Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка нейронної мережі для створення ігрових ситуацій»

Виконав: студент 2-го курсу, групи ЗІП-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
_____ Доценко Д. В.
(прізвище та ініціали)

Керівник: д.т.н., проф., зав. каф. ПЗ
_____ Романюк О.Н.
(прізвище та ініціали)
«08» грудня 2023 р.

Опонент:
_____ Крупельницький Л.В.
(прізвище та ініціали)
«08» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«08» грудня 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Рівень вищої освіти II-й (магістерський)

Галузь знань 12 – Інформаційні технології

Спеціальність 121 – Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Доценку Дмитру Володимировичу

1. Тема роботи – «Розробка нейронної мережі для створення ігрових ситуацій».

Керівник роботи: Романюк Олександр Никифорович, д.т.н., професор, завідувач кафедри ПЗ, затверджені наказом вищого навчального закладу «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи – «5» грудня 2023 р.

3. Вихідні дані до роботи: кольоровий режим –True Color; максимальний розмір дискретного координатного простору – 4096*4096; тривимірні матриці про досягнення гравцем рівнів; інформація про майстерність користувача при проходженні рівнів; час затрачений на проходження кожної ігрової ситуації; вихідні дані – ігрова ситуація, що була згенерована з використанням нейронної мережі.

4. Зміст текстової частини: вступ; обґрунтування розробки; розробка методу адаптивної зміни складності гри за рахунок використання нейронної мережі; розробка комп'ютерної гри для демонстрації результатів нейронної мережі; тестування системи; висновки.

5. Перелік графічного матеріалу: галузі застосування нейронних мереж для створення ігрових ситуацій; методи адаптивної зміни складності гри за рахунок використання нейронних мереж; методи аналізу складності ігрової ситуації; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-5	Романюк О. Н., д.т.н., завідувач кафедри ПЗ	19.09.23	05.12.23
6	Причепа І. В., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання «19» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Обгрунтування розробки	20.09.2023 – 25.09.2023	вик.
2	Розробка методу адаптивної зміни складності гри за рахунок використання нейронної мережі	26.09.2023 – 5.10.2023	вик.
3	Розробка комп'ютерної гри для демонстрації результатів нейронної мережі	6.10.2023 – 15.10.2023	вик
4	Тестування роботи системи	16.10.2023 – 25.10.2023	вик
5	Оформлення матеріалів до захисту МКР	26.10.2023 – 10.11.2023	вик.
6	Економічна частина	10.11.2023 – 1.12.2023	вик.

Студент _____ Доценко Д. В.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Романюк О. Н.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Доценко Д. В. Розробка нейронної мережі для генерації ігрових ситуацій. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 100 с.

На укр. мові. Бібліогр.: 50 назв; рис.: 32; табл. 10.

Проведено аналіз предметної галузі, створення динамічної складності в іграх, розглянуто методи демонстрації результатів виконання нейронної мережі, розглянуто методи використання нейронних мереж в іграх, обґрунтовано вибір фреймворку для розробки нейронної мережі для створення ігрових ситуацій.

Уперше запропоновано метод адаптивної зміни складності гри, особливість якого полягає у використанні нейронної мережі для динамічного аналізу майстерності гравця, що дало можливість інтелектуалізувати гру.

Уперше запропоновано метод, особливість якого полягає у використання нейронної мережі для аналізу складності ігрової ситуації, що дозволило покращити точність аналізу майстерності гравця.

Розроблено комп’ютерну гру в ігровому рушієві Unity для демонстрації створених ігрових ситуацій. Розроблено HTTP сервер для обміну даних між нейронною мережею та грою для генерації ігрових ситуацій.

Проведено тестування системи, що підтвердило відповідну до очікуваних результатів роботу системи.

Отриманні в магістерській роботі наукові та практичні результати можна використати для розробки систем адаптивної складності в іграх, на основі аналізу майстерності гравця.

ABSTRACT

Dotsenko D. V. Development of a neural network for generating game situations. Master's thesis in specialty 121 – software engineering, educational program – software engineering. Vinnytsia: VNTU, 2023. 100 p.

In Ukrainian language. Bibliography: 50 titles; figures: 32; tables: 10.

An analysis of the subject area was conducted, focusing on the creation of dynamic difficulty in games. Methods for demonstrating the results of the neural network's performance were considered, approaches to using neural networks in games were examined, and the choice of a framework for developing a neural network for creating game situations was justified.

For the first time, a method of adaptive game difficulty adjustment is proposed, characterized by the use of a neural network for dynamically analyzing the player's skill, enabling the intellectualization of the game.

A new method is proposed, utilizing a neural network to analyze the complexity of game situations, improving the accuracy of player skill analysis.

A computer game was developed using the Unity game engine to demonstrate created game situations. An HTTP server was developed for data exchange between the neural network and the game to generate game situations.

System testing was conducted, confirming the system's performance in line with expected results.

The scientific and practical results obtained in the master's thesis can be used for the development of systems with adaptive difficulty in games based on the analysis of player skill.

ЗМІСТ

ВСТУП

Розробка ігор є складним завданням, яке вимагає координації різних аспектів, таких як геймдизайн, графіка, фізика, штучний інтелект (ШІ), звуковий дизайн і багато інших. Використання нейронних мереж може полегшити розробку ігор в таких аспектах.

Використання нейронних мереж (НМ) для моделювання поведінки в ігрових персонажів може значно полегшити створення реалістичної ШІ. Нейронні мережі дозволяють створити агентів, які вчаться та адаптуються до змін у грі, забезпечуючи більшу гнучкість у проявленні різних видів поведінки.

НМ можуть бути використані для створення реалістичних анімацій та спеціальних ефектів. Наприклад, глибокі нейронні мережі можуть генерувати анімацію персонажів або текстури для ігрових об'єктів, що заощаджує час і ресурси на створенні анімацій вручну.

Нейронні мережі можуть бути використані для генерації ігрових рівнів, персонажів, предметів, музики та інших аспектів гри. Це полегшує завдання створення різноманітного контенту для гри, особливо в іграх з великим відкритим світом.

НМ можуть аналізувати взаємодію гравців з грою та надавати рекомендації розробникам щодо оптимізації геймплею, балансу гри та виявлення помилок в дизайні.

Використання нейронних мереж у графіці та ШІ допомагає створити більш реалістичну та іммерсивну гру, що робить ігровий досвід більш захоплюючим для гравців.

Аналіз гравців та реагування. Нейронні мережі можуть аналізувати взаємодію гравців з грою, щоб вирішити, як покращити гру в майбутньому, враховуючи відгуки гравців.

Таким чином, нейронні мережі забезпечують підвищення інтелектуального рівня гри за рахунок аналізу ігрових ситуацій. Тому актуальною є задача розробки нейронної мережі для генерації ігрових ситуацій.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета магістерської кваліфікаційної роботи полягає в розширенні функціональних можливостей гри за рахунок збільшення ігрових ситуацій та покращення динамічної складності. **Головні задачі дослідження є:**

- провести аналіз існуючих методів створення ігрових ситуацій, що використовуються при розробці сучасних ігор та методів розробки нейронної мережі;
- запропонувати:
  - метод адаптивної зміни складності гри, особливість якого полягає у використанні нейронної мережі для динамічного аналізу майстерності гравця, що дало можливість інтелектуалізувати гру;
  - метод, який полягає у аналізі ігрових ситуацій за рахунок використання нової моделі НМ для аналізу складності ігрової ситуації, що дозволило покращити точність аналізу майстерності гравця;
  - метод аналізу зображення за рахунок використання функції активації параметричного зрізаного лінійного вузла та додання додаткових шарів згортки, що дало можливість збільшити точність аналізу.
- розробити та провести тренування нейронної мережі для генерації ігрових ситуацій на основі уявних користувачів;
- розробити гру для показу результату виконання нейронної мережі;
- провести експериментальні дослідження розроблених засобів.

Об'єкт дослідження – процес розробки нейронної мережі для генерації ігрових ситуацій.

Предмет дослідження є методи та засоби розробки нейронної мережі для генерації ігрових ситуацій.

Методи дослідження. У процесі розробки магістерської кваліфікаційної роботи було використано: теорія машинного навчання, теорія нейронних мереж, теорія комп'ютерної графіки, теорія чисел для розробки методів і моделей, теорія збереження пам'яті виконання нейронної мережі для розробки нейронних мереж, інтерпретація результатів, теорія підготовки даних, архітектура нейронної мережі для тренування нейронної мережі, засоби розробки ігор для демонстрації результатів нейронної мережі, метод класифікації зображень для покращення точності аналізу ігрової ситуації.

Наукова новизна отриманих результатів:

- уперше розроблено метод адаптивної зміни складності гри, особливість якого полягає у використанні нейронної мережі для динамічного аналізу майстерності гравця, що дало можливість збільшити інтелектуальний рівень гри і розширити її функціональні можливості;
- уперше розроблено метод, який полягає у аналізі ігрових ситуацій за рахунок використання нової моделі НМ для аналізу складності ігрової ситуації, що дозволило покращити точність аналізу майстерності гравця;
- подальшого розвитку отримав метод аналізу зображення за рахунок використання функції активації параметричного зрізаного лінійного вузла та додання додаткових шарів згортки, що дало можливість збільшити точність аналізу.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих у магістерській кваліфікаційній роботі результатів розроблено алгоритми та програми для нейронної мережі, яка генерує ігрові ситуацій, що може бути використана для пришвидшення розробки гри, збільшення різноманітності ігрових ситуацій та динамічної складності.

Особистий внесок здобувача. Усі наукові результати викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У роботах, написаних у співавторстві, автору належить: алгоритм оцінювання адаптивної складності гри [1]; структура нейронної мережі [2].

Апробація роботи матеріали магістерської кваліфікаційної роботи доповідались на конференціях: VI Всеукраїнська науково-практична інтернет-конференція молодих вчених та студентів «Сучасні інформаційні системи та технології», Міжнародна науково-практична конференція «Інформаційні технології та автоматизація – 2023».

Публікації. За тематикою дослідження опубліковано 2 наукові праці у матеріалах конференції.

1 ОБҐРУНТУВАННЯ РОЗРОБКИ

1.1 Аналіз предметної галузі

Гра – діяльність людини з моделювання іншого виду діяльності з розважальною чи навчальною метою. Гра відрізняється від роботи тим, що не ставить перед собою безпосередньо корисної мети, хоча сама гра може мати свою власну корисність, а також тісно межує з мистецтвом, хоч зазвичай не створює художніх цінностей [3].

У контексті ігор, "ситуація" [4] вказує на обставини, обстановку, або стан, в якому гравці опиняються під час гри. Ситуація в іграх може включати в себе різні фактори, такі як завдання, правила, оточуючий світ, взаємодія з іншими гравцями або персонажами, та може бути специфічною для кожної гри. Ситуація визначає, які дії гравців можуть виконувати та як вони взаємодіють з грою, і впливає на хід гри та її результат. В іграх ситуації можуть бути створені розробниками гри або виникати в результаті взаємодії гравців, що додає варіативність та цікавість до ігрового процесу.

Ігрові ситуації можна класифікувати за різними критеріями, такими як цілі гри, тип взаємодії між гравцями та ігровим середовищем. Ось деякі типи ігрових ситуацій:

Конкурентні ситуації: у таких ситуаціях гравці змагаються один проти одного з метою перемогти чи виявити свою перевагу над іншими. Прикладами є спортивні ігри, шахи, багато онлайн-ігор.

Співпраця (кооперативні) ситуації: гравці працюють разом, співпрацюючи для досягнення спільних цілей. Це може бути спільна робота в команді в онлайн-іграх або вирішення головоломок в парному або груповому режимі.

Ситуації дослідження: гравці вивчають нові галузі чи світи, вивчаючи їх і взаємодіючи з ними. Це типово для пригодницьких ігор, віртуальної реальності та ігор "відкритого світу".

Рольові ситуації: гравці виконують певні ролі та взаємодіють відповідно до цих ролей. Це характерно для рольових ігор, де гравці створюють персонажів і розвивають їхню історію.

Головоломкові ситуації: гравцям потрібно розв'язати складні головоломки, логічні завдання та задачі для перемоги. Це може бути частиною гри або визначальним елементом.

Стратегічні ситуації: гравці мають розробляти стратегію та планувати свої дії, щоб перемогти. Це типово для стратегічних ігор, таких як шахи, стратегічні військові ігри тощо.

Ситуації зіткнення [5]: гравці вступають в битви в режимі реального часу, де динамічна взаємодія та швидкість важливі. Це може включати в себе шутери, виживальні ігри, ігри масового спільного гри та інші.

Ці типи ігрових ситуацій можуть співіснувати або поєднуватися в одній грі для створення багатошарового ігрового досвіду.

Створення ігрових ситуацій є складним та важким процесом, який вимагає ретельного планування, розробки та тестування.

Нейронні мережі [6] представляють собою складну систему алгоритмів, які стали важливою частиною сучасного напрямку в глибокому навчанні. Вони намагаються відтворити функції мозку, зокрема його здатність вирішувати складні завдання розпізнавання та класифікації.

Історія нейронних мереж [7] має свої корені в роботах Уоррена МакКалока та Уолтера Пітца в 1943 році, які створили модель нейрону, базуючись на біологічних принципах. Протягом багатьох десятиліть цей напрямок розвивався повільно через обмежену обчислювальну потужність та відсутність ефективних алгоритмів навчання.

Однак з появою сучасних обчислювальних засобів та вдосконалення алгоритмів навчання, таких як зворотне поширення, нейронні мережі отримали новий імпульс розвитку. В 2012 році архітектура глибокого навчання, відома як "AlexNet" [8], встановила новий стандарт в задачах розпізнавання об'єктів на

зображеннях, що відзначило початок буму застосування нейронних мереж у різних галузях.

Нейронна мережа складається з вхідного шару, прихованих шарів і вихідного шару. Нейрони в кожному шарі пов'язані з нейронами попереднього та наступного шарів, утворюючи взаємозв'язану мережу.

Існує кілька типів нейронних мереж, таких як:

- Feedforward Neural Networks (FNN): інформація рухається від вхідного до вихідного шару без зворотного зв'язку;
- Recurrent Neural Networks (RNN): мають циклічні зв'язки, що дозволяють зберігати інформацію про попередні вхідні дані;
- Convolutional Neural Networks (CNN): спеціалізовані для обробки зображень, використовують згорткові шари для виділення важливих ознак.

Ці архітектури [9] можуть бути адаптовані до різних завдань, забезпечуючи гнучкість та ефективність вирішення різноманітних проблем.

Нейронна мережа складається з трьох основних типів шарів: вхідний, прихований та вихідний. Вхідний шар отримує вхідні дані, приховані шари обробляють ці дані, а вихідний шар генерує кінцевий результат.

Кожен нейрон в мережі пов'язаний з нейронами попереднього та наступного шарів через зв'язки. Кожен зв'язок має вагу, яка регулює вплив сигналу на наступний нейрон.

Активаційні функції визначають вихід нейрона на основі його вхідних сигналів та ваг. Поширено використовувані функції включають сигмоїду, гіперболічний тангенс та ReLU.

Нейронні мережі стали ефективним інструментом для завдань розпізнавання об'єктів та зображень. Зокрема, архітектури типу CNN [10] виявляють вражаючі результати у сферах комп'ютерного зору, автоматичного водіння та медицинського зображення.

Застосування нейронних мереж у сфері обробки природних мов [11] дозволяє досягти високої точності у завданнях машинного перекладу, розпізнавання мовлення, аналізу тексту та генерації природних мовленнєвих моделей.

Нейронні мережі застосовуються в галузі відеоігор для створення реалістичних графічних образів, покращення штучного інтелекту в іграх та персоналізації геймплейу.

Використання нейронних мереж у гейм-розробці може значно пришвидшити створення ігрових ситуацій завдяки їхнім здатностям до автоматизації та оптимізації різних аспектів геймплею, дозволяє розробникам створювати більш різноманітні, цікаві та інноваційні ігри, зменшуючи завдання ручного створення і розширюючи можливості ігрового досвіду для гравців.. Нейронні мережі можуть бути натреновані на створення процедурно генерованих рівнів гри. Вони можуть аналізувати певні параметри, такі як складність, розмір, типи об'єктів, і створювати рівні відповідно до цих параметрів. Це дозволяє створювати безкінечну кількість різноманітних рівнів для гравців. Використання нейронних мереж для генерації ігрових ситуацій.

Отже, для створення магістерської роботи доцільно розробити нейрону мережу для генерації ігрових ситуацій.

1.2 Методи відображення результатів використання нейронної мережі

При використанні нейронних мереж для різних завдань, результати роботи нейронної мережі можуть бути відображені та аналізовані різними способами в залежності від конкретної задачі. Ось деякі загальні методи відображення результатів:

Графік функції втрат [12]: під час навчання нейронної мережі рекомендується візуалізувати графік функції втрат для визначення якості моделі на різних епохах навчання.

Точність: Вимірює, як часто модель правильно класифікує дані у задачах класифікації.

Інтерактивні інтерфейси та демонстрації: Створення інтерактивних додатків для аналізу та демонстрації результатів моделі, які дозволяють користувачам взаємодіяти з передбаченнями та іншими аспектами моделі.

Для магістерської роботи доцільно розробити інтерактивний інтерфейс, тобто гру, для показу результату розробки.

Розглянемо різні типи ігор, де гравці потрапляють у ситуації зіткнення.

Шутери з виживанням. У таких іграх гравці зазвичай опиняються в небезпечних умовах та повинні боротися з хвилями монстрів, використовуючи різні види зброї.

Action RPG. У цьому типі ігор гравці взаємодіють з монстрами у фентезійних світах. Вони можуть вдосконалювати свого персонажа, отримувати нове зброю та магічні навички.

Tower Defense [13]. Гравці в цих іграх розташовують вежі та перешкоди для захисту свого території від наступаючих монстрів.

Ігри жахів. Ці ігри побудовані на створенні жахливої атмосфери та напруги. Гравцям доводиться зіткнутися з жахливими монстрами, використовуючи обмежені ресурси.

Платформери. У платформерах гравцям потрібно подолати перешкоди та ворогів.

Roguelike [14], також мандрівні ігри. Roguelike – це жанр ігор, де гравці відправляються на відкриття підземелля, в якому вони зустрічають різні монстри та загадкові істоти. Ці ігри часто відомі за своєю випадковою генерацією рівнів та перманентними смертями, які роблять кожну гру унікальною.

Отже слід розробити гру у жанрі «roguelike», що дозволить збільшити кількість ситуацій, де використання нейронних мереж є доцільним та показати ефективність використання машинного навчання, розробленої у магістерській роботі.

1.3 Методи використання нейронної мережі для генерації ігрових ситуацій

Використання нейронних мереж для генерації ігрових ситуацій може бути важливим етапом у розробці ігор. Ось декілька методів, які можна використовувати для досягнення цієї мети:

Генерація об'єктів та об'єктних параметрів: нейронні мережі можуть створювати об'єкти в ігровому світі, такі як персонажі, предмети, архітектурні структури тощо. Наприклад, мережа може генерувати різноманітні персонажі з різними характеристиками, навичками та властивостями.

Процедурне створення рівнів [15]: нейронні мережі можуть генерувати ігрові рівні або сценарії. Це включає в себе створення ландшафтів, розміщення ворожих сил, визначення цілей гравця тощо.

Підганяння даних взаємодії зі світом: Нейронні мережі можуть аналізувати взаємодію гравців з ігровим світом та реагувати на неї. Наприклад, мережа може пристосовувати складність гри відповідно до рівня майстерності гравця.

Системи моралі та поведінки: Нейронні мережі можуть бути використані для моделювання моралі і поведінки в ігрових персонажів, забезпечуючи більш складні реакції на дії гравця.

Генерація діалогів та історії: Нейронні мережі можуть генерувати текстові діалоги, історії та квести в грі, що робить їх більш цікавими для гравців.

Генерація кінематографії та анімації: Мережі можуть бути використані для створення кінематографії та анімації в іграх, які рухаються згідно зі сюжетними потребами.

Аналіз та оптимізація геймплею [16]: Нейронні мережі можуть аналізувати взаємодію гравця з грою та надавати рекомендації розробникам щодо оптимізації геймплею та балансування гри.

При розробці магістерської роботи було прийнято рішення використання нейронної мережі для генерації ігрового контенту та аналізу гравців і взаємодії.

1.4 Вибір фреймворку для реалізації нейронної мережі

Порівняння різних фреймворків для створення нейронних мереж може варіюватися в залежності від конкретних потреб та завдань. Ось кілька популярних фреймворків та їх характеристики.

TensorFlow:

- Масштабований та гнучкий.
- Підтримує обране API для вищого рівня, таке як Keras.
- Велика спільнота користувачів та багато матеріалів для навчання.
- Відкрите джерело.

PyTorch:

- Динамічний граф обчислень, що полегшує налагодження та дозволяє більшу гнучкість.
- Підтримує популярні бібліотеки для обробки даних, такі як NumPy.
- Широка підтримка для комп'ютерного бачення та обробки природних мов.
- Відкрите джерело.

Caffe:

- Оптимізований для швидкості та обробки зображень.
- Використовується в багатьох дослідницьких проектах та програмах розпізнавання обличчя.
- Має обмежену підтримку для послідовних моделей.
- Відкрите джерело.

MXNet:

- Підтримує гнучкий імперативний та символічний підхід до обчислень.
- Швидкий та оптимізований для мультиплатформеного використання.
- Підтримує комп'ютерне бачення, оброблення природних мов та рекомендаційні системи.

CNTK (Microsoft Cognitive Toolkit):

- Розроблений для швидкості та високопродуктивних обчислень.

- Підтримує різні мови програмування, включаючи Python та C#.
- Використовується для глибокого навчання, розпізнавання мови, обробки зображень та інших завдань.

Для розробки нейронної мережі, слід обрати TensorFlow через його масштабованість, гнучкість та активну спільноту, яка надає робочі рішення для різних завдань глибокого навчання, від комп'ютерного бачення до обробки природних мов, і дозволяє легко впроваджувати нейронні мережі на великій кількості пристроїв.

1.5 Постановка задачі для розробки

На основі технічного завдання та проведеного аналізу предметної галузі, доцільною є розробка нейронної мережі для генерації ігрових ситуацій,. Необхідно створити та натренувати нейрону мережу, що буде використана для створення ігрових ситуацій та створення динамічної складності для гравця.

Потрібно провести аналіз існуючих методів створення ігрових ситуацій, що використовуються при розробці сучасних ігор, а саме провести аналіз предметної галузі, проаналізувати методи відображення результату виконання нейронних мереж, обрати метод використання нейронної мережі для генерації ігрових ситуацій, порівняти фреймворки для реалізації нейронної мережі.

Необхідно розробити нейрону мережу для генерації ігрових ситуації, що включає у себе аналіз архітектури нейронних мереж, розробка моделі нейронної мережі, аналіз методів навчання нейронної мережі, створення уявних гравців для тренування нейронної мережі, тренування нейронної мережі.

Слід створити гру у жанрі «Roguelike», для показу роботи нейронної мережі, а саме провести аналіз ігрових движків для розробки, розробити взаємодію гравця з оточенням, розробити об’єкти, що будуть використані нейронної мережею для генерації ситуації, інтегрувати нейрону мережу до гри.

Перевірити роботу гри, що показує виконання нейронної мережі, затвердити роботу нейронної мережі для генерації сценаріїв на відсутність помилок та відповідність поставленим задачам.

1.6 Висновки

У цьому розділі було наведено результати проведеного аналізу предметної галузі, методів відображення результатів виконання нейронної мережі, визначено мету використання нейронної мережі для генерації ігрових ситуацій, обрано фреймворк для розробки нейронної мережі, та визначено головні задачі.

У результаті було підтверджено доцільність розробки програмного продукту.

2 РОЗРОБКА МЕТОДУ АДАПТИВНОЇ ЗМІНИ СКЛАДНОСТІ ГРИ ЗА РАХУНОК ВИКОРИСТАННЯ НЕЙРОННОЇ МЕРЕЖІ

2.1 Постановка задачі для розробки нейронної мережі

Для аналізу майстерності гри необхідно надати нейронній мережі дані про проходження гравцем попередньо створених рівнів, а саме структуру рівня та параметри, що визначають майстерність гравця. На основі цих даних нейрона мережа зробить аналіз необхідності ускладнення чи спрощення згенерованих рівнів, практика надання попередніх даних перед створення передбачень нейронною мережею називається “Warmup” [17].

