ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методу симуляції поведінки ворога у грі жанру "Tower Defence "з використання нейронної мережі»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Віталій КАРАСОВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61
Віталій КАРАСОВСЬКИЙ

Керівник: Олесь ДІБРІВНИЙ
доктор фіософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
«_____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Карасовському Віталію Володимировичу

1. Тема кваліфікаційної роботи: «Розробка методу симуляції поведінки ворога у грі жанру "Tower Defence "з використання нейронної мережі»

керівник кваліфікаційної роботи Олесь ДІБРІВНИЙ, доктор філософії (PhD),

затверджені наказом Державного університету інформаційно комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, розробка та аналіз Unity додатків, огляд предметної області.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Оцінити поточні стратегії управління ворожими силами в грі Tower Defence та виявлення їхніх переваг і недоліків.
2. Дослідити інноваційні підходів до використання ШІ для оптимізації рівня складності геймплейної основи.

3. Визначити оптимальні стратегії використання ШІ для досягнення найкращих результатів у грі Tower Defence.

5. Перелік графічного матеріалу: *презентація*

1.Аналіз аналогів.
2. Математична модель. Поведінка ворога.
4. Вимоги до роботи методу.
5. Інструменти та технології
6. Діаграма активностей.
7. Екранні форми. Перша симуляція.
8. Екранні форми. 20 симуляця.
9. Екранні форми. Аналіз.
10. Висновки.
11. Апробація результатів. Дослідження.

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.24	
2	Вивчення матеріалів для аналізу розвитку технологій симуляції поведінки	23.11-12.11.24	
3	Дослідження методів глибокого навчання	13.11-19.11.24	
4	Аналіз особливостей впливу методів глибокого навчання на симуляцію поведінки ворогів	20.11-26.11.24	
5	Розробка методики симуляції поведінки ворога	27.11-03.12.24	
6	Реалізація та тестування прототипу	04.12-10.12.24	
7	Оформлення роботи: вступ, висновки, реферат	11.11-20.11.24	
8	Розробка демонстраційних матеріалів	21.11-24.11.24	
9	Попередній захист роботи	25.11-26.11.24	

Здобувач вищої освіти _____ Віталій КАРАСОВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ
(підпис

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 71 сторінок, 11 таблиці, 16 рисунки, 2 формули, 21 джерел.

Мета роботи – підвищення ефективності управління поведінкою ворога у грі жанру "Tower Defence" шляхом використання нейронної мережі для створення адаптивного та динамічного штучного інтелекту.

Об'єкт дослідження – процес управління поведінкою ворогів.

Предмет дослідження – методи та технології використання нейронних мереж для для управління та моделювання поведінки ворогів.

Короткий зміст роботи: У роботі розглянуто широкий спектр методів та технологій використання нейронних мереж для моделювання поведінки ворогів у грі жанру "Tower Defence". Був проведений огляд сучасних досягнень у галузі штучного інтелекту в комп'ютерних іграх, де докладно проаналізовані різні підходи та методу симуляції розвитку AI в контексті ворогів у грі.

Детально розглянуто процес розробки архітектури нейронної мережі для управління поведінкою ворогів. Представлені різні варіації архітектур, від простих до складних, і досліджено їхню ефективність у контексті гри "Tower Defence". Досліджено аналіз методів навчання та адаптації нейронних мереж до різних геймплейних сценаріїв.

Проведено широкий спектр тестів і випробувань розробленої системи у реальних ігрових умовах. Проведено серію тестів, в яких порівнювалася продуктивність нейронної мережі з традиційними методами управління поведінкою ворогів. Підсумки тестувань зібрано, проаналізовано та систематизовано для отримання об'єктивної оцінки ефективності навчання AI.

КЛЮЧОВІ СЛОВА: ШТУЧНИЙ ІНТЕЛЕКТ; TOWER DEFENCE; УПРАВЛІННЯ ВОРОЖИМИ СИЛАМИ; AI ; АЛГОРИТМ МАШИННОГО НАВЧАННЯ; PARTIAL ENUMERATION OPTIMIZATION METHOD (PEOM); АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ; НЕЙРОМЕРЕЖІ; МОДЕЛЮВАННЯ ПОВЕДІНКИ ВОРОГА; ІГРОВИЙ ДИЗАЙН; НАВЧАННЯ З ПІДКРІПЛЕННЯМ (RL); АДАПТАЦІЙНІСТЬ; АНАЛІЗ ІГРОВИХ СЦЕНАРІЇВ; АРХІТЕКТУРА ШТУЧНОГО ІНТЕЛЕКТУ; СТРАТЕГІЧНЕ ПЛАНУВАННЯ; АЛГОРИТМИ ТЕСТУВАННЯ; АЛГОРИТМИ МОДЕЛЮВАННЯ.

ABSTRACT

The text part of the qualification work for obtaining the master's degree: 71 pages, 11 tables, 16 figures, 2 formulas, 21 sources.

