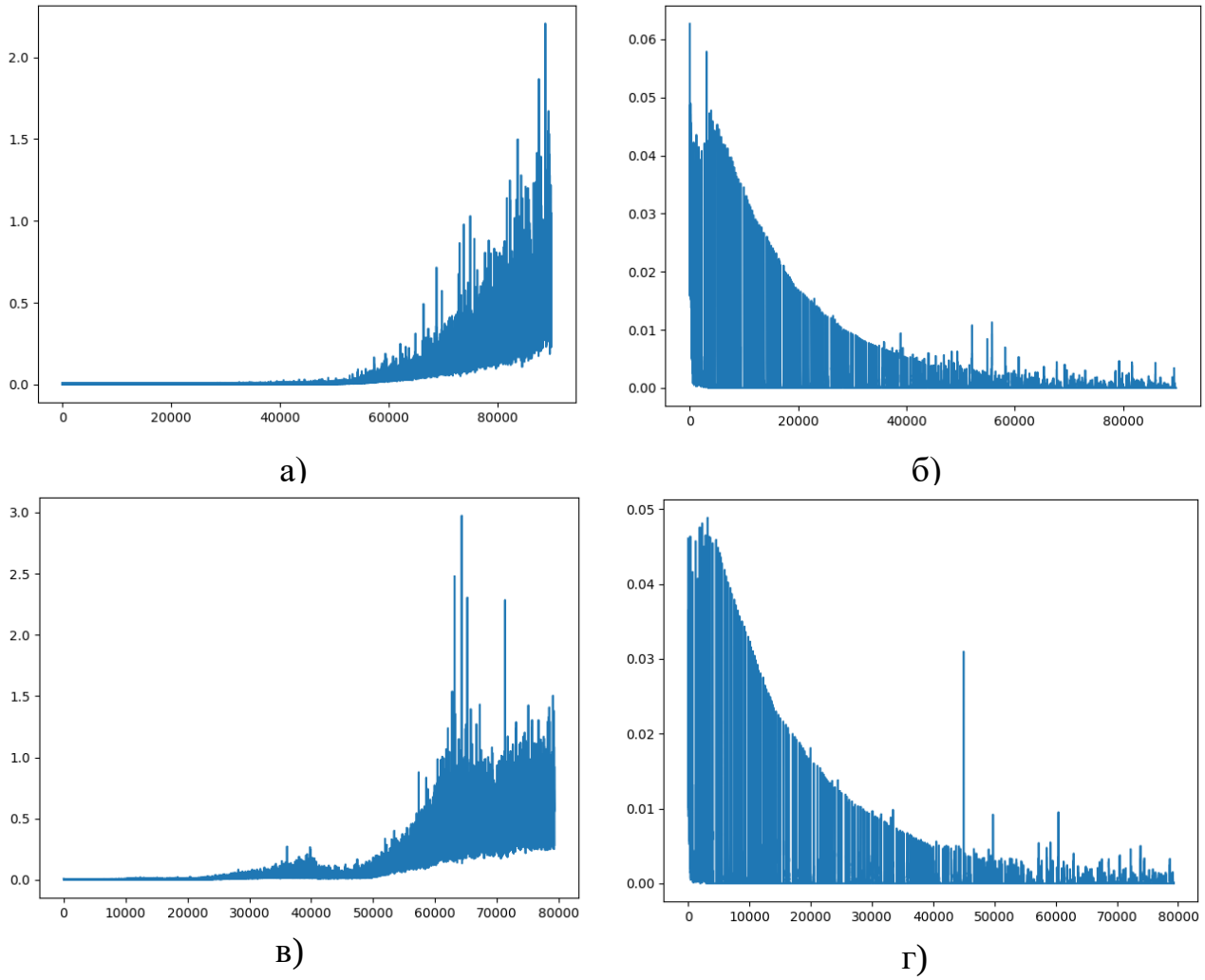
ДОДАТОК А



а) б)

в) г)

Рисунок А.1 – Графіки залежності похибки відносно епізодів для DQN: а – для під-агенту пересування суперника NPC; б – для під-агенту пострілів суперника NPC; а – для під-агенту пересування NPC гравця; б – для під-агенту пострілів NPC гравця



а) б)

в) г)

Рисунок А.2 – Графіки залежності похибки відносно епізодів для DQRN: а – для під-агенту пересування суперника NPC; б – для під-агенту пострілів суперника NPC; а – для під-агенту пересування NPC гравця; б – для під-агенту пострілів NPC гравця