Для генерації наступних ігрових ситуації потрібно надати дані про попередньо створені та згенеровані нейронною мережею рівні, та дані про майстерність гравця на пройдених рівнях, рисунок 2.1.

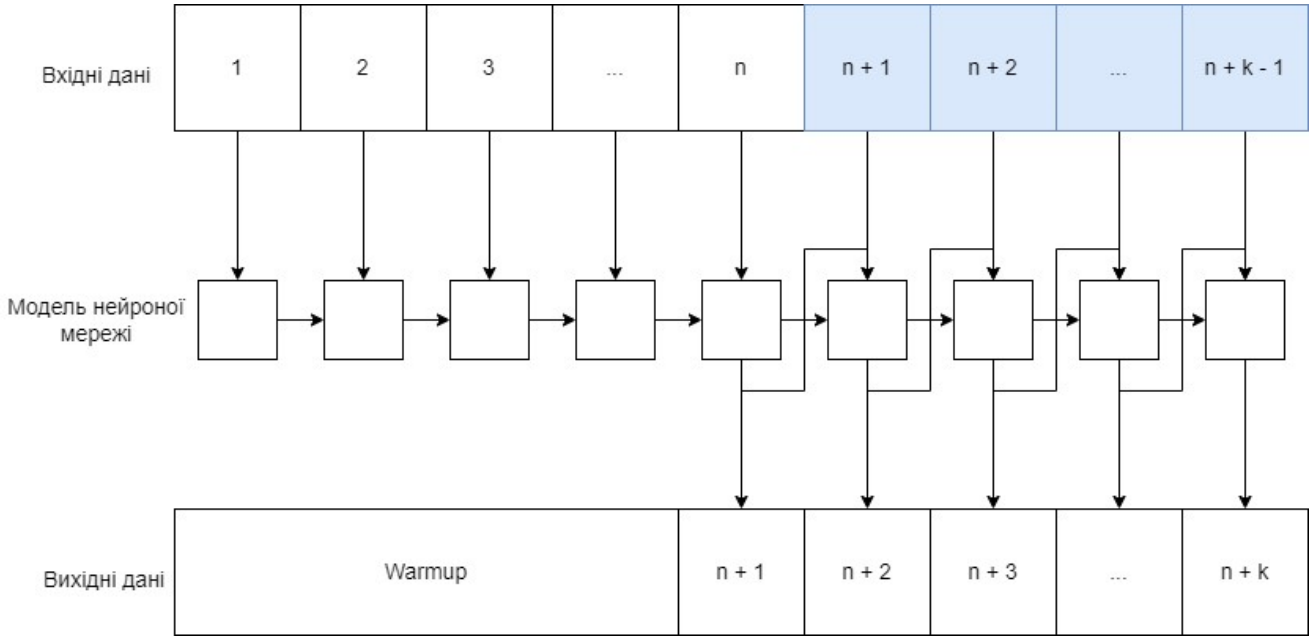


Рисунок 2.1 – Візуалізація методу роботи нейронної мережі

На рисунку зображено вхідні дані, що надаються нейронній мережі, а саме: на проміжку від 1 до n, дані про попередньо створені рівні та параметри майстерності гравця; на проміжку від n + 1 до n + k − 1, дані про майстерність

гравця на згенерованих нейронною мережею рівнях. Зміна стану моделі нейронної мережі на основі вхідних даних та передбачень. Вихідні дані, ігрові ситуації, що було згенеровано нейронною мережею.

Тому потрібно розробити нейрону мережу, що на вхід отримуватиме дані про майстерність гравця та попередньо створені рівні та буде їх зберігати у пам’яті, на основі чого буде виконуватись генерація ігрових ситуацій.

2.2 Покращення моделі нейронної мережі для аналізу складності ігрової ситуацій

Перед генерацією ігрової ситуації нейрона мережа має провести аналіз існуючих рівнів та передати результати аналізу, без урахування вмінь гравця, що дозволить збільшити точність подальшого аналізу вмінь гравця.

Для цього було покращено метод для класифікації зображень на основі CNN шарів [18].

Нейрона мережа може бути використана для аналізу існуючих рівнів та зробити передбачення у складності для середньостатистичного користувача. Для цього слід підготовити дані структури ігрового рівня у форматі, що буде зрозумілим нейронній мережі, наприклад використання трьох мірних матриць, у якої перші два виміри будуть відповідати за розташування, а третій за тип об’єкта.

Для аналізу рівня слід використати шар згортки (convolutional) нейронної мережі, що дозволить зберегти у собі дані про попередні ігрові ситуації, у випадку аналізу даних в матриці потрібно використати Conv2D, це двох мірний шар згортки з внутрішньою матрицею.

Для подальшої передачі даних, з метою генерації рівня, слід використати формат даних, що зрозумілий для нейронної, для цього слід виходи шару перенести у одновимірний масив з 3 мірної матриці, це можливо використовуючи шар виірвнювання.

Між шарами нейронної мережі необхідно обрати функції активації, це перетворення виходів попередніх шарів перед подачею на наступний. Найбільш відомий тип активації, ReLU – зрізаний лінійний вузол [19], рисунок 2.2.

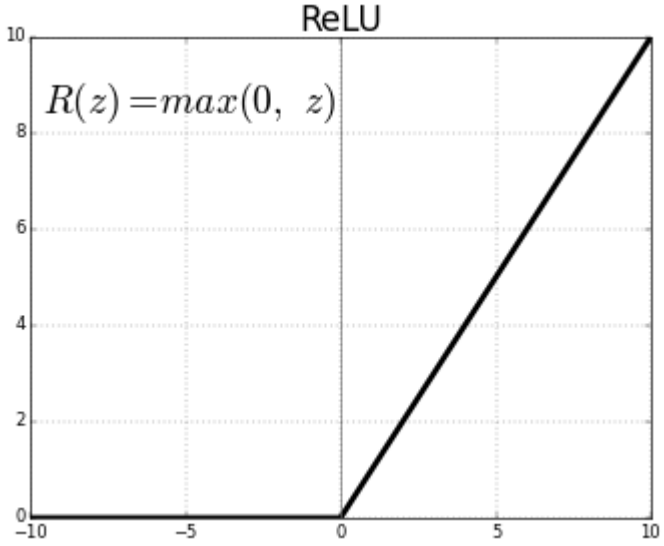


Рисунок 2.2 – Графік зрізаного лінійного вузла

Недоліком ReLU є відсутність тренування блоків нейронної мережі у випадку коли один з виходів знаходиться на від’ємному відрізку, що зменшує можливості аналізу нейронної мережі, тому слід використати PreLU [20], параметрична ReLU.

$$f(x) = \begin{cases} x, & x > 0 \\ ax, & x < 0 \end{cases}$$

Зазвичай значення, що береться у від’ємному проміжку множиться на 0.3, що дозволяє продовжити тренування нейронної мережі навіть для не активованих виходів.

Слід добавити декілька шарів згортки з метою спрощення аналізу даних, оскільки кожний шар зменшує розмір матриці у залежності від розміру рецептивного поля, що полегшить подальший аналіз нейронною мережею.

Між шарами згортки та виходу нейронної мережі слід розрівняти виходи з шарів згортки, що дозволить отримати виходи у форматі одномірного масиву.

Для виходу нейронної мережі слід використати щільний шар (dense) нейронної мережі з кількістю виходів рівному кількості параметрів для аналізу

ігрової ситуації, наприклад у випадку використання затраченого часу та втраченого здоров’я гравцем слід використати два виходи.

Нехай нам необхідно аналізувати ігрову ситуації, що можна зобразити матрицею з розмірами 9 на 9 на 3 та потрібно отримати результат з одним виходом, то модель нейронної мережі може використати три шари згортки, з розміром рецептивного поля у 3 на 3 та 16 виходів, шар для вирівнювання виходів шарів згортки та один щільний шар з одним виходом, рисунок 2.3.

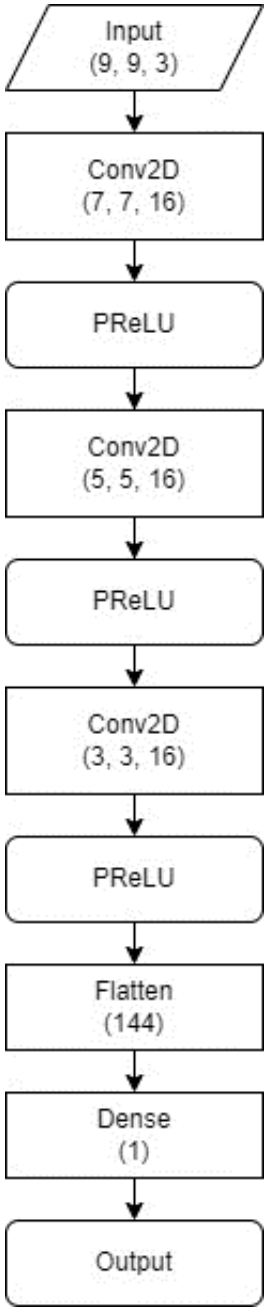


Рисунок 2.3 – Модель нейронної мережі

Отже, було розроблено модель нейронної мережі, що отримує історію пройдених гравцем рівня та на основі цього визначає, незалежно від умінь гравця, складність рівня.

2.3 Розробка моделі нейронної мережі для генерації ігрових ситуацій

На основі виходів моделі нейронної мережі, що було описано у попередньому розділі, та майстерності гравця, слід розробити модель нейронної мережі, що буде аналізувати вхідні дані та генерувати ігрові ситуації.

Для цього потрібно провести аналіз майстерності користувача, щоб визначити необхідність ускладнення або спрощення наступних рівнів, тому використаємо LSTM для проведення аналізу з функцією активації PreLU.

Результати цього аналізу потрібно об’єднати з виходом моделі нейронної мережі для екстраполяції складності ігрових ситуацій.

Потім потрібно почати генерувати ігрову ситуацію, для цього слід використати щільний шар (Dense), також відомий як повністю зв'язаний шар, і використовується для абстрактних представлень вхідних даних. У цьому шарі нейрони з'єднані з кожним нейроном попереднього шару. Розмір шару слід виставити на добуток розмірів вхідної матриці.

Функцію активації слід обрати з перетворенням виходу до проміжку з визначеним граничними значеннями. Існує декілька варіантів, що можуть забезпечити необхідну умову.

Сигмоїдна функція активації [21] є популярним вибором для нелінійної функції активації у нейронних мережах. Однією з причин популярності є те, що вона має значення виходу між 0 та 1, що відтворює значення ймовірностей. Тому вона використовується для перетворення дійсно значеного виходу лінійного шару у ймовірність. Це також робить її важливою частиною методів логістичної регресії, яку можна використовувати безпосередньо для бінарної класифікації, рисунок 2.4.

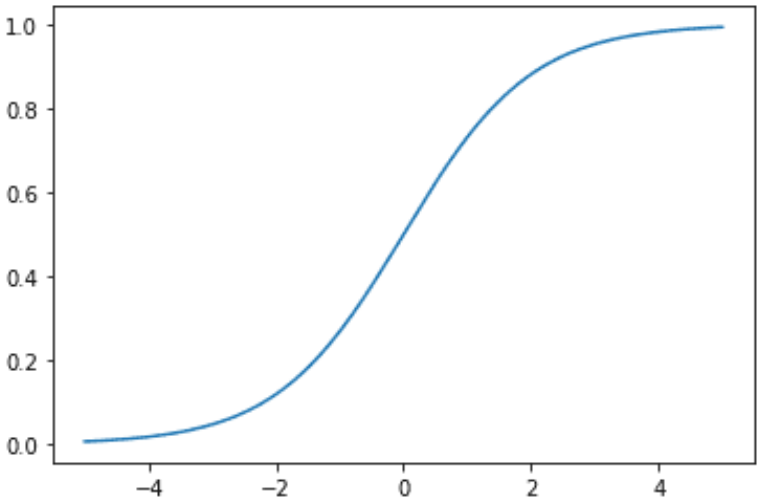


Рисунок 2.4 – Графік сигмоїдної функції активації

Ще одна функція активації для розгляду – це функція активації гіперболічного тангенсу [22], також відома як функція гіперболічного тангенсу. Вона має більший діапазон значень виходу порівняно з сигмоїдною функцією і більший максимальний градієнт, рисунок 2.5.

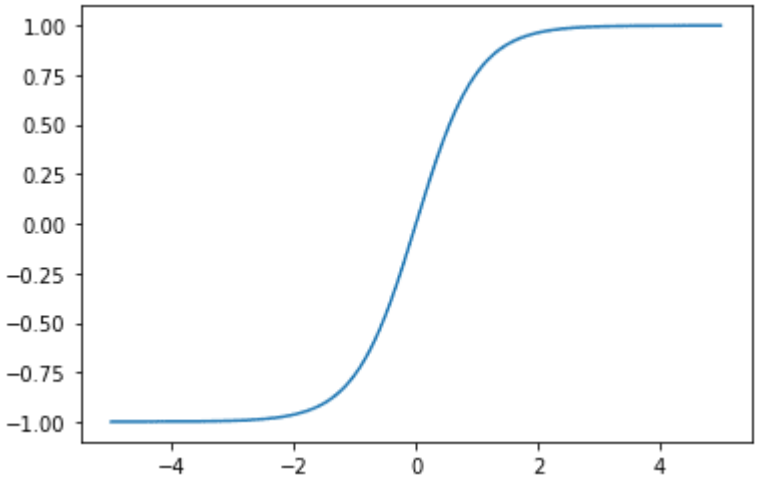


Рисунок 2.5 – Графік гіперболічного тангенсу

На основі цього слід використати сигмоїдну функцію активації, що дозволить показати пріоритет розташування об’єктів.

Вихідний шар нейронної мережі буде перетворювати вихід попереднього шару у формат 3 мірної матриці, що буде символізувати згенеровану нейронною мережею, ігрову ситуацію.

Розробимо модель нейронної мережі для генерації ігрових ситуацій, рисунок 2.6.

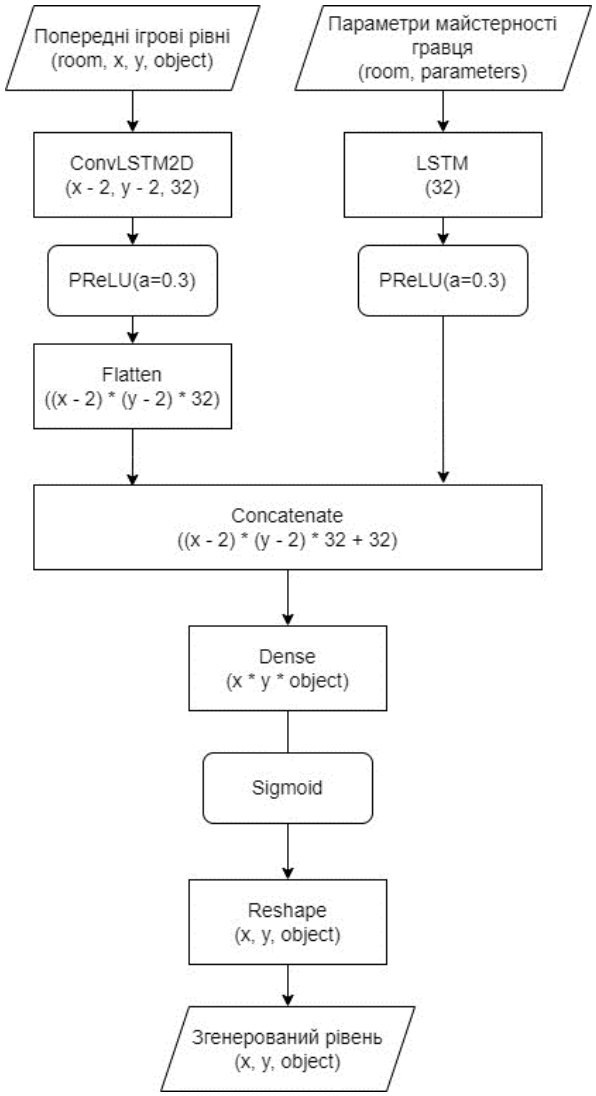


Рисунок 2.6 – Модель нейронної мережі для генерації ігрових ситуацій

Отже, було розроблено модель нейронної мережі, що аналізує історію гри гравця та на основі цього генерую ігрову ситуацію, що підлаштовується під гравця.

2.4 Розробка функції втрат нейронної мережі

Для проведення тренування нейронної мережі необхідно вказати нейронній мережі, наскільки результати відповідають до очікуваних значень, у нашому випадку потрібно провести аналіз згенерованої ігрової ситуації та перевірити наскільки це складний рівень для користувача.

Існує декілька варіантів для аналізу складності ігрової ситуації, перший включає в себе попереднє створення великої кількості ігрових ситуацій та наступне їх проходження, але це потребує великої затрати часу та людей.

У другому варіанті використовуємо штучних гравців, що будуть проходити рівні, які будуть генеруватись нейронною мережею, але виникає проблема у невідповідності роботи комп’ютера та людини при проходженні ігрової ситуації.

У третьому варіанті поєднуємо обидва варіанти, створюємо декілька попередньо згенерованих рівнів, проходимо їх та на основі цих даних робимо аналіз рівнів згенерованих нейронною мережею, цей метод дозволить уникнути проблему у затраті часу та різниці між людьми та комп’ютерами.

Також функція втрат має штрафувати нейрону мережу у випадку помилкових генерацій, наприклад два об’єкта згенерованих в одному місці.

Для визначення втрат в обрахунках нам необхідно визначити по яких параметрах буде визначатись майстерність користувача.

Використання часу, що був затрачений гравцем на проходження згенерованої кімнати, що дозволить легку імплементацію для функції втрат без необхідності в детальному аналізі, але викликає проблему в можливості створення лабіринтів нейронною мережею.

Використання нанесеного та отриманого пошкодження гравцем, аналізуючи ці метрики, надає можливість нейронній мережі більш збалансовано розміщувати ворогів та їх кількість, але викликає проблему для передбачення штучними гравцями втраченого здоров’я та зменшує зв’язок з об’єктами, що не мають пошкоджень, наприклад стін.

Тому слід використати перший метод з збільшенням покарання за кожний однаковий розміщений об’єкт.

Лістинг функції втрат та штучних гравців, розміщено у додатку В.2.

2.5 Підготування до тренування нейронної мережі

Тренування нейронної мережі - це процес навчання мережі за допомогою вхідних даних з метою визначення оптимальних ваг та параметрів для вирішення конкретної задачі. Процес тренування нейронних мереж включає в себе:

Підготовка даних: Це включає збір, очищення та підготовку вхідних даних для тренування. Це може включати масштабування, нормалізацію та розбиття даних на тренувальний, валідаційний та тестувальний набори. У нашому випадку підготовку даних було розглянуто у розділі про розробку нейронної мережі для екстраполяції складності ігрової ситуацій. Розбиття даних на тренувальний, валідаційний та тестувальний, відбувається таким чином:

- дані тренування, результати проходження штучними гравцями;
- дані валідації, результати проходження гравцем;
- дані тестування, буде згенеровано ігрові ситуації нейронною мережею та пройдено гравцем з метою незалежного тестування.

Вибір архітектури мережі: Обрання кількості шарів, кількості нейронів у кожному шарі, видів функцій активації тощо. Цей етап було виконано у розділі розробки моделі нейронної мережі для генерації ігрових ситуацій.

Ініціалізація ваг: Початкові значення ваг визначаються випадковим чином або за допомогою певних методів ініціалізації. В нашому випадку усі ваги було ініціалізовано нульовими значеннями, що надає однакову початкову точку для тренування нейронних мереж, що корисно при порівнянні ефективності різних методів навчання.

Процес навчання: Мережа навчається шляхом передачі даних через мережу, обчислення вихідних значень та порівняння їх з очікуваними значеннями. Ваги мережі оновлюються відповідно до виявленої помилки.

Для нашого випадку необхідно створити нову функцію для тренування нейронної мережі, адже для генерації наступної ігрової ситуації необхідно надати

дані про попередньо згенеровані рівні, лістинг коду для тренування нейронної мережі можна знайти в додатку В.3.

Розглянемо частину коду, що відповідає за тренування нейронної мережі на протязі одної ітерації для однієї групи даних.

```
def train_step(self, data):
    x, y, sample_weight = data_adapter.unpack_x_y_sample_weight(data)
```

Спершу необхідно отримати вхідні дані та вихідні дані, у нашому випадку вони не використовуються, оскільки порівняння відбувається не з дійсним значенням, а з результатом проходження рівня.

```
user = UserTester()
```

Ініціалізація штучного гравця, код надано в додатку В.2.

```
with tf.GradientTape() as tape:
    save_pred = []
    save_params = []
    for i in range(16):
```

Етап прогрівання нейронної мережі, у нашому випадку, це генерація 16 ігрових ситуацій та проведення їхнього аналізу, на цьому етапі відсутнє тренування.

```
        if len(save_pred):
            tmp = x[0][:, i:]
            for j in save_pred:
                tmp = tf.concat([tmp, [j]], axis=1)
            tmp_2 = x[1][:, i:]
            for j in save_params:
                j = tf.convert_to_tensor([[j]])
                tmp_2 = tf.concat([tmp_2, [j]], axis=1)
            y_pred = self((tmp, tmp_2), training=False)
```

```
            save_pred.append(y_pred)
            save_params.append(tf.convert_to_tensor(user.get_total(y_pred)))
        else:
            y_pred = self(x, training=False)
            save_pred.append(y_pred)
            save_params.append(tf.convert_to_tensor(user.get_total(y_pred)))
```

Етап генерації ігрових ситуації у нашому випадку створюється 16 рівнів з матою пришвидшення тренування нейронної мережі, після кожної генерації відбувається етап тренування.

```
    for i in range(16):
        with tf.GradientTape() as tape:
            _tmp = tmp[:, i:]
            _tmp_2 = tmp_2[:, i:]
            y_pred = self((_tmp, _tmp_2), training=True)
            self.loss.user = user
            loss = self.compute_loss((_tmp, _tmp_2), y, y_pred, sample_weight)
            tmp = tf.concat([tmp, [y_pred]], axis=1)
            tmp_2 = tf.concat([tmp_2, [tf.convert_to_tensor([[user.get_total(y_pred)]])]], axis=1)
```

Етап тренування нейронної мережі, що проводить аналіз результатів функції втрат та змін у попередніх ітераціях.

```
        self._validate_target_and_loss(y, loss)
        self.optimizer.minimize(loss, self.trainable_variables, tape=tape)
```

В кінці необхідно повернути результати виконання у цієї групи даних.

```
    return self.compute_metrics(x, y, y_pred, sample_weight)
```

Оцінка результатів: Після кожної ітерації модель оцінюється на валідаційному наборі для оцінки її продуктивності. Це допомагає уникнути перенавчання та підтвердити, чи модель загалом вчиться правильно. В нашому

випадку це порівняння з результатами отриманими на основі проходження ігрових ситуацій гравцем також результати функції втрат на протязі епохи використовуються для порівняння з найкращими, у випадку покращення результатів збережеться кращий результат нейронної мережі.

Тестування моделі: Після завершення тренування модель тестується на раніше не бачених даних для оцінки її загальної продуктивності та узагальнювання. Для тестування розробленої нейронної мережі буде використано гру розроблену в розділі 3.

Отже було розглянуто функцію тренування на протязі обробки даних одного гравця у одні ітерації.

2.6 Навчання нейронної мережі

Перед початком навчання нейронної мережі слід обрати алгоритм оптимізації. Розглянемо самі відомі методи оптимізації навчання нейронної мережі.

Градієнтний спуск (Gradient Descent): Основний та базовий алгоритм оптимізації, який оновлює ваги в напрямку, протилежному градієнту функції втрат. Має різні варіанти, такі як:

Стохастичний градієнтний спуск (SGD): Оновлює ваги для кожного навчального прикладу окремо, що може прискорити зближення до мінімуму.

Міні-пакетний градієнтний спуск (Mini-Batch GD): Оновлює ваги партіями даних, що сприяє балансу між швидкістю та стабільністю навчання.