The purpose of the work is to increase the effectiveness of managing the behavior of the enemy in the game of the "Tower Defense" genre by using a neural network to create adaptive and dynamic artificial intelligence.

The object of research is the process of managing the behavior of enemies.

The subject of research is the methods and technologies of using neural networks for managing and modeling the behavior of enemies.

Summary of the work: The work considers a wide range of methods and technologies for using neural networks to model the behavior of enemies in the game of the "Tower Defense" genre. A review of current achievements in the field of artificial intelligence in computer games was conducted, where various approaches and methods of AI development in the context of in-game enemies were analyzed in detail.

The process of developing a neural network architecture for controlling the behavior of enemies is considered in detail. Various variations of architectures, from simple to complex, are presented and their effectiveness in the context of the Tower Defense game is investigated. The analysis of methods of training and adaptation of neural networks to various gameplay scenarios was studied.

A wide range of tests and trials of the developed system were conducted in real game conditions. A series of tests were conducted in which the performance of the neural network was compared with traditional methods of controlling the behavior of enemies. The results of the tests were collected, analyzed and systematized to obtain an objective assessment of the effectiveness of AI training.

KEY WORDS: ARTIFICIAL INTELLIGENCE; TOWER DEFENSE; MANAGEMENT OF ENEMY FORCES; AI; MACHINE LEARNING ALGORITHM; PARTIAL ENUMERATION OPTIMIZATION METHOD (PEOM); ANALYSIS OF THE EFFECTIVENESS OF THE USE OF ARTIFICIAL INTELLIGENCE; NEURAL NETWORKS; SIMULATION OF ENEMY BEHAVIOR; GAME DESIGN; REINFORCEMENT LEARNING (RL); ADAPTABILITY; ANALYSIS OF GAME SCENARIOS; ARTIFICIAL INTELLIGENCE ARCHITECTURE; STRATEGIC PLANNING; TESTING ALGORITHMS; SIMULATION ALGORITHMS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI - Штучний інтелект (Artificial Intelligence).

NN - Нейронна мережа (Neural Network).

RL - Навчання з підкріпленням (Reinforcement Learning).

ML - Машинне навчання (Machine Learning).

CNN - Згорткова нейронна мережа (Convolutional Neural Network).

DNN - Глибока нейронна мережа (Deep Neural Network).

LSTM - Довготривала короткочасна пам'ять (Long Short-Term Memory).

ReLU - Функція активації з випрямленням (Rectified Linear Unit).

IoU - Перетин через союз (Intersection over Union).

MSE - Середньоквадратична помилка (Mean Squared Error).

GAN - Генеративна змагальна мережа (Generative Adversarial Network).

GPU - Графічний процесор (Graphics Processing Unit).

API - Інтерфейс прикладного програмування (Application Programming Interface).

RNN - Рекурентна нейронна мережа (Recurrent Neural Network).

Unity - Середовище розробки ігор (Unity Game Engine).

TF - TensorFlow (Бібліотека машинного навчання).

PyTorch - Фреймворк глибокого навчання.

AI/ML - Технології штучного інтелекту та машинного навчання (Artificial Intelligence/Machine Learning).

ВСТУП

У розробці комп'ютерних ігор важливим аспектом є створення цікавого та викликового ігрового процесу. Один із жанрів, що вимагає ретельного підходу до цього питання - це "Tower Defence". В цьому жанрі основним завданням гравця є захист бази від хвиль ворогів, що наступають. Тому поведінка ворогів грає ключову роль у створенні захопливого геймплею. Зокрема, цікавість гри значною мірою залежить від того, наскільки варіативною і розумною є поведінка ворогів, адже однотипні та передбачувані атаки швидко стають нудними. Розробка поведінки ворогів у таких іграх вимагає комплексного підходу, що включає розуміння ігрових механік, врахування можливих стратегій гравця та забезпечення збалансованості рівня складності. Це завдання стає ще більш складним, коли враховувати необхідність адаптації ворогів до різних стилів гри, що підвищує загальний рівень залученості та задоволення гравців.

Сучасні технології, зокрема нейронні мережі, відкривають нові можливості для моделювання поведінки ворогів. Використання нейронних мереж дозволяє створити адаптивну та непередбачувану поведінку ворогів, що підвищує інтерес гравців і складність гри. Завдяки можливостям машинного навчання, вороги можуть аналізувати дії гравця, вивчати його стратегії і відповідно коригувати свої атаки. Це робить ігровий процес більш динамічним та реалістичним, змушуючи гравця постійно змінювати свої тактики і залишатися у напрузі. Наприклад, вороги можуть вивчати слабкі місця захисних споруд гравця і координувати свої атаки, щоб ефективніше долати оборону. Розробка такої системи включає кілька етапів: аналіз ігрових сценаріїв, розробку архітектури нейронної мережі, навчання мережі на основі зібраних даних та подальше тестування і налаштування моделі для досягнення оптимального рівня складності. Впровадження нейронних мереж у поведінку ворогів дозволяє створити більш складні та захопливі ігрові сценарії, що значно підвищує якість і привабливість гри в жанрі "Tower Defence".