Adam (Adaptive Moment Estimation): Комбінує ідеї інших методів оптимізації, таких як момент та RMSprop. Використовується для адаптивної настройки швидкості навчання.

RMSprop (Root Mean Square Propagation): Адаптивний метод, який регулює швидкість навчання для кожного параметра окремо, використовуючи квадратний корінь середньоквадратичного градієнта.

Adagrad (Adaptive Gradient Algorithm): Алгоритм, який адаптивно налаштовує швидкість навчання для кожного параметра, залежно від частоти його оновлення.

Adadelta: Алгоритм, схожий на Adagrad, але з меншою чутливістю до старих градієнтів. Використовує експоненційне згладжування.

На основі цього слід використати алгоритм Adam [23], оскільки в більшості випадків він надає результати швидше ніж RMSprop та має краще розуміння про втрати ніж алгоритми градієнтного спуску. Також наша задача не потребує врахування частоти оновлення параметрів. Також на основі використання різних алгоритмів оптимізації у «MNIST digit recognition», рисунок 2.7.

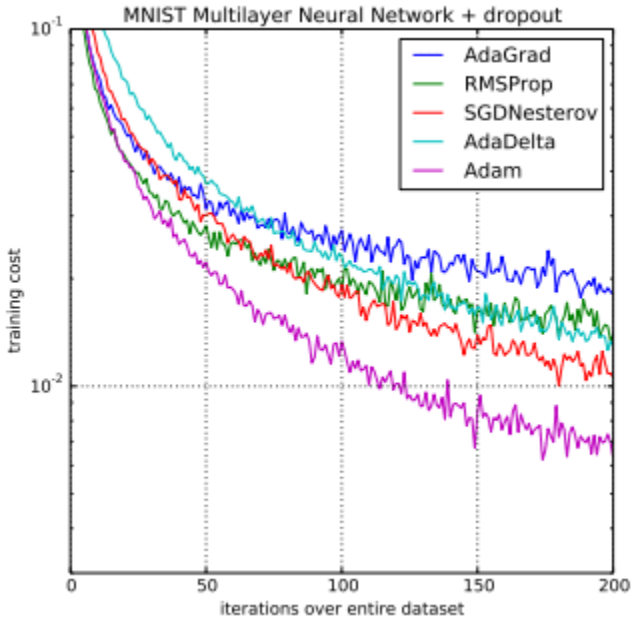


Рисунок 2.7 – Порівняння зміни втрат різних функцій оптимізації

Таким чином потрібно почати навчання нейронної мережі.

```
def train(input_data_1, input_data_2, out_data):
    model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="nn_1_1")
    model_holder.load_model()
    model_holder.model.compile(
```

```
    loss=loss_function(),
    optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
    metrics=[], run_eagerly=True
)
history = model_holder.fit(
    x_room=input_data_1,
    x_metric=input_data_2,
    y=test_y,
    epochs=300
)
```

Цей код було використано для навчання нейронної мережі, спершу відбувається процес ініціалізації моделі, у випадку наявності збереженої мережі збережені параметри зчитуються з файлів.

Потім відбувається процес збірки нейронної мережі в якому вказуються: функції втрат, функції оптимізації та метрики, в нашому випадку слід також вказати «run_eagerly», оскільки в нас використовується складний метод визначення втрат.

У кінці вказується вхідні, вихідні дані та кількість ітерації навчання нейронної мережі, в нашому випадку 300. Вхідні дані також використовуються для вказування кількості штучних гравців, при навчанні було використано 32 штучних гравця.

Розмірність ігрової ситуації, що буде генеруватись 11 на 11 клітинок з 3 можливими параметрами, що дозволить розташувати об’єкти 3 типів, нехай це буде стіна, ворог з ближньою атакою та ворог з дальньою атакою.

Отже в цьому розділі було проведено навчання нейронної мережі за використанням штучних гравців та методу оптимізації Adam.

2.7 Метод використання нейронної мережі для аналізу складності ігрових ситуацій

У розділі 2.2 було розглянуто розробку моделі для аналізу складності ігрових ситуацій для середньо статистичного користувача з використанням шарів згортки, на основі цього розробимо метод використання нейронної мережі для аналізу складності ігрових ситуацій.

Спершу слід змінити кількість шарів згортки в залежності від розмірності ігрової ситуації, що використовується грою. Для квадратної матриці, у більшості випадків, слід розміщувати таку кількість шарів згортки: розмір матриці поділено на два з округленням в меншу сторону. Наприклад при наявності трьох мірної матриці з розмірами одинадцять на одинадцять на три, слід п’ять шарів згортки:

$$\frac{11}{2} = 5.5 => 5$$

Також необхідно вказати виходи нейронної мережі, що може включати у себе передбачення часу проходження рівня, отримане пошкодження гравцем, тощо. У випадку магістерської кваліфікаційної роботи було використано вирівняння виходи шарів згортки, оскільки дані напряму передавались до нейронної мережі для створення ігрових ситуацій.

В наступному етапі слід провести навчання нейронної мережі, що потребує дані про структуру ігрових ситуацій та метрики майстерності гравців при їх проходженні.

Для тренування слід використати функцію втрат середньоквадратичної похибки, основною сферою застосування є задачі регресії. Оскільки це добре диференційована функція, що спрощує процес навчання за допомогою градієнтного спуску. Також функція середньоквадратичної похибки є менш чутливою до випадкових шумів або викидів в даних порівняно з іншими функціями втрат.

Обчислення середньоквадратичної похибки досить просте і швидше за інші функції втрат, такі як категоріальна крос-ентропія, яка використовується для класифікації.

Після завершення тренування нейронної мережі вона може бути використана для аналізу складності ігрової ситуації для середньо статистичного гравця, що був використаний у процесі тренування. Для цього необхідно передати нейронній мережі ігрову ситуації, що необхідно проаналізувати після обчислень нейронною мережею буде видано складність рівня, що може бути використана для порівняння з метриками конкретного гравця, що покращить аналіз його майстерності.

Отже було розроблено метод використання нейронної мережі для аналізу складності ігрової ситуації.

2.8 Метод використання нейронної мережі для створення ігрових ситуацій

У розділах 2.3 – 2.6 було розглянуто розробку моделі для створення нейронної мережі та її тренування, на основі цього розробимо метод використання нейронної мережі для створення ігрових ситуацій.

Спершу нам необхідно виставити розміри виходів нейронної мережі, а саме щільного шару та шару зміни форми. Для МКР, що використовує трьох мірну матрицю з розмірами одинадцять на одинадцять на три, слід вказати кількість одиниць для щільного шару на триста шістдесят три та для шару зміни форми, форму матриці у одинадцять, одинадцять, три.

Далі нам необхідно вказати входи для нейронної мережі, один з яких відповідає за майстерність конкретного гравця на пройдених ним ігрових ситуація, інший відповідає за складність пройдених рівнів. У нашому випадку для цього використовується нейронна мережа, метод використання якої, знаходиться у розділі 2.7.

Далі нам необхідно провести навчання нейронної мережі, що включає у себе створення штучних гравців, що проводять аналіз згенерованих рівнів на основі

чого визначаються втрати нейронної мережі, створення штучних гравців для тренування нейронної мережі було розглянуто у розділі 2.4.

При тренування слід використати змінену функцію тренування, що буде виконувати прогрівання нейронної мережі перед створенням ігрової ситуації, що було розглянуто у розділі 2.5.

Перед тренуванням необхідно обрати функцію оптимізації тренування нейронної мережі, що було розглянуто у розділі 2.6.

Для навчання нейронної мережі слід вказати кількість ітерацій для тренування та кількість штучних гравців, у цій роботі було використано п’ятсот ітерація тренування та тридцять два штучних гравця.

При використанні нейронної мережі слід надати дані про попередні пройдені гравцем рівні та його параметри майстерності для кожного з них, що буде використано нейронною мережею для створення наступного рівня.

Для передачі даних між нейронною мережею та кінцевим додатком слід використати проміжний інтерфейс, наприклад HTTP сервер, що буде отримувати дані про пройдені рівні та передавати їх до нейронної мережі, така реалізація спростить інтеграцію нейронної мережі до гри та дозволить запустити нейронну мережу в хмарному середовищі.

2.9 Висновки

Розроблено модель нейронної мережі, що виконує аналіз згенерованих кімнат та екстраполює незалежну від гравця складність. Розроблено модель нейронної мережі, що проводить аналіз гри гравця на протязі певного періоду та на основі цього проводить генерацію ігрових ситуацій. Розроблено функцію втрат, що визначає на скільки результат нейронної мережі виконує поставлену задачу. Та було проведено тренування нейронної мережі.

У результаті було отримано натреновану нейронну мережу, що аналізує майстерність гравця та пройдені рівні для генерації наступних ігрових ситуацій.

3 РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ ДЛЯ ДЕМОНСТРАЦІЇ РЕЗУЛЬТАТІВ ВИКОРИСТАННЯ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору рушія для розробки ігор

Ігровий рушій [24] – це програмне забезпечення, яке використовується для створення відеоігор. Це складна система, яка надає інструменти для розробки ігрового середовища, включаючи графіку, фізику, штучний інтелект, звук та багато іншого/

Ігрові рушії включають в себе різноманітні компоненти, такі як:

- Графічний рушій: Відповідає за оброблення графіки в грі, включаючи рендеринг 2D та 3D об'єктів, освітлення, текстури, анімацію тощо.
- Фізичний рушій: Відповідає за симуляцію фізики в ігровому світі, таку як рух об'єктів, зіткнення, гравітацію тощо.
- Штучний інтелект: Дозволяє створювати поведінку в грі для персонажів, ворогів або інших об'єктів.
- Звуковий рушій: Відповідає за оброблення звукових ефектів, музики та аудіо в грі.
- Інструменти розробки: Це набір програмних інструментів, що допомагають розробникам створювати гру, включаючи редактори рівнів, інструменти моделювання, скриптові мови тощо.

Існує багато різних ігрових рушіїв, кожен з яких має свої особливості, переваги та недоліки. Деякі з найпопулярніших ігрових рушіїв на сьогоднішній день включають Unity, Unreal Engine, CryEngine, Godot.

Unity [25]:

- Простота використання: Має дружній інтерфейс та широкий спектр документації, що допомагає початківцям.
- Мови програмування: Підтримує C#.
- Екосистема: Має велику спільноту користувачів, доступ до різноманітних ресурсів та великий магазин активів.

Godot:

- Безкоштовність та відкритий код: Це безкоштовний рушій з відкритим вихідним кодом.
- Мови програмування: Підтримує мови, такі як GDScript, яка є схожою на Python, та C#.
- Малоресурсний: Добре підходить для проектів малого розміру та працює добре на менш потужних пристроях.

Unreal Engine:

- Графіка та візуалізація: Має потужні засоби для створення вражаючої графіки та візуалізації.
- Мови програмування: Використовує мову програмування C++.
- Розширені можливості: Підтримує різні функції для розширення гри, але може бути складнішим для проектів малого масштабу.

CryEngine:

- Візуальні можливості: CryEngine відомий своєю потужною графікою та реалістичною візуалізацією.
- Вимоги до ресурсів: Через високу якість графіки, CryEngine може бути вимогливим до обладнання.
- Навчання та підтримка: CryEngine може бути менш доступним для новачків через менш розвинену спільноту та документацію.
- Функціональні можливості: Він має досить розширені можливості, проте це може зробити процес розробки складнішим для менших проектів.

При урахуванні, що головна мета комп’ютерної гри, що розробляється, показати можливості використання нейронної мережі для генерації ігрових ситуацій, слід використати ігровий рушій Unity, оскільки він надає достатню потужність у розробці та не вимагає великої затрати часу.

3.2 Розробка об’єктів для генерації ігрової ситуації

Для використання результатів генерації ігрових ситуацій нейронної мережі, нам необхідно створити об’єкти та текстури для них, що буде розміщено, ці об’єкти включають у себе:

- кімната у якій буде розміщено об’єкти;
- камера, точка з якої буде передаватись гравцю інформація про стан гри;
- освітлення, що наддасть грі об’ємності та полегшить розпізнання зображення;
- колона, перший тип об’єкта, що буде використовуватись для генерації ігрової ситуації;
- ворог з ближньою атакою;
- ворог з дальньою атакою та дальні атаки для нього;

Спершу було додано напрямлене світло з м’якими тінями під нахилом в 50 градусів по осі X та -30 по осі Y.

Для створення кімнати, слід зробити її розмір з відношенням сторін близьким до стандартного 16:9, що усуне проблему з необхідністю додання обрамлення. Рівень складається з 4 стін та підлоги.

Створено текстури розміром 64 на 64 пікселі та додано їх на матеріал з повторенням відповідним до розміру об’єктів.

Далі було додано об’єкт колони, що складається з двох об’єктів, циліндра та кола, що потрібно для однорідної текстури, матеріал використаний для циліндра відповідний до матеріалу стін зі зміненим коефіцієнтом повторення.

Також було додано два об’єкти у формі капсул, що будуть виступати, як вороги, червоного кольору буде ворог з ближньою атакою, фіолетовий з дальньою.

В завершення було створено атаку для дальнього ворога з доданим компонентом фізичного тіла для аналізу, коли об’єкт взаємодіє з іншим.

Результати створення об’єктів, можна переглянути на рисунку 3.1, що показує рівень, у форматі який буде спостерігатися гравцем, та 4 об’єкти ворог з ближньою атакою, колона, ворог з дальньою атакою та його снарядом.

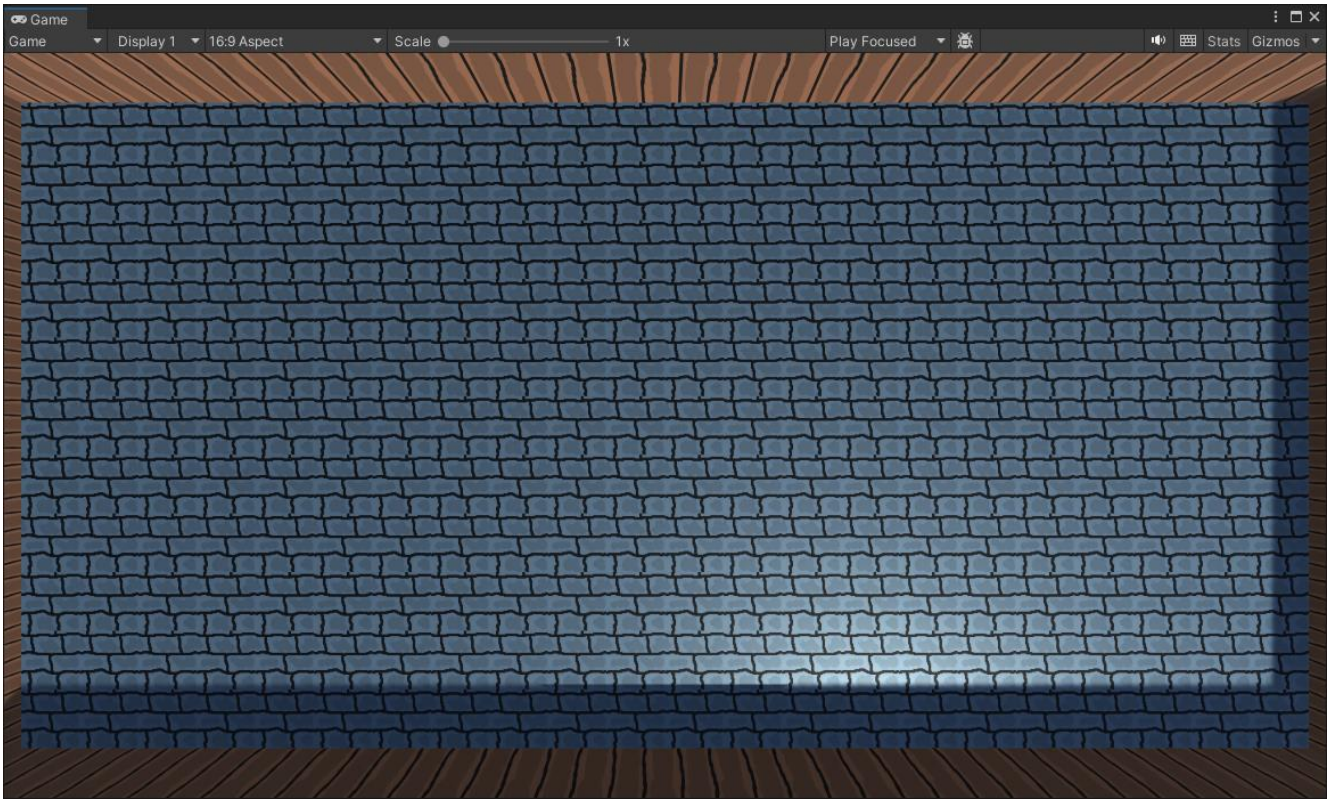


Рисунок 3.1 – Об’єкти для імплементації ігрової ситуації

Отже було створено усі необхідні об’єкти для обробки результату нейронної мережі з метою показу ігрової ситуації.

3.3 Використання штучного інтелекту для рухомих об’єктів

Для того, щоб гравець завершити ігрову ситуацію, необхідно реалізувати штучний інтелект, що буде відповідати за рух ворогів, нанесення пошкодження та пошук гравця.

Повний лістинг, що відповідає за контроль ворогів знаходиться у додатках Г.1, Г.2, Г.3.

Спочатку ворог має отримати дані про гравця та додати собі модуль, що буде відповідати за переміщення.

Пошук гравця, відбувається за тегом, що був виданий гравцю. У випадку відсутності гравця, виводиться сповіщення про помилку.

Ці операції відбуваються при створені об’єкта ворога.

```
void Start()
{
    controller = gameObject.AddComponent<CharacterController>();
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    if (player == null){
        Debug.Log("error");
    }
}
```

Наступне, що необхідно реалізувати, це переміщення у напрямку гравця з швидкістю, що може бути зміненою, ворог зупиниться при наближені до гравця на радіус його атаки, наприкінці додано гравітацію, ці обчислення викликаються раз за кадр.

Наступне, що необхідно реалізувати це умова знищення ворога та надання можливості атаки.

У випадку коли здоров’я ворога спускається нижче нуля видаляємо об’єкт ворога та усі під’єднані до нього об’єкти.

У випадку коли ворог знаходиться у радіусі атака він звертається до скрипту, що відповідає за атаку.

```
void Update()
{
    if(health <= 0){
        Destroy(gameObject);
    }
    distance = Vector3.Distance (player.transform.position, transform.position);
    if (distance < gameObject.GetComponent<Attack>().range && cooldown == 0)
    {
        cooldown = 720;
        gameObject.GetComponent<Attack>().enabled = true;
    }
    if (cooldown > 0){
        cooldown -= 1;
    }

}
```

Блок схему алгоритму руху представлено на рисунку 3.2.

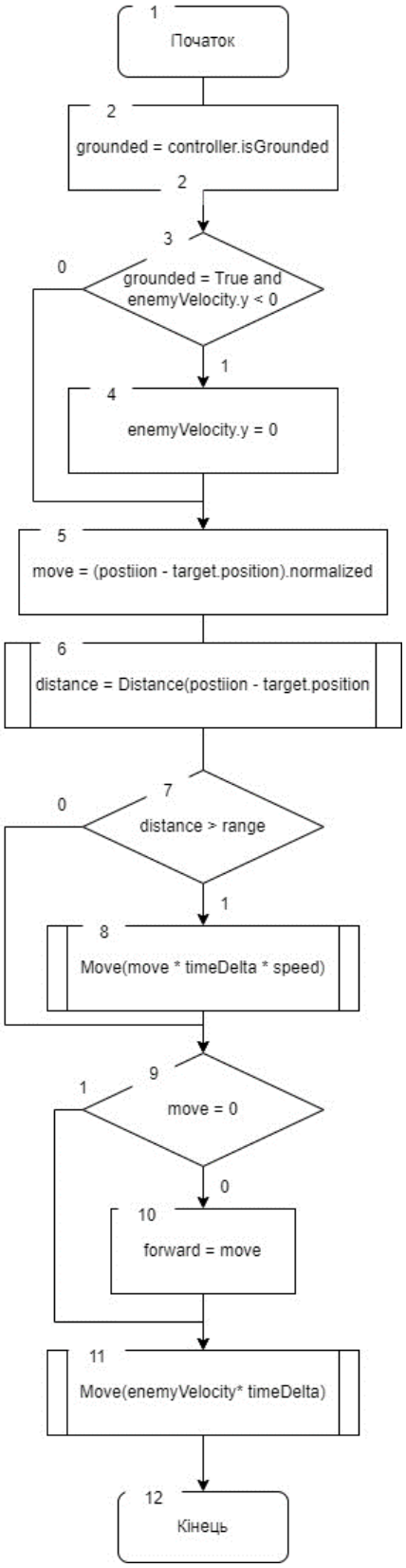


Рисунок 3.2 – Алгоритм руху ворога до цілі

Розробимо модуль, що відповідає за ближню атаку, рисунок 3.3.

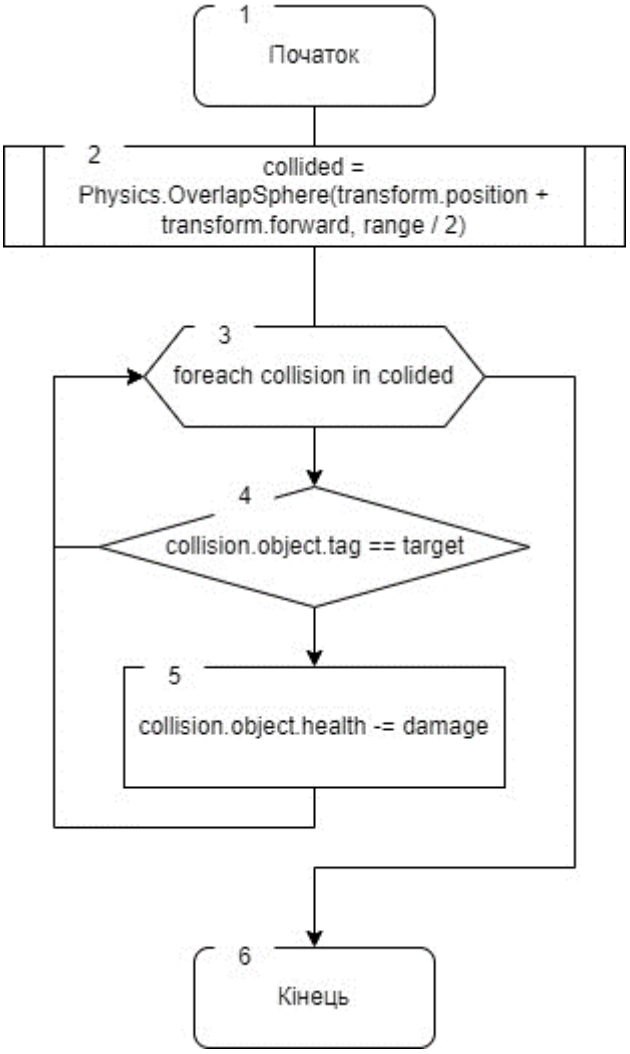


Рисунок 3.3 – Алгоритм ближньої атаки

Дальня атака складається з двох складових перша створює снаряд перед ворогом у напрямлені до цілі. Другий відповідає за рух снаряду та його знищення при зіткненні.

Отже було реалізовано штучний інтелект, що відповідає за керування ворогом, що намагається приблизитись до гравця та створено модулі, що відповідають за різні типи атаки.

3.4 Розробка модулю для генерації ігрових ситуацій за допомогою нейронної мережі

Для імплементації модулю, що буде обробляти дані отримані від нейронної мережі нам необхідно визначити головні його задачі.

Задача 1. Тримати у пам’яті усі об’єкти згенеровані під час створення ігрової ситуації та аналізувати наявність «живих» ворогів та видаляти їх при завершені рівня.

Задача 2. Запуск нейронної мережі та передача історії гри на вхід.

Задача 3. Обробка ігрової ситуації отриманої від нейронної мережі та ініціалізація об’єктів.

Блок схема рішення для задачі 1, рисунок 3.4.

Для рішення другої задачі нам необхідно реалізувати, сервер на який буде відправлятись запит з часом проходження рівня та отримувати у відповідь згенеровану кімнату у форматі байтів.

Сервер до якого відбувається з’єднання знаходиться на “localhost” на порті 25000. Сервер розроблений за допомогою Flask. Лістинг коду серверу знаходиться у додатку Д.

На сервері існує один шлях, який очікує отримати у параметрах час проходження ігрової ситуації, що було попередньо згенеровано. У відповідь запускається нейрона мережа, що обраховує затрачений час та історію згенерованих ситуацій, на основі чого генерує наступну ситуацію.

Запит реалізовано за рахунок використання «UnityEngine.Networking», що надає можливість реалізації HTTP запитів.

```
List<IMultipartFormSection> formData = new List<IMultipartFormSection>();
formData.Add(new MultipartFormDataSection("time=" + timespent.ToString("R")));
UnityWebRequest www = UnityWebRequest.Post("localhost:25000", formData);
yield return www.SendWebRequest();

if (www.result != UnityWebRequest.Result.Success) {
   Debug.Log(www.error);
}
```

```
else {
    Debug.Log(www.downloadHandler.text);
    byte[] results = www.downloadHandler.data;
}
```

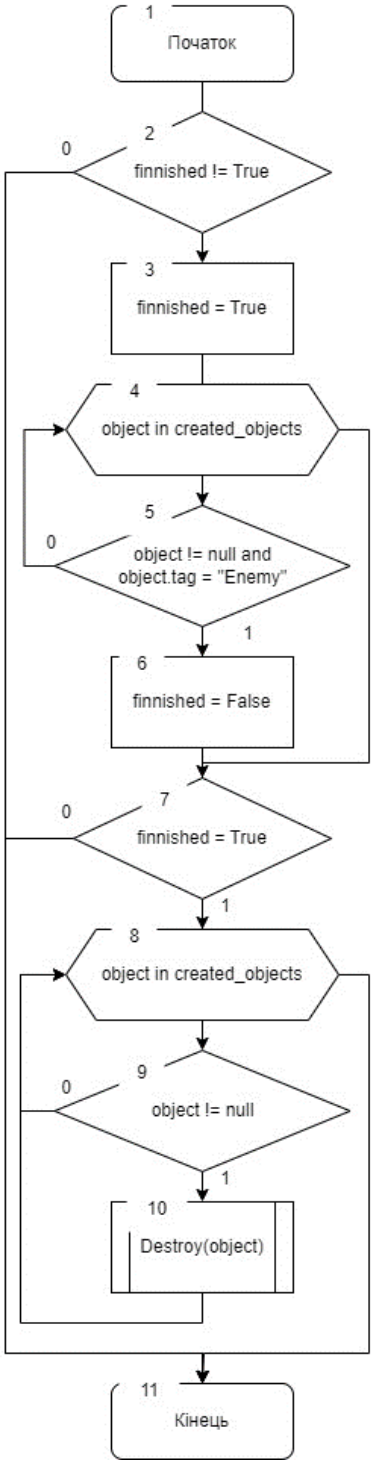


Рисунок 3.4 – Алгоритм керування створеними об’єктами

Для виконання задачі 3 необхідно пройти дані, що було отримано від нейронної мережі та згенерувати вказаний об’єкт у відповідному місці, алгоритм зображено на рисунку 3.5.

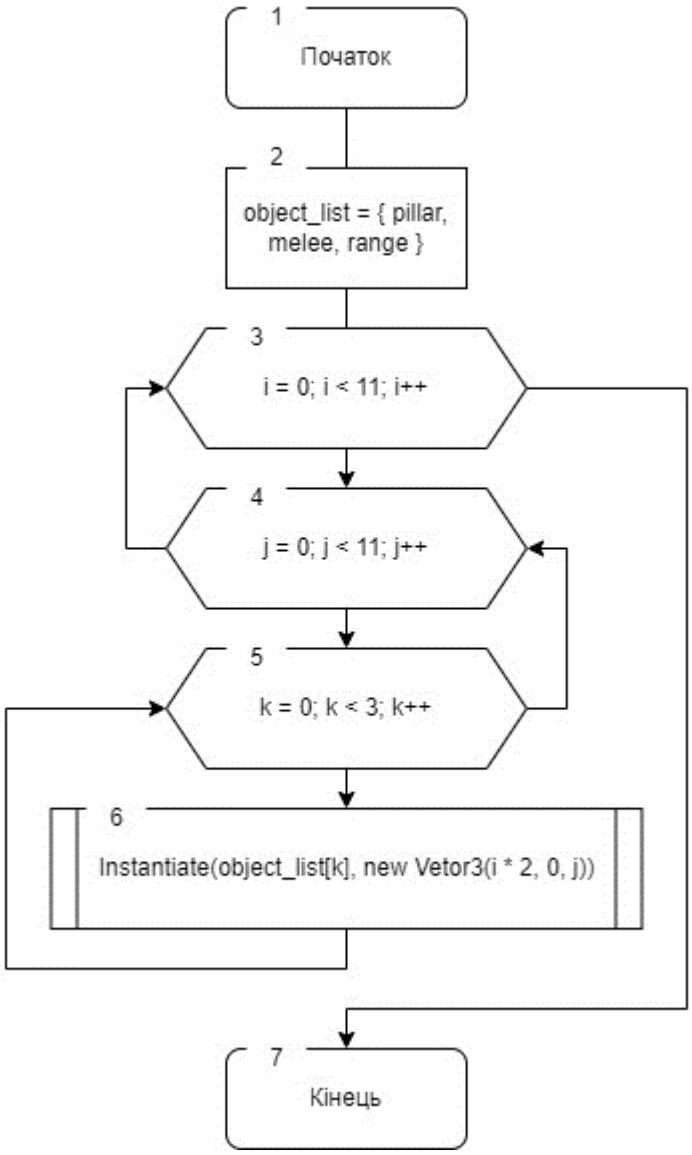


Рисунок 3.5 – Алгоритм створення ігрової ситуації

Отже було реалізовано модуль, що відповідає за роботу з нейронною мережею, створення ігрової ситуації на основі отриманих даних та керування створеними об’єктами.

3.5 Розробка інтерфейсу користувача та системи керування

Інтерфейс користувача в іграх - це усе, що гравець бачить на екрані під час гри, крім основної ігрової частини.

Він повинен бути інтуїтивним та легко зрозумілим, не заважати геймплею та не перегружати гравця надмірною інформацією. Інтерфейс користувача повинен бути пристосованим до жанру гри та передавати необхідну інформацію без зайвого шуму.

Дизайн інтерфейсу користувача може бути різним в залежності від жанру гри. Наприклад, у шутерах першої особи HUD (Head-Up Display) може показувати зброю та кількість набоїв, в той час як у рольових іграх може бути більше уваги приділено інвентарю та характеристикам персонажа.

Для показу статусу здоров’я гравця було додано слайдер зеленого кольору, що розташований у верхньому лівому куті екрану, рисунок 3.6.

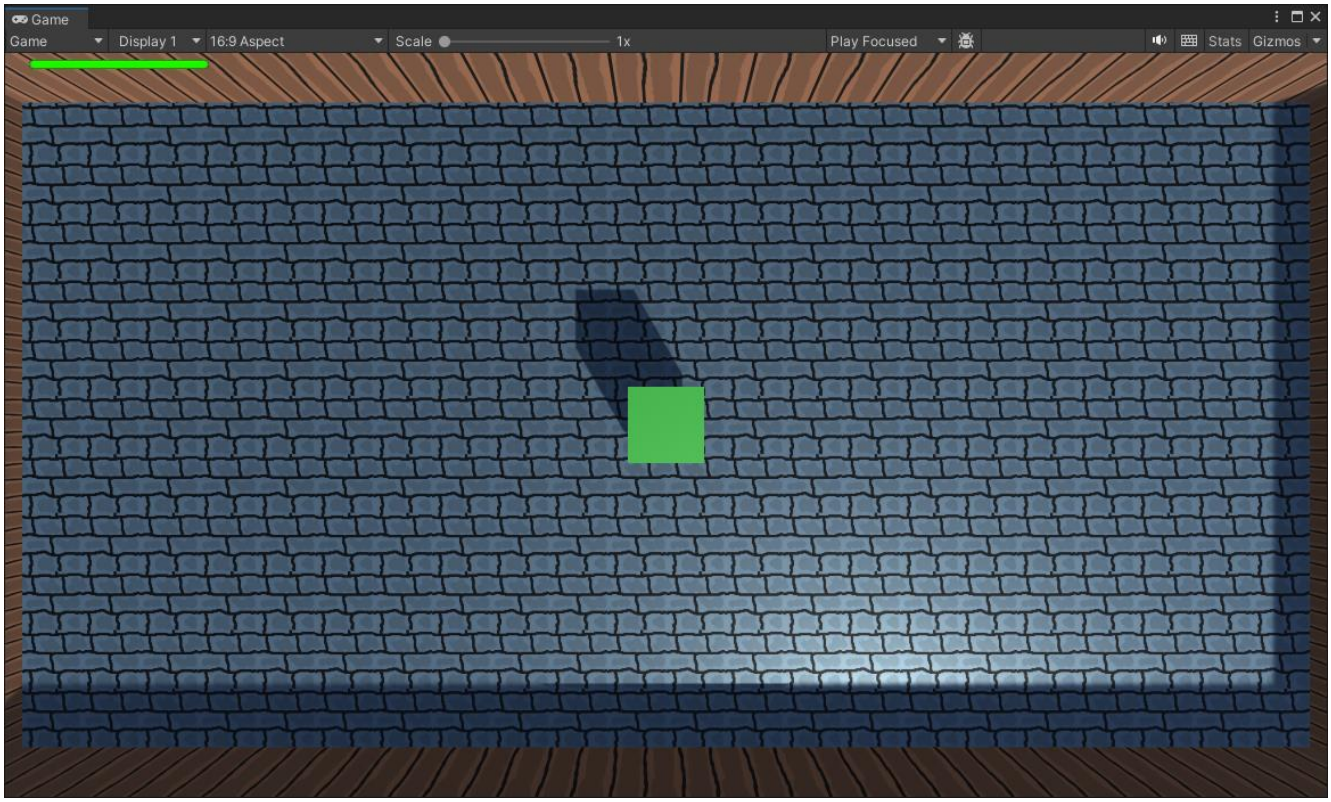


Рисунок 3.6 – Інтерфейс користувача

Наступне потрібно реалізувати рух гравця та надати йому можливість атаки, рисунок 3.7.

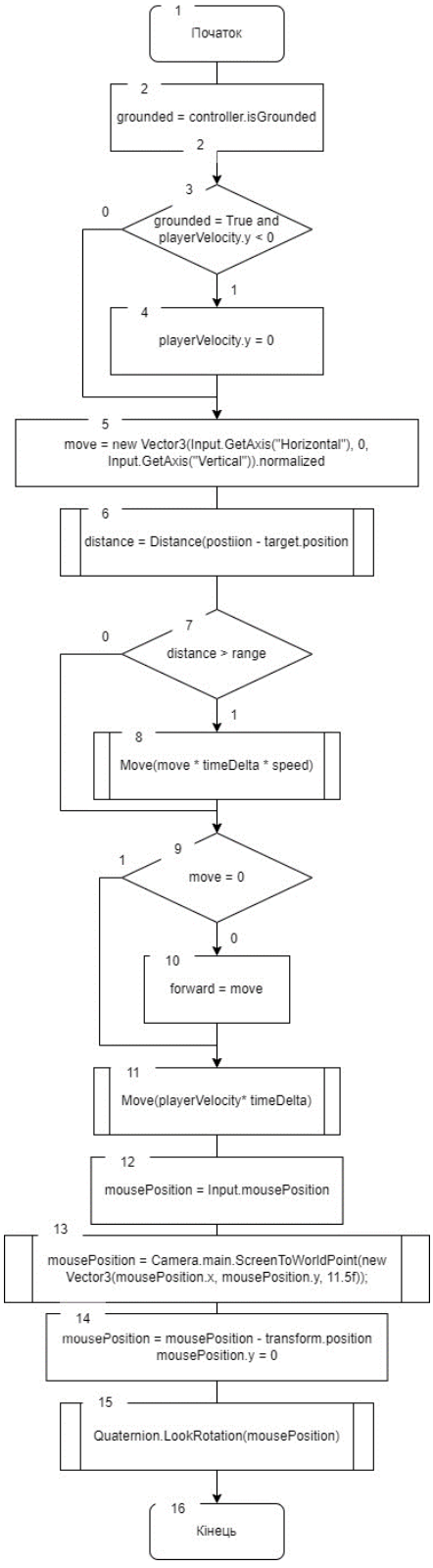


Рисунок 3.7 – Алгоритм руху гравця

Алгоритм руху гравця подібний до алгоритму руху ворогів за винятком отримання напрямку руху з вхідних даних клавіатури та добавлення блоку повернення об’єкту гравця у сторону курсору миші.

Для атаки гравця було використано модуль ближньої атаки ворогів, для обробки вхідних сигналів гравця та керування його здоров’я розроблено модуль керування об’єкта гравця, рисунок 3.8.

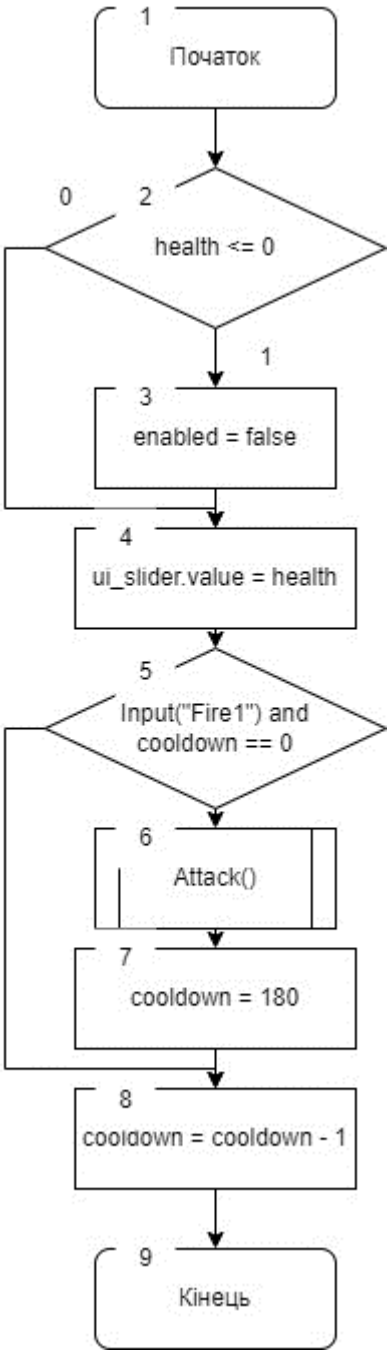


Рисунок 3.8 – Алгоритм контролю об’єкта гравця

Отже було розроблено інтерфейс користувача та додано модуль для контролю його управління та атаки.

3.6 Розробка HTTP серверу для нейронної мережі

Для використання нейронної мережі грою, необхідно реалізувати інтерфейс, що буде зберігати дані про пройдені користувачем рівні та буде передавати створенні нейронною мережею.

Існує декілька способів використання нейронної мережі за допомогою Unity.

Unity Python Interoperability (безпосередньо в Unity) [26]. Unity підтримує використання скриптів Python, використовуючи плагін DLR (Dynamic Language Runtime). Щоб це зробити, необхідно використовувати бібліотеку IronPython, яка є реалізацією Python для .NET.

Використання subprocess (зовнішній виклик скрипта) [27]. Можливо викликати скрипт Python з Unity, використовуючи бібліотеку System.Diagnostics для виклику зовнішніх процесів.

Використання HTTP-протоколу для взаємодії між Unity та Python.

Виконання python скрипту на пряму викликає можливі проблеми з безпекою гри, оскільки мова програмування python може бути змінена при виконанні та відсутність компіляції, що може викликати проблеми пошуку вірусів. Тому слід реалізувати HTTP сервер, що надає додаткові переваги, а саме.

Розділення Фронтенду та Бекенду. HTTP дозволяє розділити фронтенд (Unity) та бекенд (Python) компоненти додатку. Це забезпечує більш чисту архітектуру та полегшує модифікації та розширення.

Масштабованість. HTTP є стандартним протоколом для взаємодії між серверами та клієнтами. Це дозволяє вам легко масштабувати ваші системи, додавати нові компоненти та інтегрувати інші служби.

Гнучкість та Надійність. HTTP протокол є надійним та гнучким, що дозволяє вам передавати дані в різних форматах (наприклад, JSON) та використовувати різні методи HTTP (GET, POST, PUT, DELETE) для різних завдань.

Підтримка Кросс-Платформеності. Використання HTTP дозволяє вам забезпечити кросс-платформеність між Unity та Python. Оскільки HTTP є стандартним протоколом, його підтримують багато платформ та мов програмування.

Сервіс-Орієнтований Підхід. HTTP дозволяє вам реалізувати сервіс-орієнтований підхід, де окремі компоненти програми можуть надавати та споживати послуги через HTTP API. Це полегшує побудову розподілених систем.

Легка Інтеграція з Іншими Сервісами. Використання HTTP уможливлює інтеграцію з різними інтернет-сервісами, такими як бази даних, хмарні сервіси, API сторонніх служб тощо.

Розглянемо різні фреймворки для реалізації HTTP серверу.

Flask є легким та простим у використанні фреймворком. Він надає основні можливості для створення веб-додатків та RESTful API. Flask дозволяє швидко розпочати проект та надає гнучкість для розширення.

Django є повноцінним веб-фреймворком, який надає багато вбудованих функцій, таких як аутентифікація, ORM, адмін-панель та інші. Він добре підходить для великих та складних веб-додатків.

FastAPI є новим та ефективним фреймворком, який дозволяє швидко розробляти веб-додатки та RESTful API. Він використовує типи Python для автоматичної генерації документації та перевірки типів.

Tornado призначений для створення масштабованих та асинхронних веб-додатків. Це дозволяє обробляти тисячі одночасних з'єднань та використовується для створення реального часу (real-time) додатків.

Bottle - це мінімалістичний фреймворк, який легко встановлюється та використовується. Він добре підходить для невеликих проектів або додатків, де вам не потрібні великі фреймворки.

Отже слід використати flask, оскільки він надає наступні переваги:

Легкість використання. Flask є дуже легким та простим у використанні фреймворком. Він має мінімальний набір компонентів, що дозволяє швидко розпочати роботу без надмірного конфігурування.

Гнучкість та Розширюваність. Flask надає базовий набір функціональності, але в той же час він дозволяє розширювати його за допомогою різноманітних розширень і бібліотек, забезпечуючи гнучкість для розробки за вашими потребами.

Близькість до Python. Flask дозволяє використовувати всі можливості мови програмування Python. Він не нав'язує власні правила та архітектуру, що полегшує розробку та відладку коду.

Ідеальний для Малих та Середніх Проектів. Якщо проект є невеликим чи середнім за розміром, і немає потреби у великій функціональності та складності, Flask може бути ідеальним вибором.

RESTful Розробка. Flask добре підтримує створення RESTful API, що робить його хорошим вибором для створення мікросервісів або простих серверів API.

Активна Спільнота та Документація. Flask має велику та активну спільноту, а також добре написану документацію. Це полегшує вивчення та розвиток проектів на основі Flask.

Розробка Прототипів та Малих Проектів. Завдяки своїй простоті та швидкості розгортання, Flask ідеально підходить для розробки прототипів та малих проектів.

Для отримання запити необхідно вказати шлях, що передає дані про час затрачений на проходження попереднього згенерованого рівня та перетворить вихід нейронної мережі в формат, що можливо передати за допомогою HTTP протоколу. Лістинг коду обробки запитів знаходиться у додатку Д.1.

Для запуску нейронної мережі необхідно передати дані про історію гравця та завантажити навчену нейрону мережу з файлу. Лістинг коду виклику нейронної мережі для створення ігрових ситуацій знаходиться у додатку Д.2.

3.7 Висновки

У цьому розділі обґрунтовано вибір ігровиго рушія Unity для розробки гри з метою показу результатів виконання нейронної мережі.

Розроблено об’єкти з метою створення ігрових ситуації, до яких входять: кімната в якій створюються об’єкти; колони, перший об’єкт, що генерується

нейронною мережею; ворог з ближньою атакою та системою, що відповідає за атаку ближнього бою; ворог з дальньою атакою та системою створення снарядів та перевірки на зіткнення.

Розроблено:

1. Штучний інтелект ворогів, що намагається приблизитись на відстань атаки до гравця.
2. Модуль для роботи з нейронною мережею та генерації ігрової ситуації за допомогою об’єктів.
3. Інтерфейс користувача та модуль для керування об’єктом користувача.
4. HTTP сервер, що надає можливість використання нейронної мережі без потреби інтеграції системи в гру.

У результаті було створено гру, що може бути використаною для показу результату роботи нейронної мережі.

4 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

4.1 Методи тестування програмного забезпечення

Тестування комп'ютерних програм – це важливий етап розробки програмного забезпечення, спрямований на виявлення помилок та перевірку відповідності програмних продуктів вимогам. Існує кілька методів тестування, які можна застосовувати на різних етапах розробки. Ось декілька основних методів тестування.

Модульне тестування (Unit Testing) [28]. Включає в себе тестування окремих модулів або функцій програми. Мета – перевірити, чи працює кожен модуль ізольовано від інших.

Переваги:

- швидкість виконання, тести для окремих модулів можуть бути виконані дуже швидко;
- легкість у виявленні помилок, дозволяє виявити ізольовані помилки в окремих частинах коду.

Недоліки:

- не виявляє взаємодію, не виявляє помилок, пов'язаних із взаємодією між модулями;
- не відображає системні аспекти, не оцінює системні аспекти та аспекти взаємодії з іншими компонентами.

Інтеграційне тестування (Integration Testing). Перевірка взаємодії між різними модулями чи компонентами системи. Мета – виявити помилки, які можуть виникнути при взаємодії різних частин програми.

Переваги:

- виявлення помилок взаємодії, дозволяє виявити помилки, які виникають при взаємодії між модулями;
- тестування великої частини системи, дозволяє перевірити, як велика частина системи працює разом.

Недоліки:

- складність тестування, вимагає багато часу і зусиль для створення та виконання тестів із великою кількістю компонентів;
- важкість визначення причини помилок, важко визначити, який саме модуль викликав помилку при взаємодії.

Функціональне тестування (Functional Testing) [29]. Випробовує функціональність програми, перевіряючи, чи виконує вона свої функції відповідно до специфікацій.

Переваги:

- визначення відповідності вимогам, допомагає визначити, чи відповідає програма вимогам користувачів.
- тестування функціональності, дозволяє випробувати функціональність програми в найширшому розумінні.

Недоліки:

- не виявляє неочевидних помилок, може пропустити неочевидні помилки або аспекти продуктивності.
- обмежений обсягом тестування, фокусується лише на функціональності, не враховуючи інших аспектів, таких як продуктивність чи безпека.

Прийняття (Acceptance Testing). Виконується замовником чи користувачем для перевірки, чи відповідає програма вимогам і чи може бути прийнята в експлуатацію.

Переваги:

- визначення відповідності вимогам користувача, дозволяє забезпечити, що програма повністю відповідає поставленим очікуванням користувачів;
- залучення замовника чи кінцевого користувача, допомагає залучити замовника до процесу тестування та забезпечити йому можливість визначати, чи задовольняє продукт його потреби.

Недоліки:

- обмеженість покриття, зазвичай обмежене за обсягом і не виявляє всіх можливих помилок;
- вартість і час, вимагає значних витрат часу та ресурсів, оскільки виконується після більш ранніх етапів тестування.

Стрес–тестування (Stress Testing). Визначає, як програма веде себе за межами нормальних умов використання, наприклад, при великому навантаженні або в екстремальних умовах.

Переваги:

- виявлення проблем продуктивності, дозволяє виявити, як програма реагує на надмірне навантаження або стресові умови;
- підвищення надійності, допомагає забезпечити, що програма може працювати стабільно під великим тиском.

Недоліки:

- складність налаштування, вимагає обширної підготовки і налаштувань для точного моделювання стресових ситуацій;
- може бути не нерепрезентативним, реальні умови стресу можуть виявитися іншими від тих, які були враховані в тестах.