Мета роботи – підвищення ефективності управління поведінкою ворога у грі жанру "Tower Defence" шляхом використання нейронної мережі для створення адаптивного та динамічного штучного інтелекту.

Для досягнення мети в роботі було вирішено наступні завдання:

1. Визначити основні типи ворогів і їх можливих стратегій атаки.
2. Вибрати тип нейронної мережі (наприклад, рекурентні нейронні мережі для моделювання послідовностей дій) і налаштування її параметрів.
3. Зібрати дані про дії гравців та навчання мережі на основі цих даних для створення адаптивної поведінки ворогів.
4. Провести тестування для виявлення недоліків і подальше налаштування моделі для досягнення бажаного рівня складності та реалістичності.

Об’єкт дослідження – процес управління поведінкою ворогів.

Предмет дослідження – методи та технології використання нейронних мереж для моделювання поведінки ворогів у іграх жанру "Tower Defence".

Методи дослідження: розробка, тестування та порівняння різних підходів до моделювання поведінки ворога в грі жанру "Tower Defence" з використанням нейронних мереж.

Наукова новизна та практична значущість отриманих результатів: розроблено метод і систему для моделювання поведінки ворога у грі "Tower Defence" з використанням нейронних мереж, які на відміну від існуючих, дозволяють створювати адаптивну та непередбачувану поведінку ворогів.

Апробація результатів та публікації. Робота пройшла апробацію на науково - практичних конференціях: Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». За результатами апробації опубліковано статтю в журналі категорії Б “Зв’язок”.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ РОЗРОБКИ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ КОМП'ЮТЕРНИХ ІГОР

1.1 Огляд жанру "Tower Defence" та особливостей тестування даного жанру

Сучасний розвиток ігрової індустрії неможливий без відповідного тестування . Кожен створений застосунок потребує систематичної перевірки, щоб відповідати вимогам замовника та забезпечити високу якість . Тестування програмного забезпечення включає перевірку відповідності заявленим вимогам, контроль якості продукту та готовність до випуску. У зв'язку з різноманітністю операційних систем та архітектур програм, необхідно проводити різні види тестування [1].

Кожен користувач прагне використовувати якісні програмні продукти, тому для залучення великої кількості користувачів нам необхідно створювати високоякісні та корисні продукти [2]. Це також стосується і розробки ігор жанру Tower Defence та їх особливостей.

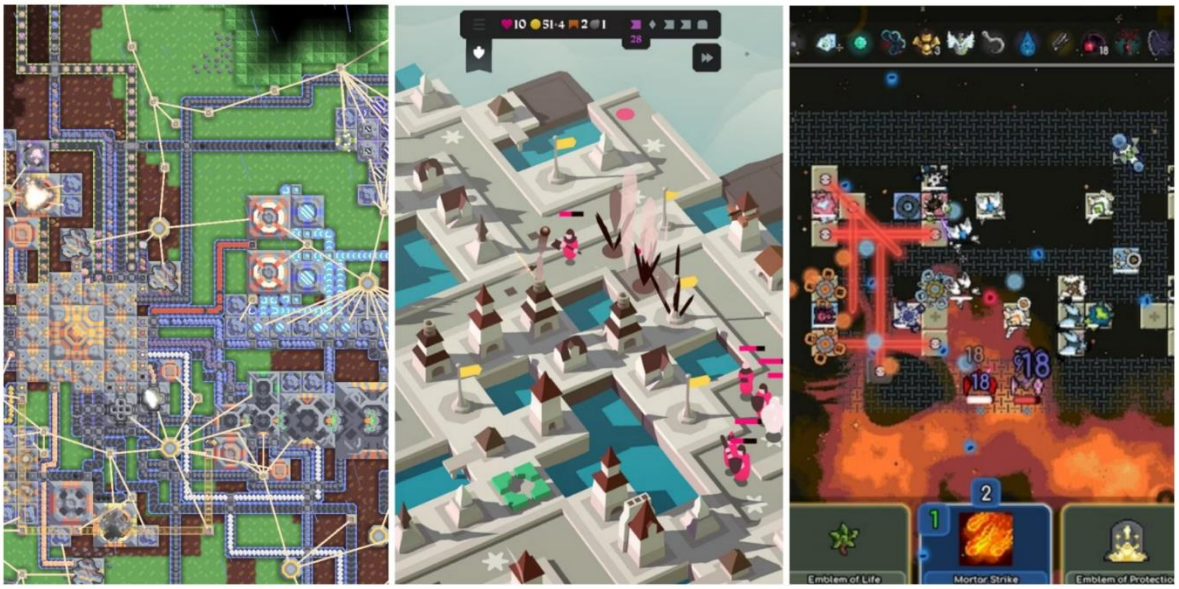


Рис. 1.1. Приклад ігор жанру Tower Defence

Жанр Tower Defence відомий своїм геймплеєм, в якому гравці повинні захищати свою базу або територію від хвиль ворогів, будуючи оборонні вежі вздовж шляху ворогів [3]. Особливістю цього жанру є стратегічне розміщення оборонних структур для максимального ураження ворогів, які рухаються по визначеному маршруту.