Тестування безпеки (Security Testing). Оцінює вразливість програми до потенційних загроз безпеці, таких як атаки хакерів чи витоки конфіденційної інформації.

Переваги:

- виявлення потенційних загроз, допомагає виявляти та усувати можливі уразливості та загрози для безпеки;
- захист від атак, дозволяє підвищити рівень безпеки продукту та захистити його від зловмисників.

Недоліки:

- не вичерпне, неможливо передбачити всі можливі атаки та уразливості;

- затримка в розробці, додаткові етапи тестування можуть затримати випуск продукту.

Тестування відновлення (Recovery Testing). Випробовує можливості програми відновлюватися після аварії чи витоку пам'яті.

Переваги:

- виявлення стійкості до відмов, тестує, наскільки ефективно програма може відновитися після виникнення помилок чи відмов;
- підвищення надійності, допомагає покращити стійкість програми та забезпечити швидке відновлення після неполадок.

Недоліки:

- не виявлення всіх можливих проблем, може бути непродуктивним при виявленні всіх можливих сценаріїв відновлення;
- додаткові витрати часу, вимагає додаткового часу для налагодження та виконання тестів відновлення.

Тестування сумісності (Compatibility Testing) [30]. Визначає, як програма працює на різних операційних системах, пристроях, браузерах тощо.

Переваги:

- визначення працездатності на різних платформах, дозволяє визначити, чи працює програма на різних операційних системах та пристроях;
- забезпечення широкого покриття, важливо для продуктів, які призначені для використання на різних платформах.

Недоліки:

- складність забезпечення повного покриття, може бути важко забезпечити повне тестування на всіх можливих платформах та пристроях;
- вартість і час, вимагає додаткових витрат часу та ресурсів для тестування на різних конфігураціях.

Тестування продуктивності (Performance Testing). Включає в себе вимірювання швидкості, обсягу обробки даних та інших характеристик продуктивності програми.

Переваги:

- визначення рівня продуктивності, дозволяє визначити, наскільки швидко та ефективно працює програма;
- виявлення можливих проблем з ресурсами, допомагає виявити проблеми з використанням пам'яті, процесорним часом та іншими ресурсами.

Недоліки:

- складність налаштування, вимагає детального планування та налаштування тестів для точного вимірювання продуктивності;
- може не покрити всі сценарії використання, може бути важко відтворити всі можливі сценарії використання в реальних умовах.

Тестування на реальних користувачів (Beta Testing) [31]. Перевірка продукту на реальних користувачах перед його випуском на ринок.

Переваги:

- залучення реального фідбеку, дозволяє отримати реальний фідбек від реальних користувачів у реальних умовах;
- виявлення проблем, непомічених іншими методами, реальні користувачі можуть виявити аспекти використання, які розробники не передбачали.

Недоліки:

- контроль над умовами тестування, розробники можуть мати обмежений контроль над умовами тестування;
- обмежений обсяг учасників, кількість бета-тестерів обмежена, що може не виявити всіх можливих проблем.

Поєднання різних методів тестування є ключовим для створення комплексної та ефективної стратегії тестування, оскільки різні методики виявляють різні види

помилок, що забезпечує більше повного та комплексного тестування. Етапи тестування можуть взаємодіяти між собою, дозволяючи виявляти та виправляти помилки на різних етапах розробки.

Тому було прийнято рішення обрати Unit Testing та Functional Testing, для тестування роботи системи, оскільки вони надають наступні переваги.

Комплексне покриття.

Unit Testing: дозволяє перевірити кожен окремий модуль або функцію програми і виявити помилки на ранніх етапах розробки.

Functional Testing: перевіряє роботу програми в комплексі та гарантує, що вона відповідає функціональним вимогам.

Ізоляція помилок.

Unit Testing: дозволяє виявляти та виправляти помилки в ізольованих компонентах, що полегшує виправлення і збереження якості коду.

Functional Testing: виявляє помилки в системі як цілому, включаючи взаємодію між компонентами.

Автоматизація та повторюваність.

Unit Testing: легко автоматизується, що дозволяє швидко та повторювано виконувати тести для окремих модулів.

Functional Testing: автоматизація тестів може забезпечити ефективне виконання тестів на різних етапах розробки та випуску.

Підтримка замовника та користувачів:

Unit Testing: використовується для підтримки розробників у процесі розробки і вдосконалення якості коду.

Functional Testing: забезпечує підтримку вимог та очікувань замовника та кінцевих користувачів.

Отже, було рішено використати Unit Testing та Functional Testing, що надає комплексний погляд на якість програмного продукту і дозволяє виявити широкий спектр помилок на різних етапах розробки.

4.2 Тестування роботи нейронної мережі

Для тестування роботи нейронної мережі слід розробити тести, що будуть перевіряти виконання тренування та створення ігрової ситуації, що включає у себе:

Ініціалізація, уявного користувача, рисунок 4.1.

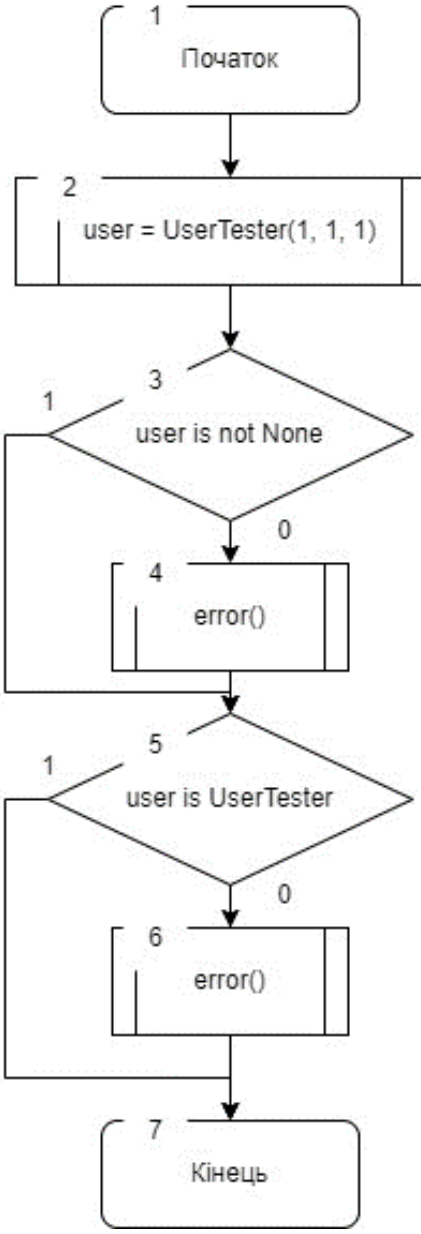


Рисунок 4.1 – Тестування створення уявного користувача

Це імітує створення екземпляра користувача з вказаними параметрами (1, 1, 1). «assertIsNotNone» переконується, що об'єкт користувача був успішно створений

і не має значення «None». «assertIsInstance» перевірка гарантує, що об'єкт є екземпляром вказаного класу (UserTester в цьому випадку).

Цей тест важливий, оскільки перевіряє правильність створення користувача та його класову належність.

Перевірка значень уявного користувача, рисунок 4.2.

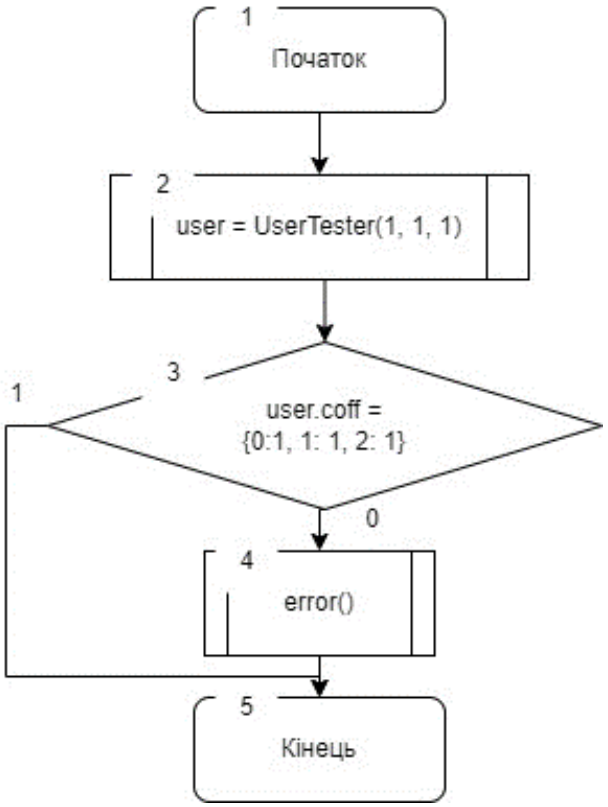


Рисунок 4.2 – Тестування ініціалізації уявного користувача

Створюється користувач, та перевіряється за допомогою «assertDictEqual» відповідність значень уявного користувача зі значеннями у хеш-таблиці.

Це важливий тест для перевірки, чи вірно встановлені значення атрибутів об'єкта користувача під час його ініціалізації. Це особливо важливо, якщо ці значення використовуються в подальших частинах програми або модулю, оскільки будь-які помилки у визначенні початкових значень можуть призвести до невірної роботи системи.

Перевірка визначення метрик майстерності уявного користувача, рисунок 4.3.

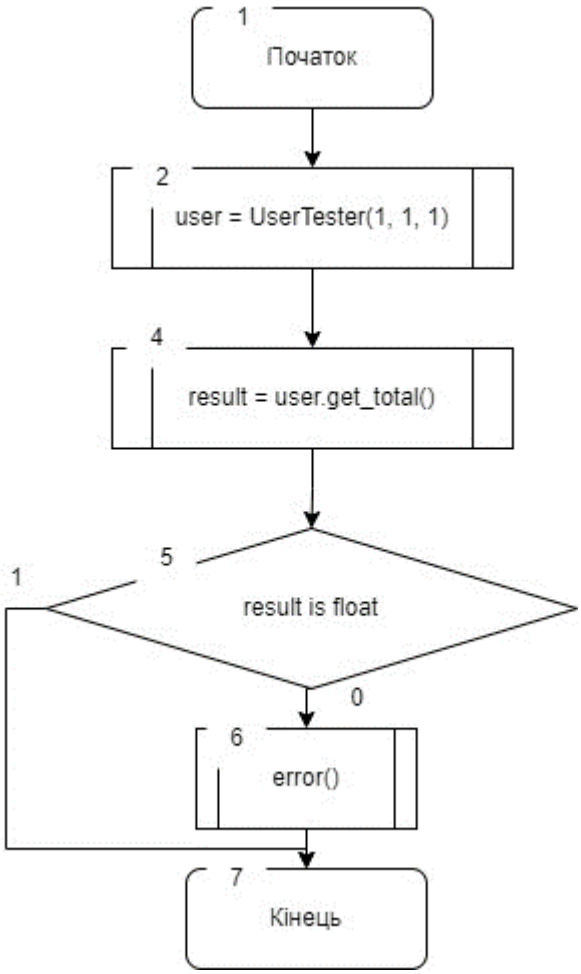


Рисунок 4.3 – Тестування функції визначення метрики

Створюється екземпляр користувача. Готуються тестові дані, у цьому випадку одна ігрова ситуація, без розташованих об’єктів, які ініціалізуються за допомогою «tf.zeros», що створює об’єкт вказаного розміру заповнений нульовими значеннями. Далі обраховується час, що займе проходження рівня. В кінці іде перевірка чи отриманий результат це число з плаваючою комою.

Цей тест важливий для перевірки, чи користувацький об'єкт правильно визначає метрику майстерності та чи повертає вірний тип даних.

Перевірка визначення втрат уявним користувачем, рисунок 4.4.

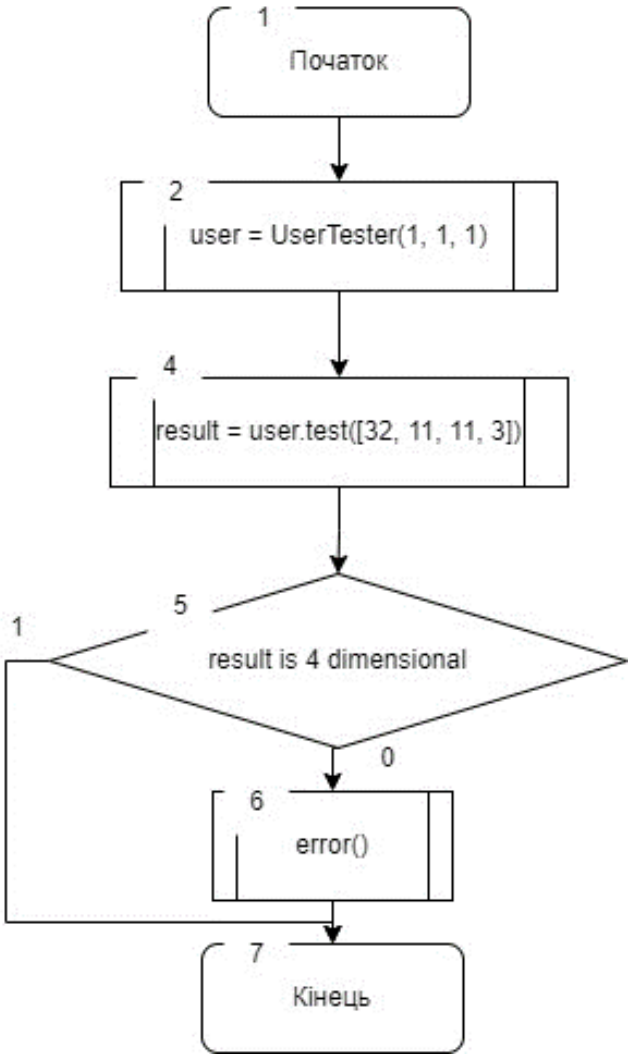


Рисунок 4.4 – Тестування функції визначення втрат

Створюється тестовий користувач. Підготовлюється група тестових ігрових ситуацій, підготовлюється до передачі до тестового користувача за допомогою перетворення у стандартний для пайтону формат даних. Уявний користувач проводить обрахунки, для визначення втрат для нейронної мережі. Перевіряється чи отримані дані мають очікувану структуру.

Цей тест важливий для перевірки, чи користувацький об'єкт правильно визначає втрати та чи повертає вірний тип даних для втрат.

Перевірка функції втрат, що використовується нейронною мережею, рисунок 4.5.

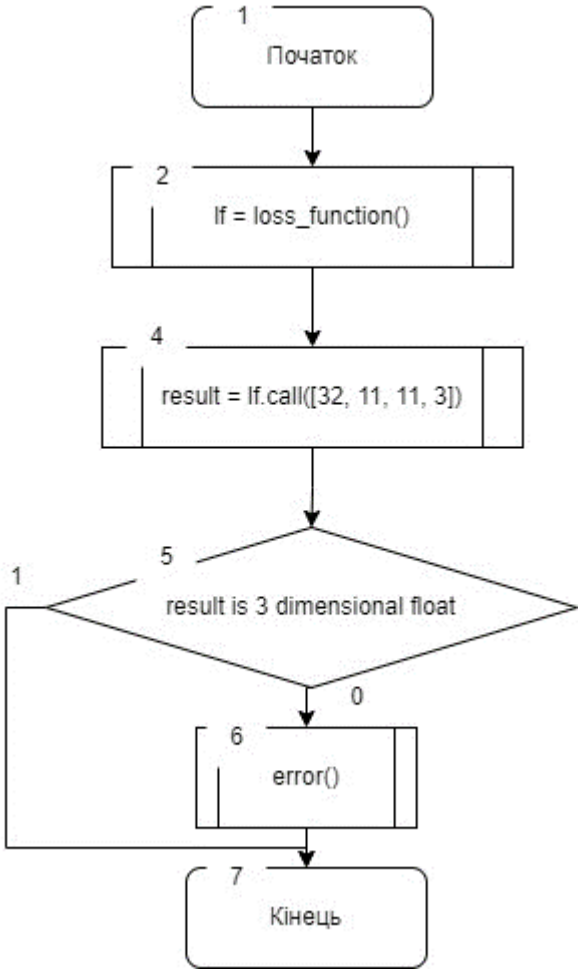


Рисунок 4.5 – Тестування функції втрат НМ

Створюється тестовий користувач. Готуються тестові дані. Проводиться обрахунок втрат у форматі зрозумілому нейронній мережі та перевіряється чи тип відповідає очікуваному. Проводиться перевірка структури даних.

Цей тест важливий для перевірки, чи функція втрат, яка використовується нейронною мережею, повертає вірний тип даних та структуру результату.

Тестування завантаження з файлу нейронної мережі, рисунок 4.6.

Спершу створюється об’єкт, що відповідає за збереження нейронної мережі та роботи з нею, при ініціалізації вказано назву нейронної мережі, що необхідно завантажити у випадку відсутності моделі на диску або помилці при завантажені повернеться значення «None», тому необхідна перевірка, що об’єкт існує.

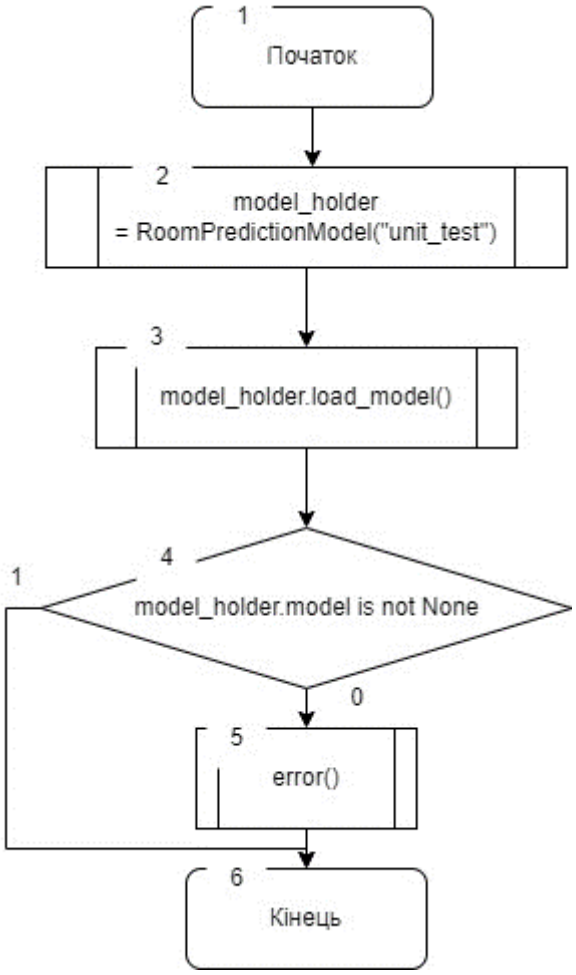


Рисунок 4.6 – Тестування завантаження НМ з диску

Цей тест важливий для переконання, що нейронна мережа успішно завантажується з файлу.

Перевірка створення нейронною мережею ігрової ситуації:

```
def test_create(self):
    room = [1, 16, 11, 11, 3]
    room = tf.zeros(room)
    metric = [1, 16, 1]
    metric = tf.zeros(metric)
    model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="unit_test")
    result = model_holder.predict((room, metric))
```

```
self.assertIsInstance(result, numpy.ndarray)
threshold = tf.cast(0.5, result.dtype)
result = tf.cast(result > threshold, result.dtype)
result = tf.make_tensor_proto(result)
result = tf.make_ndarray(result)
result = result.tolist()
self.assertIsInstance(result, list)
self.assertIsInstance(result[0], list)
self.assertIsInstance(result[0][0], list)
self.assertIsInstance(result[0][0][0], list)
self.assertIsInstance(result[0][0][0][0], float)
```

Що складається з декількох етапів:

- підготовка тестових даних, що включає у себе створення історії проходження ігрових ситуацій, у цьому випадку історія складається з проходження пустих кімнат;
- створення моделі нейронної мережі, створюється модель нейронної мережі з використанням «room_convlstm2d_lstm», що повертає модель, яка складається з двох вимірних шарів згортки та шару пам’ять для збереження певної кількості ігрових ситуацій;
- генерація ігрової ситуації, викликається модель нейронної мережі, що обраховує історію гри та генерує ігрову ситуацію, перевіряється відповідність згенерованого рівня до очікуваного типу даних;
- перевірка відповідності структури кімнати до очікуваної, для цього використовується бінарна класифікація та перетворення до стандартного для пайтона типу даних.

Цей тест важливий для переконання, що нейронна мережа правильно створює ігрову ситуацію та повертає результат в очікуваному форматі.

Тестування тренування нейронної мережі:

```
def test_train(self):
```

```
        room = [1, 16, 11, 11, 3]
        room = tf.zeros(room)
        metric = [1, 16, 1]
        metric = tf.zeros(metric)
        model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="unit_test")
        model_holder.model.compile(
           loss=loss_function(),
           optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
           metrics=[], run_eagerly=True
        )
        result = model_holder.fit(room, metric, room, 1)
        self.assertIsInstance(result.history, dict)
```

Спершу проводиться ініціалізація тестових даних для тренування нейронної мережі. Потім створюється модель для тестування навчання. Проводиться компіляція нейронної мережі з використанням функції втрат, що використовує уявних гравців, вказується функція оптимізації тренування та включення режиму трасування виконання нейронної мережі. Після чого починається тренування на протязі однієї ітерації. Проводиться перевірка на наявність історії тренування.

Цей тест важливий для переконання, що нейронна мережа правильно тренується, і її історія тренування зберігається у форматі словника.

Повни лістинг коду тестів наведено у додатку Ж.

Отже було описано модульні тести для перевірки роботи нейронної мережі, що підтверджують вірність виконання модуль, що відповідають за тренування та створення ігрових ситуацій нейронною мережею.

4.3 Тестування роботи комп’ютерної гри

Для тестування роботи комп’ютерної гри слід розробити тестові сценарії у грі, що можуть бути використані для перевірки окремих об’єктів.

Перевірка на поворот об’єкта гравця за курсором мишки, рисунок 4.7.

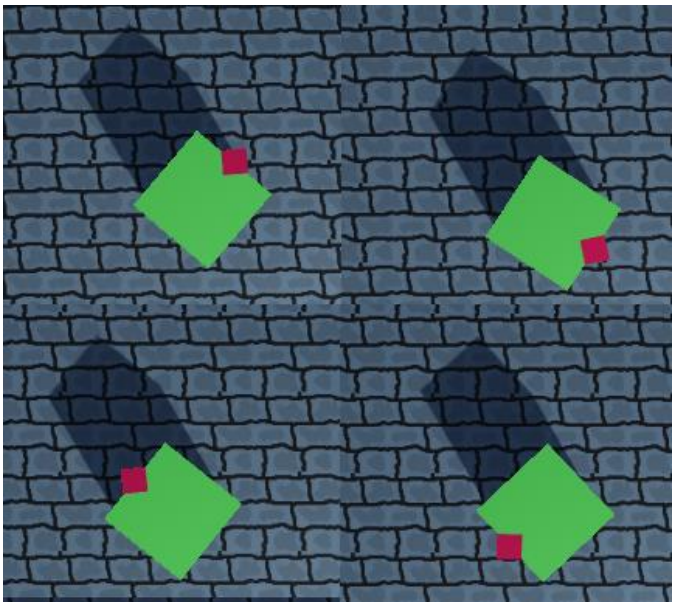


Рисунок 4.7 – Демонстрація повернення гравця за курсором мишки

Перевірка руху гравця по ігровому рівні, рисунок 4.8.

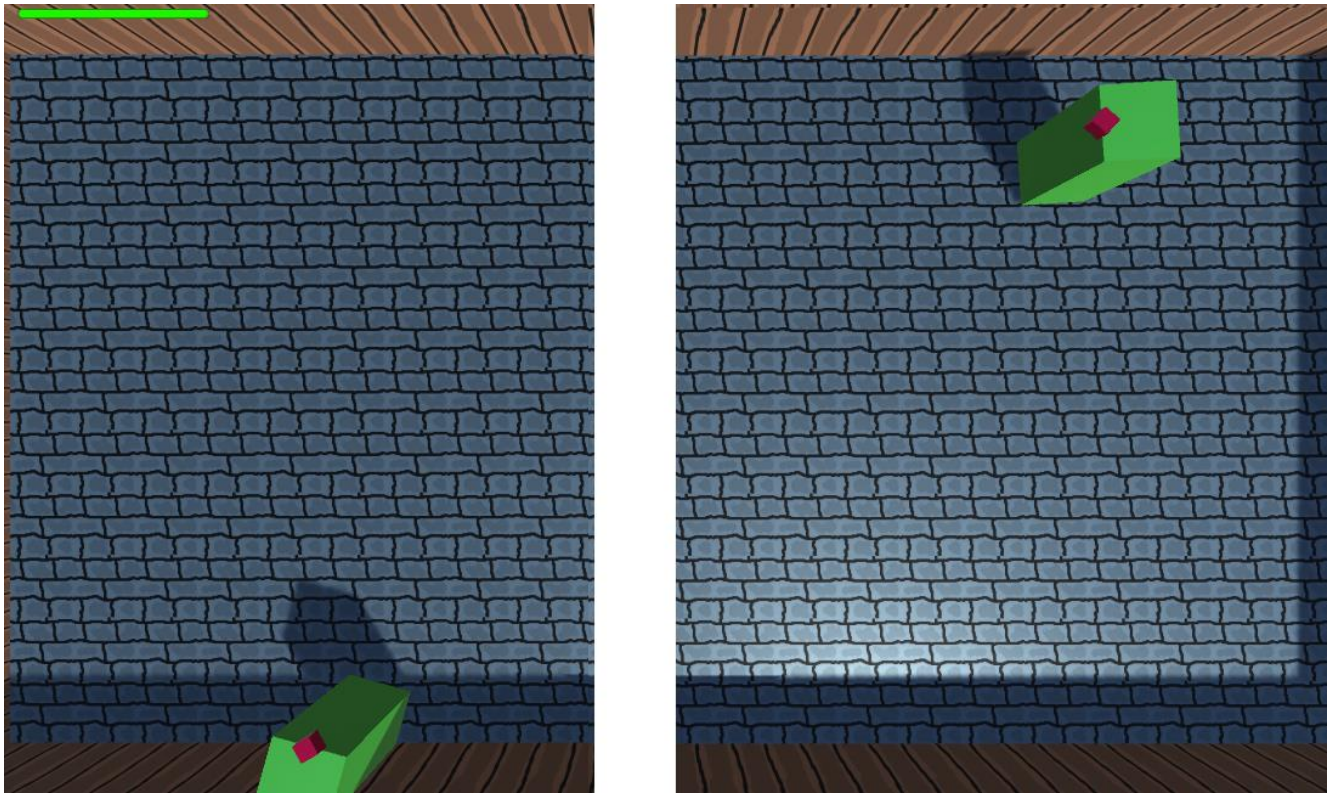


Рисунок 4.8 – Демонстрація руху гравця по ігровому рівні

Перевірка роботи ворога з ближньою атакою, рисунок 4.9.

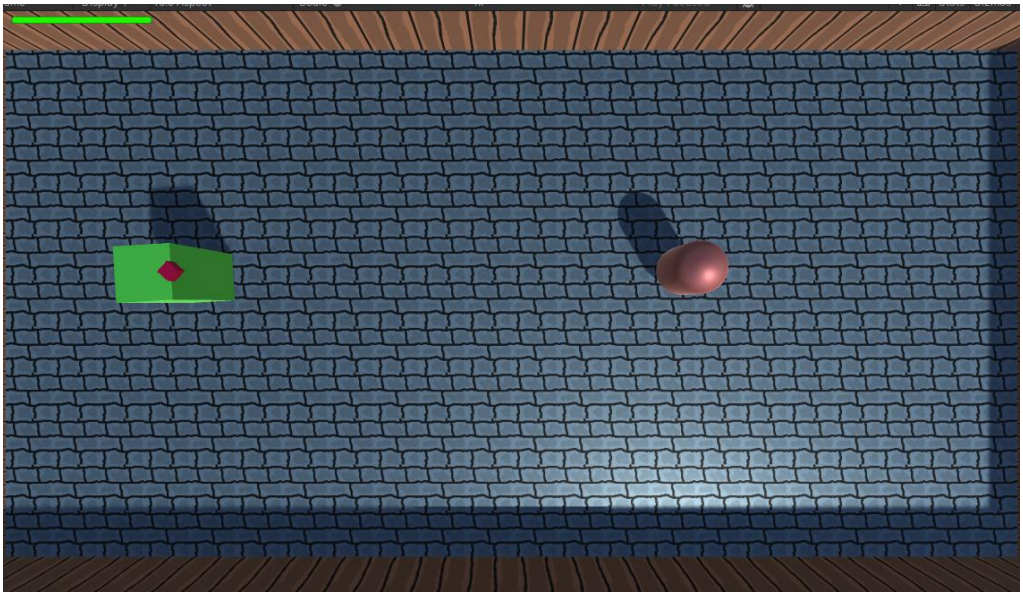


Рисунок 4.9 – Демонстрація об’єкту ворога з ближньою атакою

Перевірка руху ворога з ближньою атакою, перевіряється зближення ворога на дистанцію атаки до гравця, рисунок 4.10.

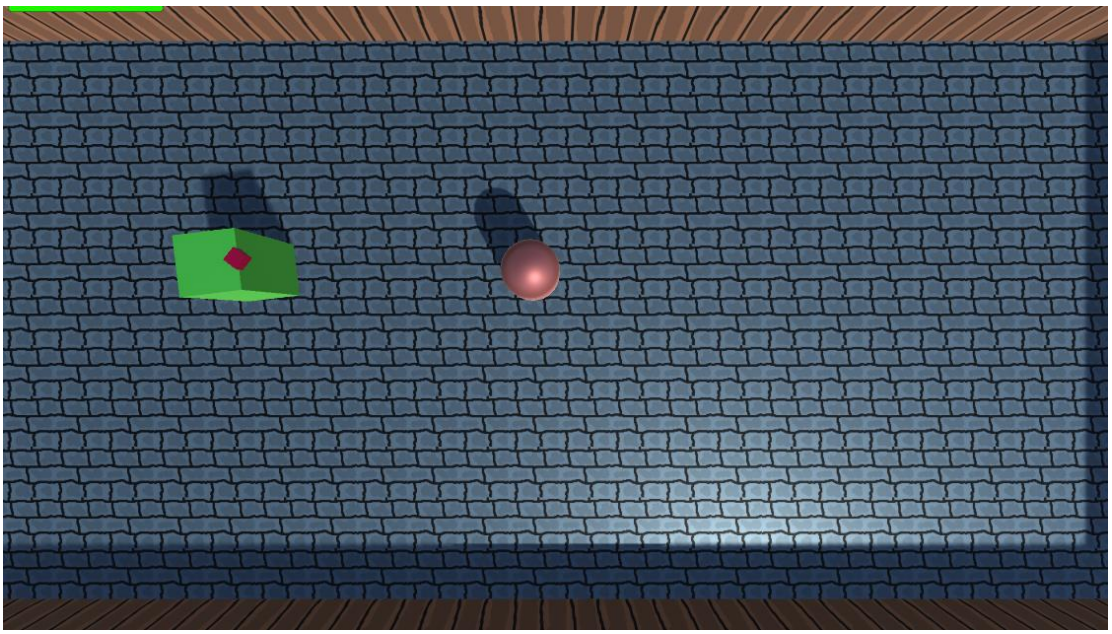


Рисунок 4.10 – Демонстрація зближення до гравця об’єкту ворога

Перевірка атаки ворога з ближньою атакою, що можна замітити по зменшенню здоров’я об’єкту гравця, рисунок 4.11.

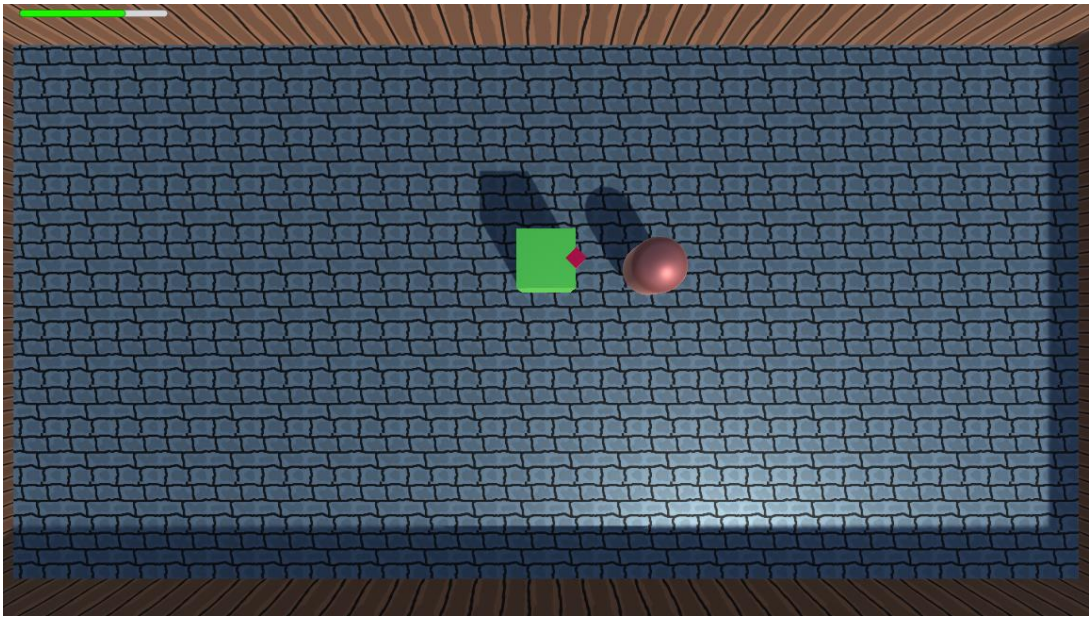


Рисунок 4.11 – Демонстрація атаки по гравцю

Перевірка атаки гравця на ворогу з ближньою атакою, рисунок 4.12.

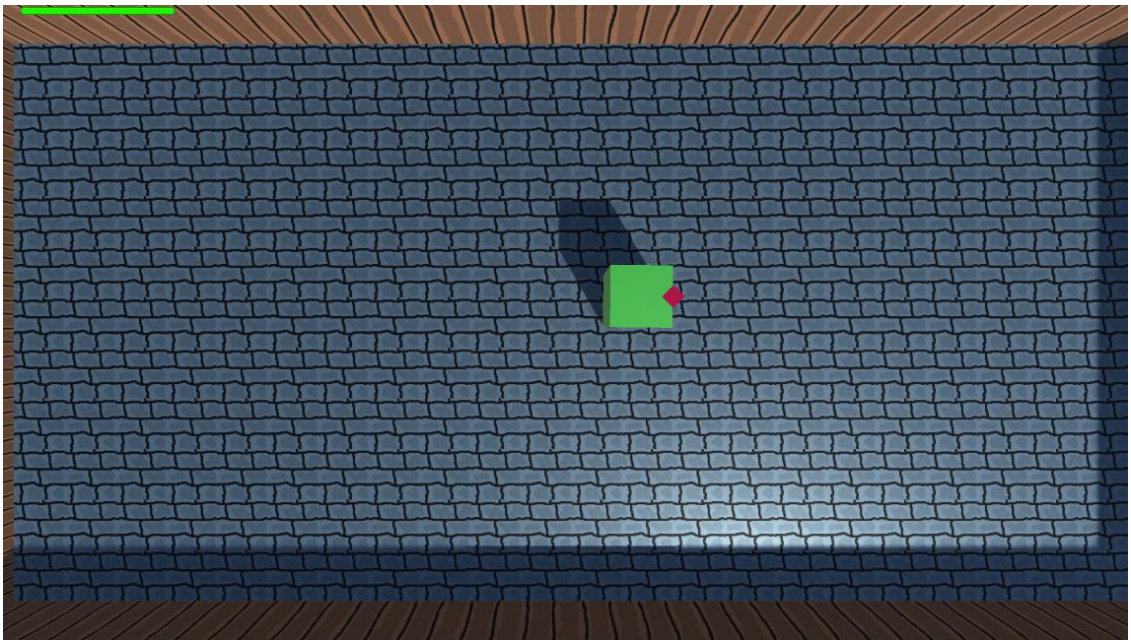


Рисунок 4.12 – Демонстрація знищення переможених об’єктів

Перевірка атаки ворога з дальньою атакою, рисунок 4.13.

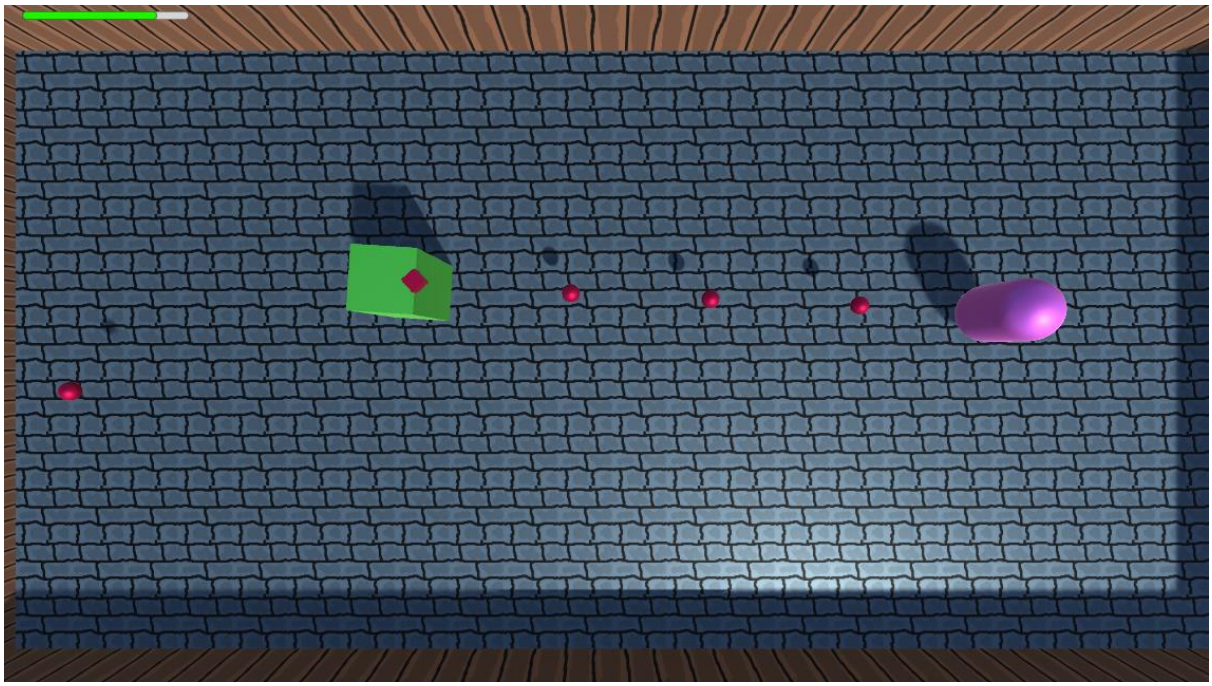


Рисунок 4.13 – Демонстрація атаки ворога дальнього бою

Перевірка роботи модуля генерації ігрової ситуації, рисунок 4.14.

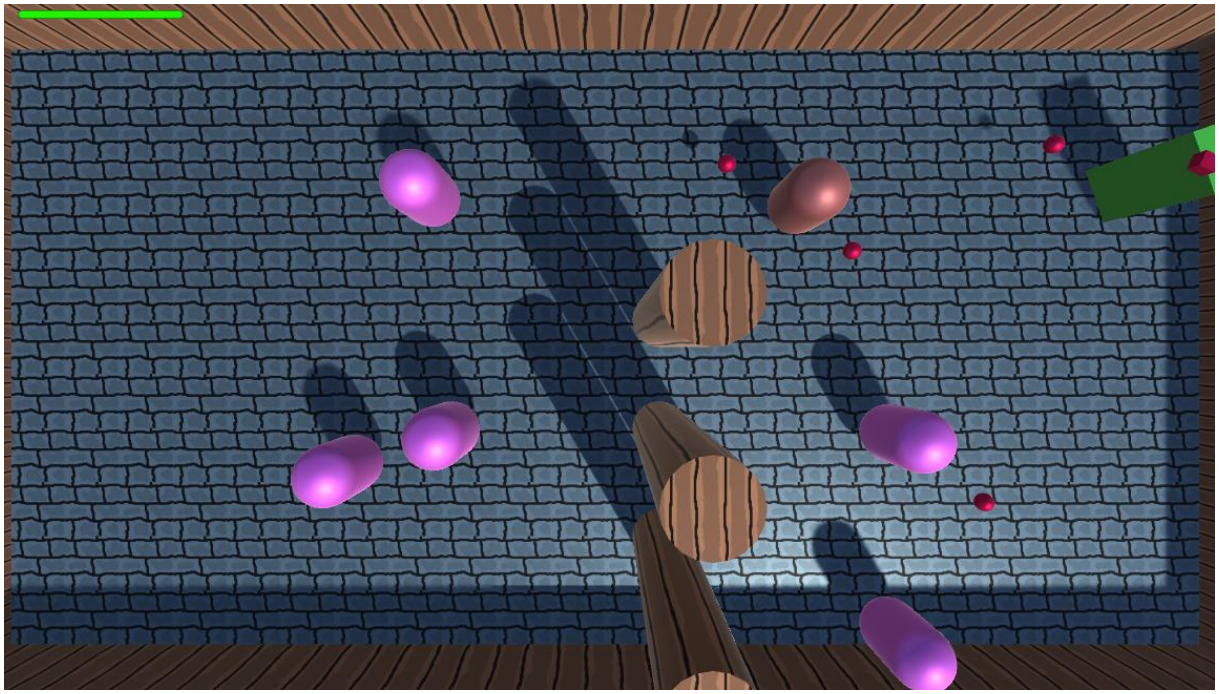


Рисунок 4.14 – Демонстрація згенерованої НМ ігрової ситуації

Отже було проведено тестування комп’ютерної гри, що підтвердило її роботу для демонстрації використання нейронної мережі з метою створення адаптивної складності.

4.4 Тестування роботи системи для генерації ігрових ситуацій

Проведемо тестування системи загалом.

Спершу потрібно запустити сервер з нейронною мережею з використанням «python.exe». Спершу необхідно активувати віртуальне середовище для завантаження бібліотек, що необхідно для виконання серверу. Далі необхідно запустити скрипт «connect.py», що почне процедуру запуску серверу “flask” в локальній мережі на порту 5000, в результаті маємо отримати повідомлення, що сервер запущено, рисунок 4.15.

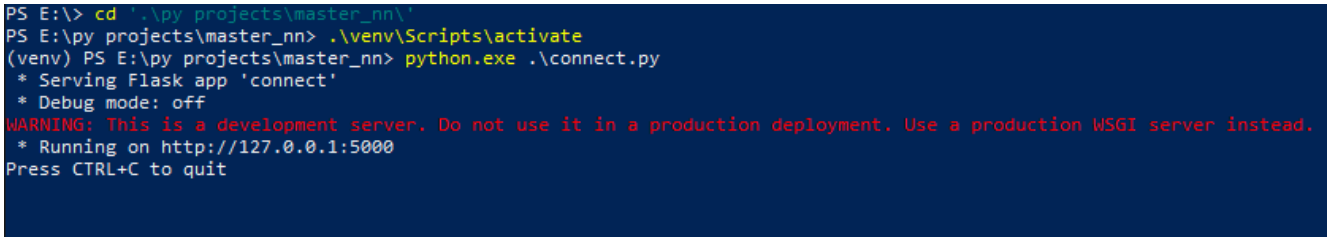


Рисунок 4.15 – Демонстрація роботи серверу

У наступному кроці необхідно відкрити папку з грою, що має вміщувати файли зображенні на рисунку 4.16.

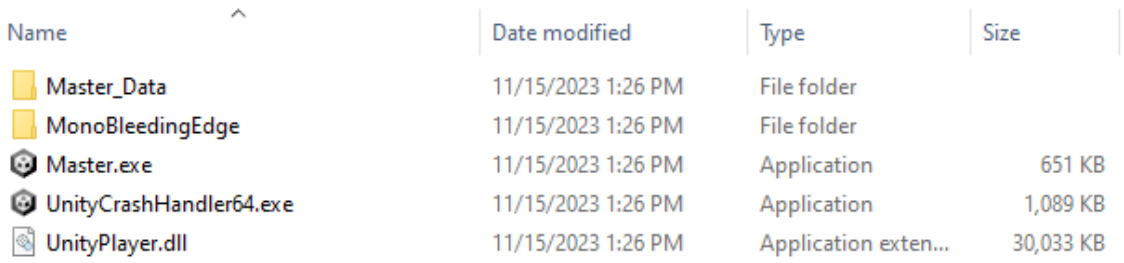


Рисунок 4.16 – Файли комп’ютерної гри

Потім потрібно запустити файл виконуваного формату «master.exe», що запустить гру у повновіконному режимі.

В наступному етапі гра звернеться з запитом до серверу нейронної мережі для генерації ігрової ситуації, рисунок 4.17.

Рисунок 4.17 – Історія запитів оброблених сервером

Після чого гра розташовує об’єкти в залежності від виходів нейронної мережі, рисунок 4.18.

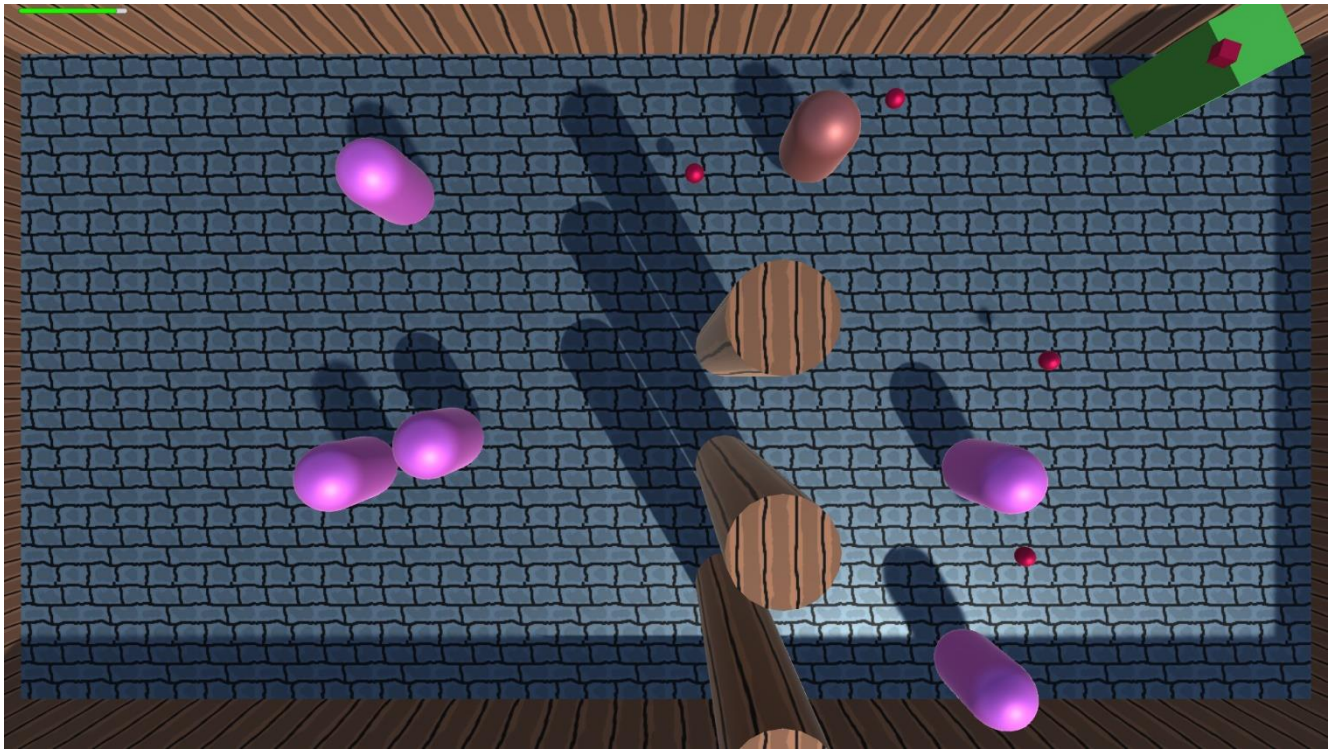


Рисунок 4.18 – Демонстрація роботи системи

Отже було перевірено роботу системи динамічної складності за рахунок використання нейронної мережі, що аналізує майстерність гравця та його історію гри.