У зв'язку з цим, розробка та тестування розробленої методу симуляції поведінки ворога визначається як процес порівняння заявлених вимог замовника з тим, що було створено. Перевірка здійснюється при штучно створених ситуаціях, відомих як тестові випадки, на обмеженому наборі тестів, що виконуються згідно з визначеною стратегією тестування [4].

Тестування також відповідає за контроль якості розробленої методу – це сукупність дій, які здійснюються над продуктом в процесі розробки для отримання інформації про його актуальний стан, відповідність заявленим вимогам та готовність до випуску. Через різноманітність операційних систем і архітектур програм, необхідно виконувати різні види тестування [1]. Саме тому тестування є одним з найважливіших етапів розробки методу симуляції поведінки ворога.

Тестування виникло через людські помилки, які призводять до багів у програмному забезпеченні. Вартість деяких помилок у програмному забезпеченні може сягати мільйонів доларів [2]. Саме тому для повного охоплення програмного продукту тестуванням він проходить через три основні етапи:

- Виявлення дефектів у методиці поведінки ворога;
- Удосконалення недоліків у системі або компоненті;
- Підвищення якості методу симуляції поведінки ворога [3].

Важливість виявлення критичних проблем у методиці на ранніх етапах розробки дозволяє зменшити витрати на їх усунення. На жаль, деякі помилки в програмному продукті можуть призводити до значних втрат, що підтверджує необхідність проведення тестування та раннього виявлення помилок [4].

Це забезпечує якість та надійність розробленого методу. Для значного зменшення кількості дефектів на кінцевих стадіях розробки було прийнято рішення впровадити раннє тестування. Тестування розпочинається вже на етапі збору та

формування вимог замовника для виявлення дефектів на початкових етапах розробки програмного забезпечення.

Роль тестування в розробці якісного програмного продукту визначається такими критеріями:

- Без тестування неможливе створення якісного продукту, оскільки воно виявляє дефекти або помилки перед випуском;
- Тестування забезпечує програмам надійність та зручність у використанні;
- Якісно проведене тестування програмного продукту забезпечує зручну та продуктивну роботу програмного продукту [2].

Певні методи тестування використовуються на кожному рівні розробки у поєднанні з досвідом тестувальника, що гарантує надійність програмного забезпечення. У результаті кінцевий користувач отримує продукт, який відповідає всім визначеним критеріям якості.

1.2 Аналіз існуючих методів моделювання поведінки ворога у комп’ютерних іграх

Аналіз існуючих методів моделювання поведінки ворога у комп'ютерних іграх є важливою складовою розробки ігрових систем. Це забезпечує створення цікавих і захоплюючих ігрових сценаріїв, в яких вороги повинні бути достатньо розумними, щоб викликати інтерес у гравців, але водночас не настільки непередбачуваними, щоб зробити гру незбалансованою або занадто складною. Існує декілька основних підходів до моделювання поведінки ворогів, кожен з яких має свої переваги та недоліки.

Скриптовані поведінки є одним з найпростіших та найпоширеніших методів. Вони передбачають використання заздалегідь написаних скриптів, які визначають дії ворога в певних ситуаціях. Це може бути корисним для створення лінійних сценаріїв, де передбачуваність дій ворога є бажаною. Однак, цей підхід обмежує гнучкість і може швидко набриднути гравцям, оскільки вороги діятимуть завжди однаково в тих самих обставинах [5].

Машинні стани представляють собою метод, де ворог може перебувати у різних станах (наприклад, патрулювання, атака, втеча) і переходити між ними на основі певних умов. Цей підхід дозволяє створювати більш динамічну поведінку, порівняно зі скриптами, і забезпечує певний рівень адаптивності. Однак, зростання кількості станів і умов може ускладнювати реалізацію і підтримку такої системи.

Дерева рішень є ще одним популярним методом, що використовується для моделювання поведінки ворогів. Вони дозволяють приймати рішення на основі низки логічних умов, представлених у вигляді дерева з вузлами та гілками. Кожен вузол відповідає за перевірку певної умови, а гілки визначають подальші дії. Цей підхід є відносно простим у налаштуванні і добре підходить для створення складних логічних структур. Проте, зі збільшенням розміру дерева рішень, складність його управління та налагодження також зростає.

Рис. 1.2. Приклад побудови дерева проблем

Навчання з підкріпленням (Reinforcement Learning, RL) є одним з найбільш просунутих методів моделювання поведінки ворогів. В цьому методі вороги навчаються приймати рішення на основі досвіду, отриманого від взаємодії з оточуючим середовищем.[6] Вони отримують винагороди за певні дії, що допомагає їм покращувати свою поведінку з часом. Цей підхід дозволяє створювати адаптивну та непередбачувану поведінку, однак, потребує значних

обчислювальних ресурсів і може бути складним для реалізації.