4.5 Висновки

У цьому розділі було розглянуто методи тестування програмного забезпечення та було обрано модульне тестування та функціональне тестування для перевірки системи адаптивної складності.

Розроблено модульні тести для перевірки роботи нейронної мережі, що покривають роботу таких систем:

- модуль штучних гравців, а саме перевірка ініціалізації та обрахунку майстерності гравця для ігрової ситуації;
- модуль обчислення втрат нейронної мережі, що включає перевірку обрахунку втрат та відповідності даних;
- модуль навчання нейронної мережі, що перевіряє, що нейронна мережа проходить тренування та збирається;
- модуль завантаження нейронної мережі, включає у себе перевірку завантаження моделі нейронної мережі з файлу використовуючи назву моделі;
- модуль генерації ігрової ситуації, що надає історію гри та майстерності гравця для створення ігрової ситуації.

Проведено тестування комп’ютерної, що включає перевірку взаємодії з користувачем, об’єктами та загальну роботу системи.

Було проведено тестування системи, що підтвердило відповідність роботи програми до поставлених вимог.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка нейронних мереж для генерації ігрових ситуацій» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка нейронних мереж для генерації ігрових ситуацій» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [32].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	4
2. Ринкові переваги (наявність аналогів)	5	4	4
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	5	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	4	3	4

8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	4
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	44	42	43
Середньоарифметична сума балів $СБ_c$	43		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [32].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка нейронних мереж для генерації ігрових ситуацій» становить 43 бали, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка нейронних мереж для генерації ігрових ситуацій», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної

та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [32]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 30000{,}00 \cdot 60 / 21 = 81818{,}18 \text{грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000	1190,48	60	71428,57
Інженер-розробник програмного забезпечення	22000	1047,62	60	62857,14
Всього				134285,71

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [32];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38$ грн.
$З_{р1} = 72{,}38 \cdot 10{,}00 = 723{,}84$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення допоміжного обладнання	10	2	1,1	72,38	723,84
Тренування системи	4	4	1,5	98,71	394,82
Інсталяція програмного забезпечення	7	4	1,5	98,71	690,94
Всього					1809,60

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (134285,71+ 1809,60) · 11 / 100% = 14970,48грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (134285,71+ 1809,60+ 14970,48) · 22 / 100% = 33234,48 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 3 · 210,00 · 1,1 - 0,000 · 0,00 = 693,0 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний Crystal (A4-500)	220	3	0	0	726
Канцелярське приладдя (набір офісного працівника)	175	2	0	0	385
Картридж для принтера HP LaserJet M1132 MFP	1100	1	0	0	1210
Диск оптичний VEKO-10 (CD-R)	15	3	0	0	49,5
Диск оптичний VEKO-W (CD-RW)	20	3	0	0	66
Всього					2436,5

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка нейронних мереж для генерації ігрових ситуацій» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.і}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

$K_і$ – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_і$ = *1,10…1,12);*

k – кількість найменувань устаткування.

$B_{спец}$ = *24999· 1 · 1,1 = 27498,9 грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Обладнання для розробки та тренування нейронних мереж	1	24 999	27498,9
Всього			27498,9

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.і} \cdot K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$K_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_і$ = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг}$ = *11600,00 · 2 · 1,1 = 25984 грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	2	11600	25984
Прикладний пакет Microsoft Office 2019	2	5500	12320
Всього			38304

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (35000{,}00 \cdot 3) / (5 \cdot 12) = 1750грн.$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп’ютер	30 000	5	3	1500,00
Персональний комп’ютер	35 000	5	3	1750,00

Робоче місце дослідника	20000	5	3	1000,00
Обладнання для розробки та тренування нейронних мереж	24999	4	3	1562,44
Оргтехніка	7000	4	3	437,50
Приміщення лабораторії	250000	20	3	3125,00
ОС Windows 11	25984	3	3	2165,33
Прикладний пакет Microsoft Office 2019	12320	3	3	1026,67
				12566,94

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,3 \cdot 480,0 \cdot 7,50 \cdot 0,95 / 0,97 = 1057,73$грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп’ютер	0,3	480	1057,73
Персональний комп’ютер	0,3	480	1057,73
Обладнання для розробки та тренування нейронних мереж	0,28	480	987,22
Робоче місце дослідника	0,15	480	528,87
Оргтехніка	0,45	10	33,05
Всього			3664,60

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (5.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св} = (134285,71 + 1809,60) \cdot 20 / 100\% = 27219,06грн.$

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$B_{сп}$ = (134285,71+ 1809,60) · 35 / 100% = 47633,36грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (134285,71+ 1809,60) · 50 / 100% = 68047,66грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$B_{нзв}$ = (134285,71+ 1809,60) · 100 / 100% = 136 095,31грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$B_{заг}$=134285,71+1809,60+14970,48+33234,47+2436,5+0,00+27498,9+38304+ 12566,94+ 3664,60+27219,06+47633,36+68047,66+136 095,31= 547766,60грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 1148934,22/ 0,9 = 608629,56грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 100 користувачів/рік;

2-й рік – 200 користувачів/рік;

3-й рік – 300 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 850 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 35000 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 5000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [33]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (850 \cdot 5000,00 + 40000 \cdot 100) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) = 1690573,5$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (850 \cdot 5000,00 + 40000 \cdot (100+200)) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) = 3329917,5$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (850 \cdot 5000,00 + 40000 \cdot (100+200+300)) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) = 5788933,5$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП =$ 1690573,5 /(1+0,25)^1+ 3329917,5 /(1+0,25)^2+ 5788933,5 /(1+0,25)^3=

6447539,95 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 608629,56 грн.

$PV = k_{інв} \cdot ЗВ =$ 2· 608629,56 = 1217259,113 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 6447539,95грн;

PV – теперішня вартість початкових інвестицій, 1217259,113 грн.

$E_{абс} = ПП - PV$ = 6447539,95 - 1217259,113 = 5230280,84грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 5230280,84грн;

PV – теперішня вартість початкових інвестицій, 1217259,113 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+5230280,84/\ 1217259,113)^{1/3} = 0,74.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{5.22}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,11+0,25 = 0,36 < 0,74 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка нейронних мереж для генерації ігрових ситуацій» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.23}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,74 = 1,35 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка нейронних мереж для генерації ігрових ситуацій» становить 43 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 1,35 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати

потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка нейронних мереж для генерації ігрових ситуацій».

ВИСНОВКИ

У магістерській кваліфікаційній роботі було проведено аналіз предметної галузі використання нейронних мереж у геймдизайні та ігровій розробці.

Проаналізовано методи відображення результатів застосування нейронних мереж, а також методи їх використання для генерації ігрових ситуацій. Визначено важливість вибору фреймворку для реалізації нейронної мережі для розробки адаптивної системи зміни складності гри з використанням нейронної мережі.

Розглянуто процес розробки моделі нейронної мережі для аналізу та генерації ігрових ситуацій, а також визначено функції втрат та методи навчання мережі.

У третьому розділі була розроблена комп'ютерної гри для демонстрації результатів роботи нейронної мережі. Обрано рушій для розробки, розроблені об'єкти для генерації ігрових ситуацій та штучний інтелект для рухомих об'єктів. Проведено розробку інтерфейсу користувача та системи керування, а також HTTP серверу для взаємодії з нейронною мережею.

У розділі четвертому були використані методи тестування для оцінки роботи розробленої системи. Проведено тестування нейронної мережі, комп'ютерної гри та системи для генерації ігрових ситуацій. Отримані результати підтвердили ефективність та стабільність розробленої системи.

Отриманні в магістерській роботі наукові та практичні результати можна використати для розробки систем адаптивної складності в іграх, на основі аналізу майстерності гравця.

Загальні висновки дають змогу стверджувати, що використання нейронних мереж у геймдизайні надає можливість збільшити рівень інтелектуалізації гри, покращити точність аналізу майстерності гравця та може значно покращити якість ігрового досвіду, забезпечуючи адаптивність та індивідуалізацію гри для кожного користувача. Розроблені методи аналізу та генерації ігрових ситуацій відкривають нові можливості у створенні захопливих та унікальних ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Доценко Д. В., Романюк О. Н., Шевчук Р. П. Використання нейронних мереж для реалізації адаптивної складності на основі аналізу майстерності гравця у комп'ютерних іграх. Матеріали XVI міжнародної науково-практичної конференції «Інформаційні технології і автоматизація - 2023», Одеса, 19-20 жовтня 2023 р. Одеса, Видавництво ОНТУ, 2023 р. С. 405-406.
2. Доценко Д.В., Романюк О.Н., Котлик С.В., Чехместрук Р.Ю., Майданюк В.П. Використання нейронних мереж для аналізу складності ігрових ситуацій у комп'ютерних іграх. Матеріали міжнародної науково-практичної конференції «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 20-21 листопада 2023 р. Суми, Видавництво НІКО, 2023 р. С. 98-100.
3. The Art of Game Design: A Book of Lenses 3rd Edition, Джессі Шелл, 2019 р. 652 – с.
4. Rules of Play: Game Design Fundamentals, Кейті Сален Текінбас, 2020 р. 688 – с.
5. Game Design Workshop: A Playcentric Approach to Creating Innovative Games, Fourth Edition, Трейсі Фулертон, 2018 р. 522 – с.
6. Deep Learning (Adaptive Computation and Machine Learning series) - Ян Гудфеллоу, Йошуа Бенжіо, Аарон Курвіль, 2018 р. 789 – с.
7. LANGUAGE AI: A Guide for Humans, Ренді Спарксмен, 2023 р. 153 – с.
8. Learning TensorFlow: A Guide to Building Deep Learning Systems 1st Edition, Том Хоуп, 2018 р. 242 – с.
9. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Ауреліан Герон, 2019 р. 856 – с.
10. Neural Networks and Deep Learning: A Textbook 1st ed., Чару Аггервель, 2018 р. 520 – с.

11. Artificial Intelligence: A Guide to Intelligent Systems (3rd Edition), Міхаель Негентевський, 2021 р. 504 – с.
12. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition, Себастіан Рашка, Вахід Міржалілі, 2019 р. 772 – с.
13. Game Development Essentials: An Introduction 3rd Edition, Дженні Новак, 2021 р. 510 – с.
14. Designing Games for Children: Developmental, Usability, and Design Considerations for Making Games for Kids, Карла Фішер, 2022 р. 276 – с.
15. Designing Games for Children: Developmental, Usability, and Design Considerations for Making Games for Kids, Карла Фішер, 2021 р. 276 – с.
16. Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras, Ражалінгаппа Шанмугамані, 2018 р. 310 – с.
17. Dive into Deep Learning 1st Edition, Астон Жанг, Захарій Ліптон, Му Лі, Александр Смола, 2023 р. 574 – с.
18. Image Classification Based On CNN: A Survey, Омар Махмуд, Ахмед Елнгар, Амар Фаті, 2020 р. 50 – с.
19. Deep Learning with PyTorch: Build, train, and tune neural networks using Python tools First Edition, Елі Стівенс, Люка Антіга, Томас Вєхманн, 2020 р. 520 – с.
20. Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning) second edition, Річард Суттон, Андрій Барто, 2018 р. 322 – с.
21. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play, Давид Фостер, 2019 р. 327 – с.
22. Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow, 2nd Edition, Іван Васільов, Даніель Слатер, Джіанмаріо Спакагна, Пітер Роелантс, 2019 р. 388 – с.
23. Adam: A Method for Stochastic Optimization, Діедрік Кінгма, 2019 р. 15 – с.
24. Designing Games for Children: Developmental, Usability, and Design Considerations for Making Games for Kids, Карла Фішер, 2021 р. 276 – с.

25. Unity in Action: Multiplatform Game Development in C# with Unity 5 3rd Edition, Джо Хокінг, 2022 р. 416 – с.
26. Unity Multiplayer Games, Алан Стагнер, 2021 р. 242 – с.
27. Mastering Unity 2D Game Development, Сімон Джексон, 2021 р. 474 – с.
28. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Джез Хамбл, Давид Фарлей, 2018 р. 512 – с.
29. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations, Ніколь Форсгрен, Джез Хамбл, Джене Кім, 2018 р. 288 – с.
30. Python Testing with pytest: Simple, Rapid, Effective, and Scalable 2nd Edition, Браян Оккен, 2022 р. 274 – с.
31. Experiences of Test Automation: Case Studies of Software Test Automation 1st Edition, Дороті Грахам, Марк Грахам, 2019 р. 672 – с.
32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.
33. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

«19» вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка нейронної мережі для створення ігрових ситуації»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

д.т.н., проф. О.Н. Романюк

"19" вересня 2023 р.

Виконав:

студент гр.3ПІ-22м Д.В. Доценко

"19" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка нейронної мережі для створення ігрових ситуації».

Галузь застосування – комп'ютерні ігри.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від 18.09.23 ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Мета роботи полягає в розширенні функціональних можливостей гри за рахунок збільшення ігрових ситуацій та покращення динамічної складності.

Призначення роботи – розробка методів і засобів використання нейронних мереж для створення ігрових ситуацій.

3 Вихідні дані для проведення МКР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. LANGUAGE AI: A Guide for Humans, Ренді Спарксмен, 2023. 153 – с.
2. Learning TensorFlow: A Guide to Building Deep Learning Systems 1st Edition, Том Хоуп, 2017, 242 – с.
3. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Ауреліан Герон, 2019, 856 – с.
4. Neural Networks and Deep Learning, Чару Аггервель, 2018, 520 – с.
5. Artificial Intelligence, Міхаель Негентевський, 2021, 504 – с.

4. Технічні вимоги

Вихідні дані до роботи: кольоровий режим –True Color; максимальний розмір дискретного координатного простору – 4096*4096; тривимірні матриці про

досягнення гравцем рівнів; інформація про майстерність користувача при проходженні рівнів; час затрачений на проходження кожної ігрової ситуації; вихідні дані – ігрова ситуація, що була згенерована з використанням нейронної мережі.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Обґрунтування розробки	20.09.2023 – 25.09.2023
2	Розробка методу адаптивної зміни складності гри за рахунок використання нейронної мережі	26.09.2023 – 5.10.2023

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
3	Розробка комп’ютерної гри для демонстрації результатів нейронної мережі	6.10.2023 – 15.10.2023
4	Тестування роботи системи	16.10.2023 – 25.10.2023
5	Оформлення матеріалів до захисту МКР	26.10.2023 – 10.11.2023
6	Економічна частина	10.11.2023 - 1.12.2021

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: «Розробка нейронної мережі для створення ігрових ситуацій»

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ЗПІ – 22м

Науковий керівник:

Unicheck	
Оригінальність	**95.3**
Схожість	4.7

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Доценко Д. В.

Керівник роботи _____ Романюк О. Н.

Додаток В. Лістинг нейронної мережі

Додаток В.1 Лістинг моделі нейронної мережі

```
import os
import gc

import tensorflow as tf

from keras import backend
from keras.regularizers import l2



class ClearMemory(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        gc.collect()
        backend.clear_session()



class RoomPredictionModel:
    def __init__(self, name: str, model=None):
        self.cp_callback = None
        self.model = model
        self.name = name
        self.checkpoint_path = os.path.join(
            os.path.dirname(os.path.abspath(__file__)),
            "save_network",
            self.name
        )
```

```
def load_model(self):
    os.makedirs(os.path.dirname(self.checkpoint_path), exist_ok=True)
    if os.path.exists(self.checkpoint_path):
        self.model = tf.keras.models.load_model(self.checkpoint_path)
    else:
        print("Err: checkpoint file not found")
    return self.model


def fit(self,
        x_room, x_metric, y,
        epochs: int,
        ):
    os.makedirs(os.path.dirname(self.checkpoint_path), exist_ok=True)
    self.cp_callback = tf.keras.callbacks.ModelCheckpoint(
        filepath=self.checkpoint_path,
        verbose=1,
        save_best_only=True,
        save_freq='epoch')
    return self.model.fit(
        x=(x_room, x_metric),
        y=y,
        epochs=epochs,
        shuffle=False,
        callbacks=[self.cp_callback, ClearMemory()]
    )
```

```
    def predict(self, input_data):
        return self.model.predict(input_data)


def room_convlstm2d_lstm(x=11, y=11, features=3, timesteps=32):

    input_room = tf.keras.layers.Input(shape=(timesteps, x, y, features))
    layer_room = tf.keras.layers.ConvLSTM2D(filters=32, kernel_size=3,
return_sequences=False)(input_room)
    layer_room = tf.keras.layers.Activation('LeakyReLU')(layer_room)
    layer_room = tf.keras.layers.Flatten()(layer_room)
    model_room = tf.keras.models.Model(inputs=input_room, outputs=layer_room)

    input_metric = tf.keras.layers.Input(shape=(timesteps, 1))
    layer_metric = tf.keras.layers.LSTM(32, return_sequences=False)(input_metric)
    layer_metric = tf.keras.layers.Activation('LeakyReLU')(layer_metric)

    combined = tf.keras.layers.concatenate([layer_room, layer_metric])

    output = tf.keras.layers.Dense(x * y * features, activation='sigmoid')(combined)
    output = tf.keras.layers.Reshape((x, y, features))(output)

    model = tf.keras.models.Model(inputs=[input_room, input_metric], outputs=output)
    return model
```

Додаток В.2 Лістинг функції втрат нейронної мережі та штучних гравців

```
import math
```

```
import os
import gc
import random
import tensorflow as tf


from keras import backend
from keras.src.engine import data_adapter
from keras.regularizers import l2
from itertools import repeat



class UserTester:
    def __init__(self,
                 wall_coff=random.randint(1, 4) / 10,
                 melee_coff=random.randint(1, 20) / 5,
                 range_coff=random.randint(1, 20) / 5
                 ):
        self.coff = {
            0: wall_coff,
            1: melee_coff,
            2: range_coff,
        }

    @staticmethod
    def check(it, num):
        it = iter(it)
        return all(any(it) for _ in range(num))
```

```
def get_total(self, batch):


    threshold = tf.cast(0.5, batch.dtype)
    batch = tf.cast(batch > threshold, batch.dtype)
    batch = tf.make_tensor_proto(batch)
    batch = tf.make_ndarray(batch)
    batch = batch.tolist()
    batch = batch[0]
    total = 0
    for y in batch:
        for x in y:
            if all(map(any, repeat(iter(x), 2))):
                continue
            for i, obj in enumerate(x):
                if obj:
                    total += self.coff[i]
    total += random.randint(1, 40) / 10
    return total


def test(self, inputs):
    return_data = []
    for batch in inputs:
        return_room = []
        total = 0
        for y in batch:
            rows = []
```

```
    for x in y:
        row = [0, 0, 0]
        if all(map(any, repeat(iter(x), 2))):
            rows.append(row)
            continue
        for i, obj in enumerate(x):
            if obj:
                total += self.coff[i]
        rows.append(row)
    return_room.append(rows)
total += random.randint(1, 40) / 10
total = (total - 20) / 40
zero = 1 - total if total < 0 else total
one = 1 - total if total > 0 else total
max_v = max([one, zero])
zero = zero / max_v
one = one / max_v

y_ = len(batch)
x_ = len(y)
for y in range(y_):
    for x in range(x_):
        if all(map(any, repeat(iter(batch[y][x]), 2))):
            return_room[y][x] = [0 if value else zero for value in batch[y][x]]
        elif any(batch[y][x]):
            if one > 0.5:
                for i, obj in enumerate(batch[y][x]):
```

```
                if obj:
                    return_room[y][x][i] = one
                else:
                    return_room[y][x][i] = zero
            else:
                if not random.randint(0, 2):
                    for i, obj in enumerate(batch[y][x]):
                        if obj:
                            return_room[y][x][i] = zero
                        else:
                            return_room[y][x][i] = one
                else:
                    for i, obj in enumerate(batch[y][x]):
                        if obj:
                            return_room[y][x][i] = one
                        else:
                            return_room[y][x][i] = zero
        else:
            if zero > 0.5:
                if not random.randint(0, 2):
                    rand_rotate = random.randint(0, 2)
                    return_room[y][x] = [zero, one, one, zero, one][rand_rotate:rand_rotate + 3]
                else:
                    return_room[y][x] = [one, one, one]
            else:
                return_room[y][x] = [zero for i, obj in enumerate(batch[y][x])]
    return_data.append(return_room)
```

```
        return return_data



class loss_function(tf.keras.losses.Loss):
    def __init__(self):
        super().__init__()
        self.user = UserTester(random.randint(1, 4) / 10, random.randint(1, 20) / 5, random.randint(1, 20)
/ 5)

    def call(self, y_true, y_pred):
        y_pred = tf.convert_to_tensor(y_pred)

        threshold = tf.cast(0.5, y_pred.dtype)
        save = tf.cast(y_pred > threshold, y_pred.dtype)
        save = tf.make_tensor_proto(save)
        save = tf.make_ndarray(save)
        save = save.tolist()
        save = self.user.test(save)

        y_true = tf.convert_to_tensor(save)
        # y_pred = tf.cast(y_pred, save.dtype)
        return backend.mean(tf.math.squared_difference(y_pred, y_true), axis=-1)
```

Додаток В.3 Лістинг коду для тренування нейронної мережі

```
class RoomModel(tf.keras.Model):
    def warmup(self, inputs):
        x, *state = self.lstm_rnn(inputs)
        prediction = self.dense(x)
        return prediction, state

    def train_step(self, data):
        x, y, sample_weight = data_adapter.unpack_x_y_sample_weight(data)
        # Run forward pass.
        user = UserTester()
        with tf.GradientTape() as tape:
            save_pred = []
            save_params = []
            for i in range(16):
                if len(save_pred):
                    tmp = x[0][:, i:]
                    for j in save_pred:
                        tmp = tf.concat([tmp, [j]], axis=1)
                    tmp_2 = x[1][:, i:]
                    for j in save_params:
                        j = tf.convert_to_tensor([[j]])
                        tmp_2 = tf.concat([tmp_2, [j]], axis=1)
                    y_pred = self((tmp, tmp_2), training=False)
                    save_pred.append(y_pred)
                    save_params.append(tf.convert_to_tensor(user.get_total(y_pred)))
                else:
                    y_pred = self(x, training=False)
```

```
                save_pred.append(y_pred)
                save_params.append(tf.convert_to_tensor(user.get_total(y_pred)))

        for i in range(16):
            with tf.GradientTape() as tape:
                _tmp = tmp[:, i:]
                _tmp_2 = tmp_2[:, i:]
                y_pred = self((_tmp, _tmp_2), training=True)
                self.loss.user = user
                loss = self.compute_loss((_tmp, _tmp_2), y, y_pred, sample_weight)
                tmp = tf.concat([tmp, [y_pred]], axis=1)
                tmp_2 = tf.concat([tmp_2, [tf.convert_to_tensor([[user.get_total(y_pred)]])]], axis=1)
            self._validate_target_and_loss(y, loss)
            self.optimizer.minimize(loss, self.trainable_variables, tape=tape)
        return self.compute_metrics(x, y, y_pred, sample_weight)

def train(input_data, out_data):

    model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="nn_1_1")
    model_holder.load_model()
    model_holder.model.compile(
        loss=loss_function(),
        optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
        metrics=[], run_eagerly=True
    )
```

Додаток Г. Лістинг гри для показу ігрової ситуації

Додаток Г.1 Лістинг модуля контролю руху ворога

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemy_melee_controller : MonoBehaviour
{
    private CharacterController controller;
    private Player player;
    private Vector3 enemyVelocity;
    private bool grounded;
    private float enemySpeed = 2.0f;
    private float gravityValue = -9.81f;
    // Start is called before the first frame update
    void Start()
    {
        controller = gameObject.AddComponent<CharacterController>();
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        if (player == null){
            Debug.Log("error");
        }

    }

    // Update is called once per frame
    void Update()
```

```
    {
        grounded = controller.isGrounded;
        if (grounded && enemyVelocity.y < 0)
        {
            enemyVelocity.y = 0f;
        }

        Vector3 move = (player.transform.position - transform.position).normalized;
        float distance = Vector3.Distance (player.transform.position, transform.position);
        if (distance > gameObject.GetComponent<Attack>().range)
        {
            controller.Move(move * Time.deltaTime * enemySpeed);
        }
        if (move != Vector3.zero)
        {
            gameObject.transform.forward = move;
        }

        enemyVelocity.y += gravityValue * Time.deltaTime;
        controller.Move(enemyVelocity * Time.deltaTime);

    }
}
```

Додаток Г.2 Лістинг модуля атаки

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class Attack : MonoBehaviour
{
    public float damage = 1.0f;
    public string target = "Player";
    public float range = 2.0f;
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }
}

public class MeleeAttack : Attack
{
    // Start is called before the first frame update
    void Start()
    {
        this.enabled = false;
    }

    // Update is called once per frame
    void Update()
```

```
    {
        foreach(Collider collider in Physics.OverlapSphere(transform.position + transform.forward, range / 2)){
            if (collider.gameObject.tag == target && collider.GetType() == typeof(CharacterController))
            {
                if(target == "Player"){
                    collider.gameObject.GetComponent<Player>().health -= damage;
                } else {
                    collider.gameObject.GetComponent<Enemy>().health -= damage;
                }
            }
        }
        this.enabled = false;
    }
}
public class RangeAttack : Attack
{
    public GameObject attack;
    // Start is called before the first frame update
    void Start()
    {
        this.enabled = false;
    }

    // Update is called once per frame
    void Update()
    {
```

```
        GameObject a = Instantiate(attack, transform.position + transform.forward, transform.rotation);
        a.SetActive(true);
        this.enabled = false;
    }
}

public class collison : MonoBehaviour
{
    public float damage = 1.0f;
    public float speed = 4.0f;
    public string target = "Player";
    // Start is called before the first frame update
    void Start()
    {

    }
    void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == target){
            if(target == "Player"){
                collision.gameObject.GetComponent<Player>().health -= damage;
            } else {
                collision.gameObject.GetComponent<Enemy>().health -= damage;
            }
        }
        Destroy(gameObject);
    }
```

```
    // Update is called once per frame
    void Update()
    {
        gameObject.GetComponent<Rigidbody>().velocity = transform.forward * speed;
    }
}
```

Додаток Г.3 Лістинг модуля контролю ворога

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Enemy : MonoBehaviour
{
    public float damage = 1.0f;
    public float health = 10.0f;
    public float distance = 10.0f;
    public int cooldown = 0;
    private Player player;
    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

    }

    // Update is called once per frame
```

```
    void Update()
    {
        if(health <= 0){
            gameObject.SetActive(false);
        }
        distance = Vector3.Distance (player.transform.position, transform.position);
        if (distance < gameObject.GetComponent<Attack>().range * 2 && cooldown == 0)
        {
            cooldown = 720;
            gameObject.GetComponent<Attack>().enabled = true;
        }
        if (cooldown > 0){
            cooldown -= 1;
        }
    }
}
```

Додаток Г.4 Лістинг модуля завантаження ігрової ситуації

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class GenerateEncounter : MonoBehaviour
{
    public int[,] objects;
    public GameObject obj1;
```

```
public GameObject obj2;
public GameObject obj3;
public List<GameObject> obj;
public bool finnished = true;
public List<GameObject> created;
private Player player;
public float timespent = 0.0f;
// Start is called before the first frame update
void Start()
{
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
    obj.Add(obj1);
    obj.Add(obj2);
    obj.Add(obj3);
    objects = new int[11, 11, 3];
    created = new List<GameObject>();
}

IEnumerator LoadEncouter(){

    List<IMultipartFormSection> formData = new List<IMultipartFormSection>();
    formData.Add(new MultipartFormDataSection("time=" + timespent.ToString("R")));
    UnityWebRequest www = UnityWebRequest.Post("localhost:25000", formData);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success) {
        Debug.Log(www.error);
```

```
    }
    else {
        Debug.Log(www.downloadHandler.text);
        byte[] results = www.downloadHandler.data;
        int x = 11;
        int y = 11;
        int o = 3;
        for(int i = 0; i < x; i++){
            for(int j = 0; j < y; j++){
                for(int k = 0; k < o; k++){
                    if(results[i * (y + o) + j * (o) + k] == '1'){
                        objects[i, j, k] = 1;
                    } else {
                        objects[i, j, k] = 0;
                    }
                }
            }
        }
    }
/*   int x = 11;
    int y = 11;
    int o = 3;
    for(int i = 0; i < x; i++){
        for(int j = 0; j < y; j++){
            for(int k = 0; k < o; k++){
                objects[i, j, k] = 0;
            }
```

```
            if(Random.Range(0, 50) == 0){
                int val = Random.Range(0, o);
                objects[i, j, val] = 1;
            }
        }
    }*/
}

// Update is called once per frame
void Update()
{

    if(!finnished){
        timespent += Time.deltaTime;
        finnished = true;
        foreach(GameObject o_ in created){
            if(o_ != null){
                if(o_.tag == "Enemy"){
                    finnished = false;
                    break;
                }
            }
        }
        if(finnished){
            foreach(GameObject o_ in created){
                if(o_ != null){
                    Destroy(o_);
```

```
            }
        }
    }
}
if(finnished){
    player.GetComponent<Player>().health = 10;
    finnished = false;
    created.Clear();
    LoadEncouter();
    timespent = 0.0f;
    int x = 11;
    int y = 11;
    int o = 3;
    for(int i = 0; i < x; i++){
        for(int j = 0; j < y; j++){
            for(int k = 0; k < o; k++){
                if(objects[i, j, k] == 1){
                    Vector3 test = new Vector3(transform.position.x, transform.position.y, transform.position.z) + new Vector3(i * 2, 2, j) -
                    new Vector3(transform.localScale.x / 2, 0, transform.localScale.z / 2);
                    GameObject created_object = Instantiate(obj[k], test, transform.rotation);
                    created_object.SetActive(true);
                    created.Add(created_object);
                }
            }
        }
    }
```

```
        }
    }
}
```

Додаток Г.5 Лістинг модуля взаємодії з гравцем

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Player : MonoBehaviour
{
    public float health = 10.0f;
    public PlayerMove player;
    public Slider slider;
    public Attack playerAttack;
    public int cooldown = 0;
    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMove>();
        playerAttack = GameObject.FindGameObjectWithTag("Player").GetComponent<Attack>();


    }


    // Update is called once per frame
    void Update()
    {
```

```
        if(health <= 0){
            player.enabled = false;
            this.enabled = false;
        }
        slider.value = health;
        if (Input.GetButtonDown("Fire1") && cooldown == 0)
        {
            cooldown = 180;
            playerAttack.enabled = true;
        }
        if (cooldown > 0){
            cooldown -= 1;
        }

    }
}
public class PlayerMove : MonoBehaviour
{
    private CharacterController controller;
    private Vector3 playerVelocity;
    private bool groundedPlayer;
    private float playerSpeed = 5.0f;
    private float gravityValue = -9.81f;
    public float MouseSpeed = 2f;
    // Start is called before the first frame update
    void Start()
    {
```

```
        controller = gameObject.AddComponent<CharacterController>();
    }

    // Update is called once per frame
    void Update()
    {
        groundedPlayer = controller.isGrounded;
        if (groundedPlayer && playerVelocity.y < 0)
        {
            playerVelocity.y = 0f;
        }

        Vector3 move = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical")).normalized;
        controller.Move(move * Time.deltaTime * playerSpeed);

        if (move != Vector3.zero)
        {
            gameObject.transform.forward = move;
        }

        playerVelocity.y += gravityValue * Time.deltaTime;
        controller.Move(playerVelocity * Time.deltaTime);

        Vector3 mousePosition = Input.mousePosition;
        mousePosition = Camera.main.ScreenToWorldPoint(new Vector3(mousePosition.x, mousePosition.y, 11.5f));
```

```
        mousePosition = mousePosition - transform.position;
        mousePosition.y = 0;



        Quaternion targetRotation = Quaternion.LookRotation(mousePosition);
        transform.rotation = Quaternion.Slerp(targetRotation, targetRotation, 10 * Time.deltaTime);

    }
}
```

Додаток Д Лістинг серверу для обробки запитів нейронній мережі

Додаток Д.1 Лістинг запуску нейронної мережі

```
import tensorflow as tf


from nn.models import RoomPredictionModel



def predict(input_data):
    if not hasattr(predict, 'room'):
        predict.room = [1, 16, 11, 11, 3]
        predict.room = tf.zeros(predict.room)
        predict.metric = [1, 16, 1]
        predict.metric = tf.zeros(predict.metric)
        predict.model_holder = RoomPredictionModel("nn_1_2")
        predict.model_holder.load_model()
    predict.metric = tf.concat([predict.metric[:, 1:], [[[input_data]]]], axis=1)
    if predict.model_holder.model is None:
        return


    results = predict.model_holder.predict((predict.room, predict.metric))
    threshold = tf.cast(0.5, results.dtype)
    results = tf.cast(results > threshold, results.dtype)
    predict.room = tf.concat([predict.room[:, 1:], [results]], axis=1)
    results = tf.make_tensor_proto(results)
    results = tf.make_ndarray(results)
    results = results.tolist()
    return results
```

Додаток Д.2 Лістинг обробки запитів

```
from itertools import repeat


from flask import Flask
from nn.predict_room import predict


app = Flask(__name__)



@app.route("/<float:time>")
def hello_world(time):
    tmp = predict(time)[0]
    ret_data = ""
    for row in tmp:
        for cell in row:
            if all(map(any, repeat(iter(cell), 2))):
                ret_data += '000'
            else:
                for obj in cell:
                    ret_data += str(int(obj))
    return ret_data



if __name__ == "__main__":
    app.run()
```

Додаток Ж. Лістинг коду модульних тестів

Додаток Ж.1 Лістинг запуску тестів

```
import unittest
from test_nn import TestUser, TestPredict, TestLoss



if __name__ == '__main__':
    unittest.main()
```

Додаток Ж.2 Лістинг тестів нейронної мережі

```
import unittest

import numpy

from nn.models import UserTester, loss_function, RoomPredictionModel, room_convlstm2d_lstm
import tensorflow as tf



class TestUser(unittest.TestCase):
    def test_creation(self):
        user = UserTester(1, 1, 1)
        self.assertIsNotNone(user)
        self.assertIsInstance(user, UserTester)

    def test_init(self):
        user = UserTester(1, 1, 1)
        self.assertDictEqual(user.coff, {0: 1, 1: 1, 2: 1})
```

```
    def test_user_total(self):
        user = UserTester(1, 1, 1)
        test_x_room = [1, 11, 11, 3]
        test_x_room = tf.zeros(test_x_room)
        result = user.get_total(test_x_room)
        self.assertIsInstance(result, float)

    def test_user_loss(self):
        user = UserTester(1, 1, 1)
        test_x_room = [32, 11, 11, 3]
        test_x_room = tf.zeros(test_x_room)
        test_x_room = tf.make_tensor_proto(test_x_room)
        test_x_room = tf.make_ndarray(test_x_room)
        test_x_room = test_x_room.tolist()
        result = user.test(test_x_room)
        self.assertIsInstance(result, list)
        self.assertIsInstance(result[0], list)
        self.assertIsInstance(result[0][0], list)
        self.assertIsInstance(result[0][0][0], list)
        self.assertIsInstance(result[0][0][0][0], float)


class TestLoss(unittest.TestCase):
    def test_loss(self):
        lf = loss_function()
        test_x_room = [32, 11, 11, 3]
        test_x_room = tf.zeros(test_x_room)
```

```
        result = lf.call(test_x_room, test_x_room)
        self.assertIsNotNone(result)
        self.assertIsInstance(result, tf.Tensor)
        result = tf.make_tensor_proto(result)
        result = tf.make_ndarray(result)
        result = result.tolist()
        self.assertIsInstance(result, list)
        self.assertIsInstance(result[0], list)
        self.assertIsInstance(result[0][0], list)
        self.assertIsInstance(result[0][0][0], float)


class TestPredict(unittest.TestCase):
    def test_load(self):
        model_holder = RoomPredictionModel("unit_test")
        model_holder.load_model()
        self.assertIsNotNone(model_holder.model)

    def test_create(self):
        room = [1, 16, 11, 11, 3]
        room = tf.zeros(room)
        metric = [1, 16, 1]
        metric = tf.zeros(metric)
        model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="unit_test")
        result = model_holder.predict((room, metric))
        self.assertIsInstance(result, numpy.ndarray)
```

```
    threshold = tf.cast(0.5, result.dtype)
    result = tf.cast(result > threshold, result.dtype)
    result = tf.make_tensor_proto(result)
    result = tf.make_ndarray(result)
    result = result.tolist()
    self.assertIsInstance(result, list)
    self.assertIsInstance(result[0], list)
    self.assertIsInstance(result[0][0], list)
    self.assertIsInstance(result[0][0][0], list)
    self.assertIsInstance(result[0][0][0][0], float)

def test_train(self):
    room = [1, 16, 11, 11, 3]
    room = tf.zeros(room)
    metric = [1, 16, 1]
    metric = tf.zeros(metric)
    model_holder = RoomPredictionModel(model=room_convlstm2d_lstm(), name="unit_test")
    model_holder.model.compile(
        loss=loss_function(),
        optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
        metrics=[], run_eagerly=True
    )
    result = model_holder.fit(room, metric, room, 1)
    self.assertIsInstance(result.history, dict)
```

Додаток К

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ СТВОРЕННЯ ІГРОВИХ СИТУАЦІЙ

Розробка нейронної мережі для створення ігрових ситуацій

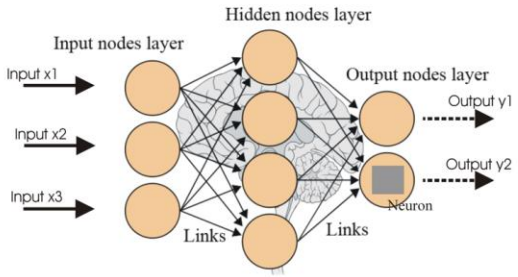
Студент гр. ЗПІ-22м Доценко Д. В.
Науковий керівник - д.т.н. проф. Романюк О. Н.

Вінниця - 2023

Рисунок В.1 – Слайд презентації № 1

Нейронні мережі

- Нейронні мережі, також відомі як штучні нейронні мережі (ШНМ) або симульовані нейронні мережі (СНМ), є підмножиною машинного навчання і є основою алгоритмів глибокого навчання. їх назва і структура натхненні людським мозком, імітуючи спосіб, як біологічні нейрони передають сигнали один одному.
- Штучні нейронні мережі (ШНМ) складаються з шарів вузлів, що включають в себе вхідний шар, один чи декілька прихованих шарів і вихідний шар. Кожен вузол, або штучний нейрон, з'єднаний з іншим і має асоційовану вагу і поріг. Якщо вихід будь-якого окремого вузла вище визначеного порогового значення, цей вузол активується, відправляючи дані на наступний шар мережі. В іншому випадку дані не передаються наступньому шару мережі.



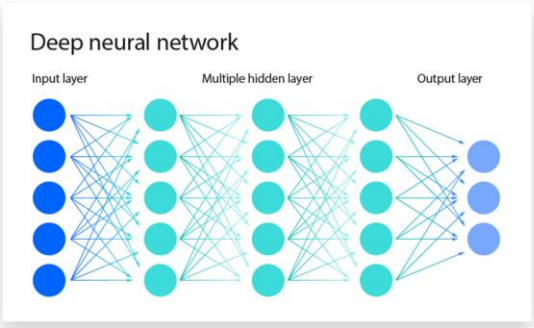
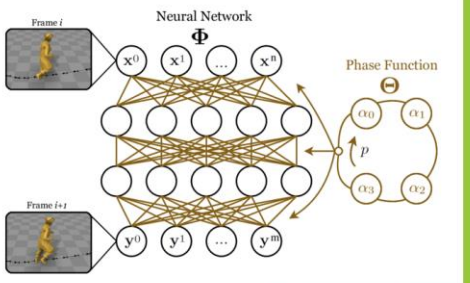


Рисунок В.2 – Слайд презентації № 2

Штучний інтелект у розробці ігор

- Концепція штучного інтелекту передбачає, що машинні технології допомагатимуть людині генерувати нові фантастичні ідеї. Найшвидше нейронні мережі розвиваються у 2D та 3D графіці. Можна за кілька секунд створити дивовижні предмети, людей та цілі планети й перенести це безпосередньо у 3D двигун.
- За допомогою нейронних мереж розробники можуть створювати обширні, різноманітні та непередбачувані ігрові середовища. Замість ручного проектування кожної гори, лісу чи річки алгоритми штучного інтелекту генерують їх, забезпечуючи унікальний досвід для кожного гравця.
- Нейронні мережі роблять оточення розумнішим. Замість використання заздалегідь визначених сценаріїв, персонажі з підвищеним штучним інтелектом можуть динамічно реагувати на дії гравця.



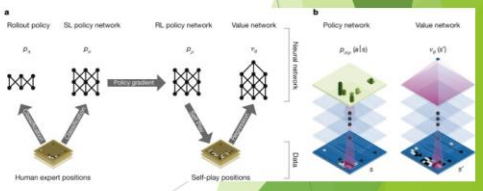


Рисунок В.3 – Слайд презентації № 3

Мета та завдання дослідження

Мета та завдання дослідження. Мета магістерської кваліфікаційної роботи полягає в розширенні функціональних можливостей гри за рахунок збільшення ігрових ситуацій та покращення динамічної складності.

Об'єкт дослідження - процес розробки нейронної мережі для генерації ігрових ситуацій.

Предмет дослідження є методи та засоби розробки нейронної мережі для генерації ігрових ситуацій.

Головні задачі дослідження є:

- Провести аналіз існуючих методів створення ігрових ситуацій, що використовуються при розробці сучасних ігор та методів розробки нейронної мережі.
- Розробити та провести тренування нейронної мережі для генерації ігрових ситуацій на основі уявних користувачів.
- Розробити гру для показу результату виконання нейронної мережі.
- Провести експериментальні дослідження розроблених засобів.

Рисунок В.4 – Слайд презентації № 4

Рисунок В.5 – Слайд презентації № 5

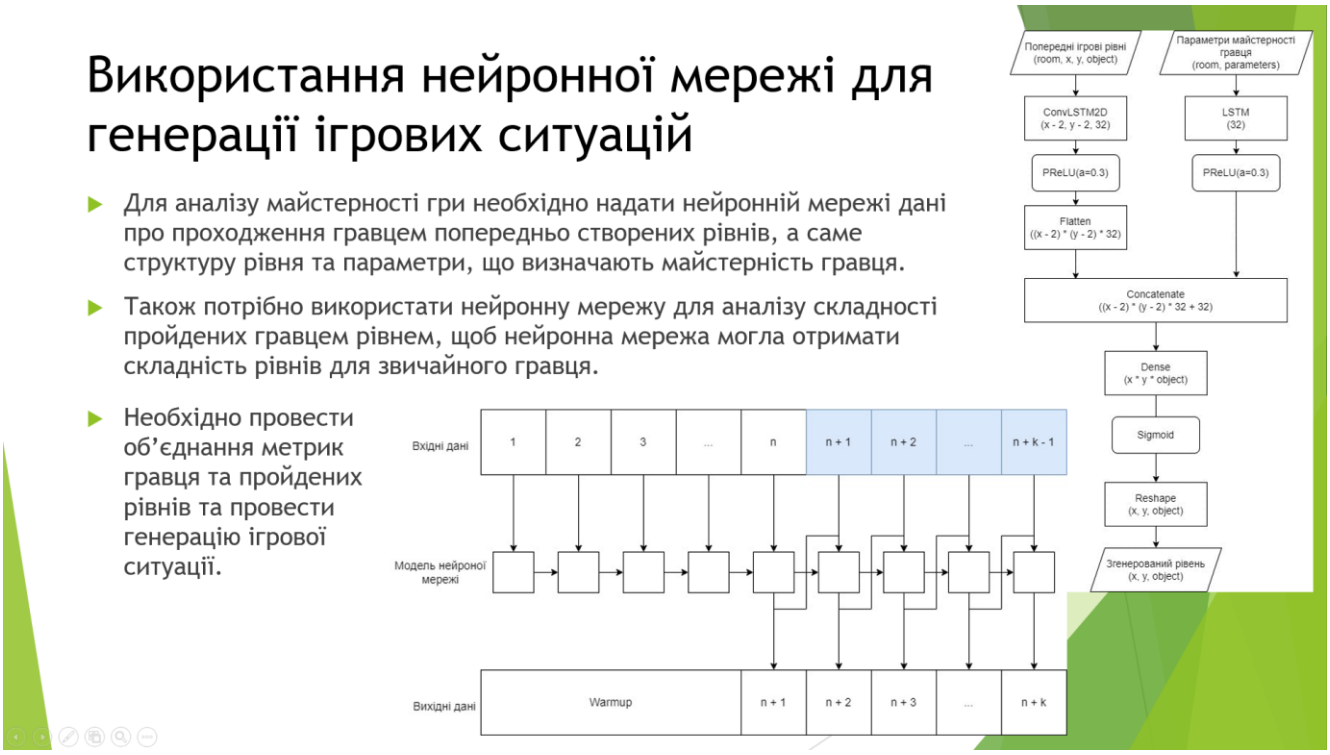
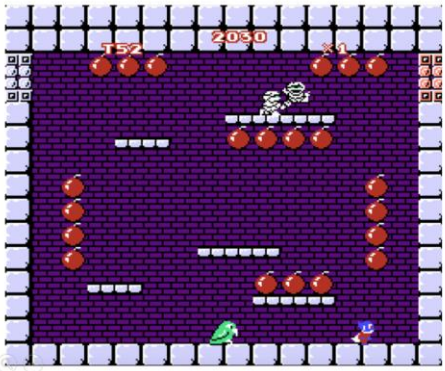


Рисунок В.6 – Слайд презентації № 6

Використання в іграх

- Розроблена нейронна мережа може бути використання в іграх за рахунок використання HTTP протоколу, що дозволяє уникнути проблему з необхідністю інтеграції НМ в гру.
- Для генерації ігрових ситуації (ІС) необхідно представити ІС у форматі матриці з попередньо визначними розмірами.
- Необхідно розробити об'єкти, що будуть розташовані в ІС.



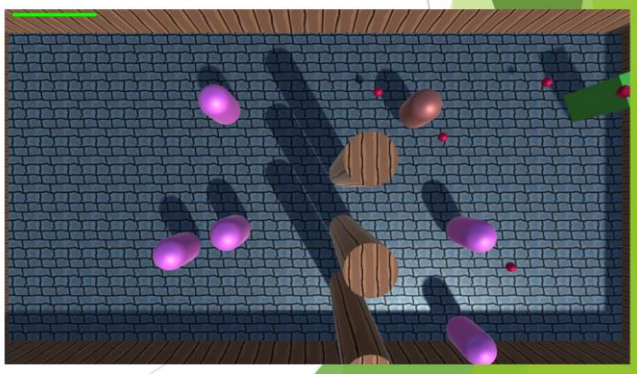


Рисунок В.7 – Слайд презентації № 7

Наукова новизна результатів. Практична цінність отриманих результатів.

- Уперше розроблено метод адаптивної зміни складності гри, особливість якого полягає у використанні нейронної мережі для динамічного аналізу майстерності гравця, що дало можливість збільшити інтелектуальний рівень гри і розширити її функціональні можливості;
- Уперше розроблено метод, який полягає у аналізі ігрових ситуацій за рахунок використання нової моделі НМ для аналізу складності ігрової ситуації, що дозволило покращити точність аналізу майстерності гравця;
- Подальшого розвитку отримав метод аналізу зображення за рахунок використання функції активації параметричного зрізаного лінійного вузла та додання додаткових шарів згортки, що дало можливість збільшити точність аналізу.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих у магістерській кваліфікаційній роботі результатів розроблено алгоритми та програми для нейронної мережі, яка генерує ігрові ситуацій, що може бути використана для пришвидшення розробки гри, збільшення різноманітності ігрових ситуацій та динамічної складності.

Рисунок В.8 – Слайд презентації № 8