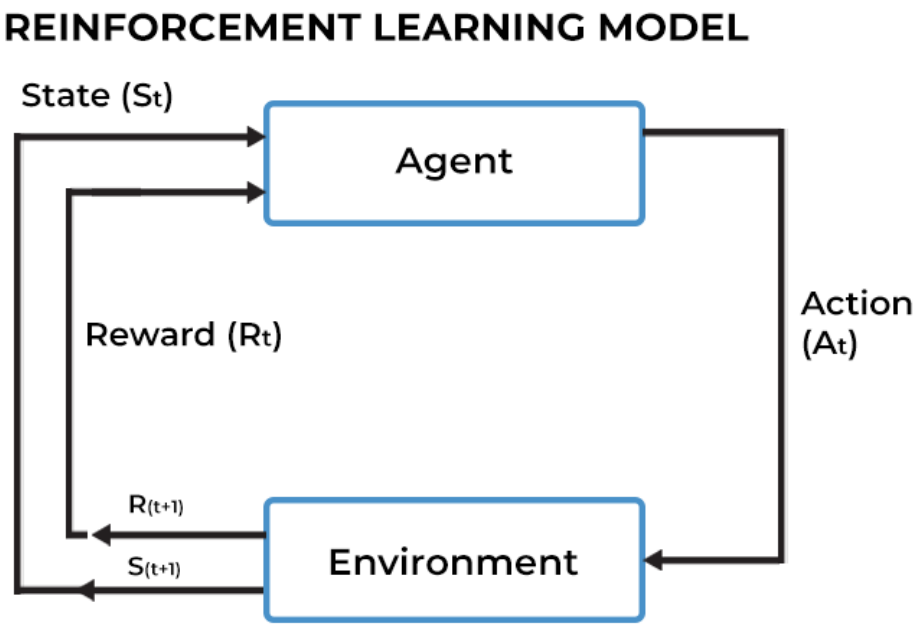


Рис. 1.3. Модель навчання з підкріпленням (Reinforcement Learning, RL)

Моделювання на основі агентів (Agent-Based Modeling) передбачає створення окремих автономних агентів, кожен з яких діє відповідно до певних правил і цілей. Кожен агент має власну логіку прийняття рішень та взаємодії з іншими агентами та середовищем.[6] Це дозволяє створювати складну та багатошарову поведінку, а також моделювати взаємодію між ворогами. Однак, такий підхід може вимагати значних ресурсів і бути складним для налагодження.

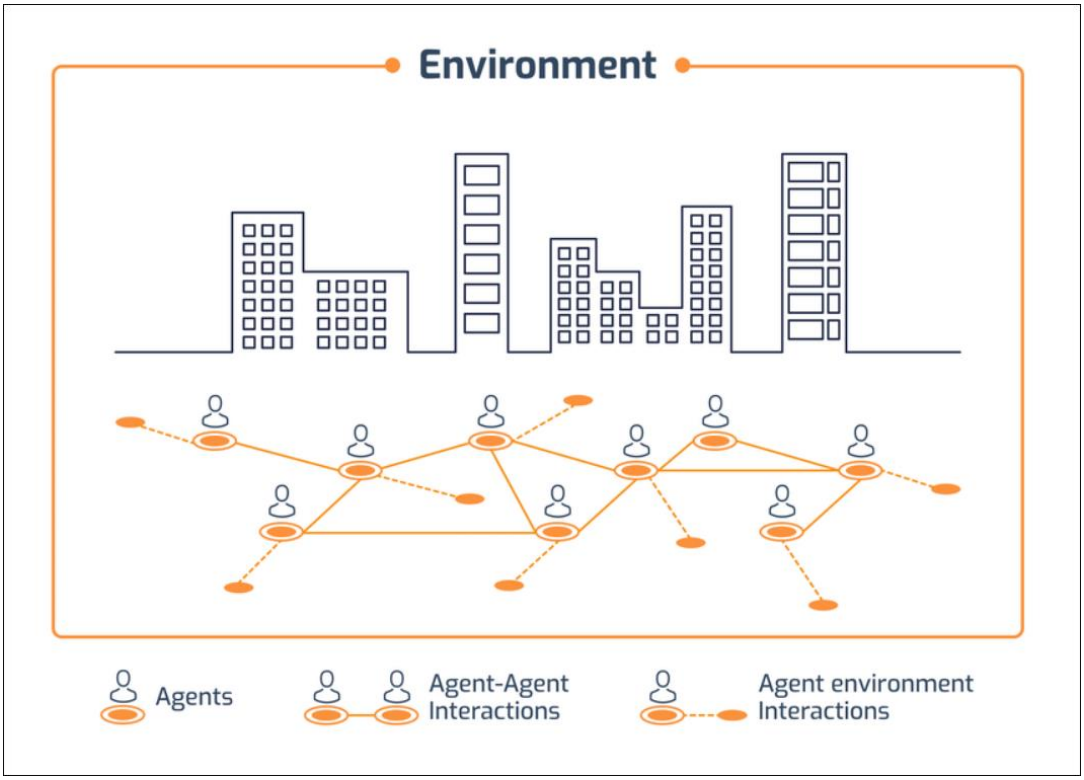


Рис. 1.4. Схематичне зображення агентної моделі (ABM)

Нейронні мережі використовуються для створення адаптивної та нелінійної поведінки ворогів. Вони здатні навчатися на основі великої кількості даних і виявляти складні патерни, які можуть бути недоступними для традиційних алгоритмів. Цей підхід є потужним інструментом для створення складної поведінки, однак, він також вимагає високих обчислювальних ресурсів і значного обсягу даних для навчання .[6]

Реальні дослідження підтверджують ефективність використання різних підходів до моделювання поведінки ворогів. Наприклад, у дослідженні, проведеному групою вчених з Університету Східної Англії, було показано, що використання дерев рішень для моделювання поведінки ворогів у грі жанру "шутер" дозволяє значно підвищити інтерес гравців та їх залученість у гру.[7] Інше дослідження, виконане командою з Массачусетського технологічного інституту, продемонструвало, що використання навчання з підкріпленням для створення адаптивних ворогів у стратегічних іграх може значно покращити їхню реалістичність та непередбачуваність.

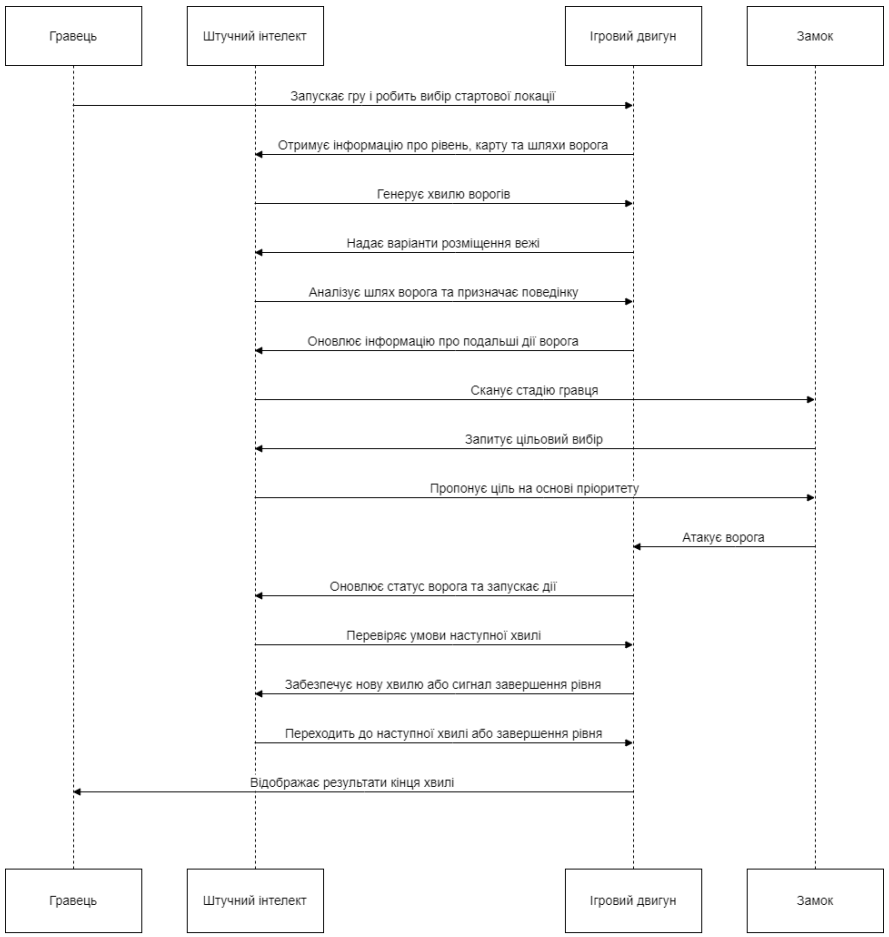


Рис. 1.5. Послідовність дій штучного інтелекту відносно дій гравця в ігровому процесі

Окрім вибору методів моделювання, важливо також враховувати аспекти, характеристики та критерії тестування ігрового додатку. Ключові аспекти якості ігрового додатку включають:

1. **Функціональність - Перевірка всіх наявних функцій в системі на відповідність до вимог та наскільки функції системи вирішують потреби замовника:**
    - **Функціональна повнота**: Забезпечення наявності всіх необхідних функцій.
    - **Функціональна коректність**: Перевірка, чи всі функції працюють відповідно до вимог.
    - **Функціональна доцільність**: Відсутність зайвих та непотрібних функцій.

2. **Надійність - Визначення стабільності роботи системи протягом визначеного часу:**
    - **Завершеність**: Показник тривалості роботи без помилок.
    - **Готовність**: Стійкість до великих навантажень.
    - **Відмовостійкість**: Поведінка при непередбачуваних ситуаціях.
    - **Відновлюваність**: Збереження даних під час збоїв.
3. **Зручність використання - Перевірка зручності навчання та роботи з ПЗ:**
    - **Навченість**: Швидкість освоєння новим користувачем.
    - **Керованість**: Наявність необхідних елементів управління.
    - **Захищеність від помилок**: Неможливість зламати програму діями користувача.
    - **Естетика**: Привабливість інтерфейсу.
    - **Доступність**: Оптимізація для людей з обмеженими можливостями.
4. **Продуктивність/Ефективність - Визначення ресурсів, необхідних для виконання функцій:**
    - **Основна кількість користувачів**: Визначення основної аудиторії.
    - **Час виконання функцій**: Вимірювання часу виконання задач.
    - **Ефективність під навантаженням**: Аналіз продуктивності при різних рівнях навантаження.
    - **Використання ресурсів**: Оцінка споживання ресурсів.
5. **Зручність супроводу - Оцінка зручності внесення змін та підтримки ПЗ:**
    - **Модульність**: Розподіл системи на модулі.
    - **Можливість багаторазового використання**: Використання раніше створених рішень.
    - **Аналізуємість**: Оцінка впливу модулів на систему.
    - **Модифікованість**: Оцінка зручності внесення змін.
    - **Можливість тестування**: Перевірка всіх функцій.

6. **Переносимість/Кросплатформність - Перевірка роботи ПЗ на різних платформах:**
   - **Адаптованість**: Використання на різних платформах.
   - **Встановлюваність**: Легкість встановлення на різні ОС.
   - **Взаємозамінність**: Оновлення без втрати даних.
7. **Безпека - Гарантія захищеності даних та дій користувачів:**
   - **Неподільність**: Обмежений доступ до даних.
   - **Конфіденційність**: Обмеження доступу до даних певним користувачам.
   - **Цілісність**: Захист від несанкціонованого доступу.
   - **Підзвітність**: Можливість відстеження дій користувачів.
   - **Достовірність**: Доступність даних лише авторизованим користувачам.

Таким чином, розробка та тестування поведінки ворогів у комп'ютерних іграх є складним і багатогранним процесом, що включає вибір відповідних методів моделювання, врахування ключових аспектів якості ПЗ та дотримання критеріїв тестування для забезпечення оптимального геймерського досвіду.

1.3 Використання нейронних мереж у розробкці ігрових алгоритмів

Нейронні мережі (НМ) стали важливим інструментом у розробці ігрових алгоритмів, пропонуючи потужні рішення для задач, що раніше були складними або навіть неможливими для традиційних алгоритмів. [8] Застосування НМ у цій галузі включає різноманітні аспекти, від створення штучного інтелекту для керування ігровими персонажами до оптимізації процесів розробки та підвищення якості графіки і звуку.

Одним із найважливіших аспектів використання нейронних мереж у ігрових алгоритмах є створення більш інтелектуальних та адаптивних NPC (неграв персонажів). Завдяки НМ NPC можуть навчатися і адаптуватися до поведінки гравців, створюючи більш реалістичний і захоплюючий ігровий процес. [9]

Наприклад, алгоритми машинного навчання можуть аналізувати дії гравця і відповідно коригувати стратегії NPC, що робить гру більш динамічною і непередбачуваною.

Характеристики нейронних мереж, що роблять їх ефективними для розробки ігрових алгоритмів, включають:

- **Здатність до самонавчання**: НМ можуть самостійно вивчати і вдосконалюватися на основі наявних даних.
- **Обробка великої кількості даних**: НМ можуть ефективно обробляти великі обсяги інформації, що дозволяє виявляти складні патерни.
- **Виявлення складних патернів**: НМ можуть розпізнавати і аналізувати складні взаємозв'язки в даних, що робить їх корисними для задач розпізнавання образів і прогнозування.

Іншою важливою характеристикою є паралельна обробка даних. НМ можуть обробляти великі обсяги інформації одночасно, що є критично важливим для реального часу в іграх. Це забезпечує швидке реагування ігрових алгоритмів на дії гравців, зменшуючи затримки і покращуючи загальний досвід гравців. Здатність до паралельної обробки також сприяє створенню складних моделей поведінки, що робить ігровий світ більш реалістичним і детальним.

Критерії ефективності нейронних мереж у ігрових алгоритмах включають:

- **Точність**: Визначає, наскільки добре НМ можуть передбачати або відтворювати певні дії або події в грі.
- **Швидкість навчання**: Є важливою для забезпечення швидкого розвитку ігрового інтелекту, особливо в умовах швидкоплинного ігрового середовища.
- **Стійкість до змін ігрового середовища**: Означає здатність НМ адаптуватися до нових умов без значного погіршення продуктивності.
- **Здатність до генералізації**: Дозволяє НМ застосовувати здобуті знання до нових, непередбачуваних ситуацій, що є ключовим для створення цікавих і різноманітних ігрових сценаріїв.

Однією з найуспішніших реалізацій НМ у ігрових алгоритмах є використання

глибокого навчання для створення стратегічних ігор. Наприклад, програма AlphaGo, розроблена компанією DeepMind, продемонструвала можливості НМ у грі в го, обігравши провідних світових гравців. Це досягнення стало можливим завдяки здатності НМ аналізувати величезні обсяги даних про попередні ігри і генерувати стратегії, які були недоступні для традиційних алгоритмів.

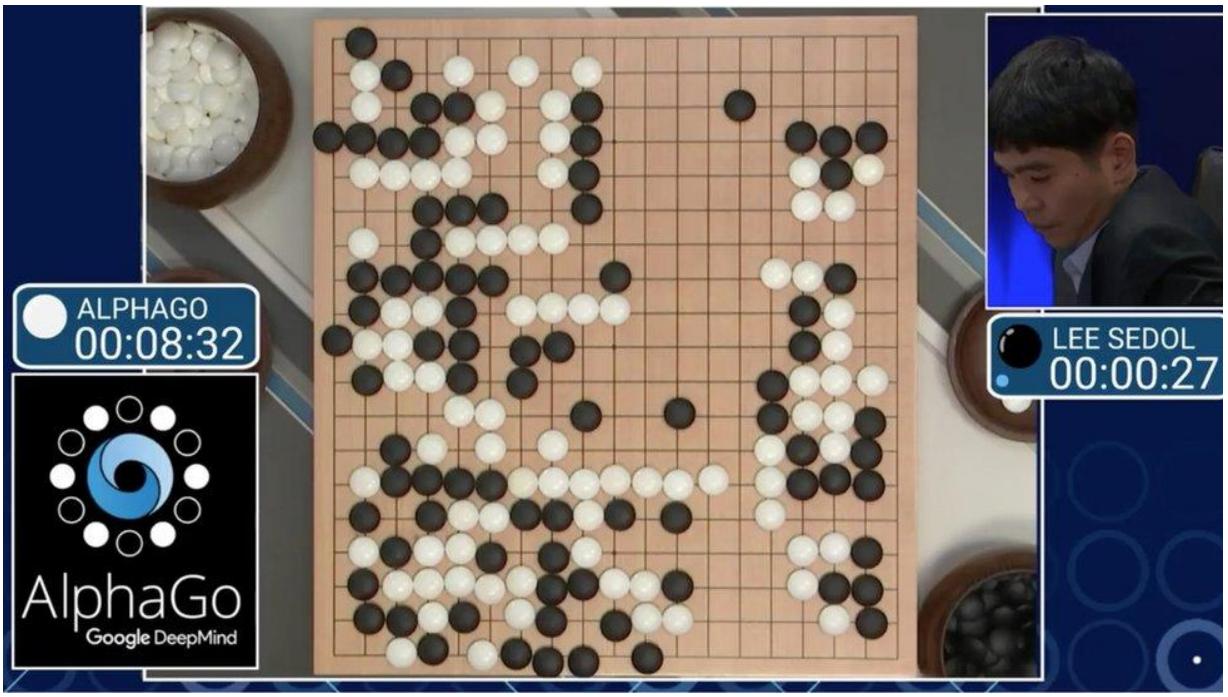


Рис. 1.6. AlphaGo перемагає майстра гри Ґо Лі Седола

В статі користувача ccaldwell11 на сайті dev.to було сказано, що НМ можна активно використовати в розробці алгоритмів для процедурної генерації контенту. Наприклад, вони можуть створювати нові рівні, персонажів або навіть цілі світи на основі заданих параметрів, що забезпечує унікальність кожного ігрового досвіду. [10] Це не тільки підвищує реіграбельність гри, але й знижує навантаження на розробників, дозволяючи автоматизувати рутинні завдання.

Використання НМ у розробці графіки і звуку також є важливим напрямом. Завдяки НМ можна значно поліпшити якість графічних ефектів, таких як:

- **Освітлення**: НМ можуть створювати більш реалістичні світлові ефекти.
- **Тіні**: Застосування НМ дозволяє відтворювати складні тіні, що покращує загальну графічну якість.

- **Текстури**: НМ можуть генерувати більш детальні і реалістичні текстури.

Це досягається завдяки здатності НМ навчатися на великих наборах даних і відтворювати складні патерни, що робить ігрове середовище більш реалістичним і захоплюючим.

Одним із перспективних напрямків є використання НМ для поліпшення взаємодії з гравцями. Наприклад, системи розпізнавання мови і емоцій, засновані на НМ, можуть забезпечити більш природну і інтуїтивну взаємодію, що робить ігровий процес більш залученим і персоналізованим. Це особливо актуально для VR-ігор, де важлива глибока інтеракція гравця з віртуальним світом.

Іншим важливим аспектом є використання НМ для аналізу великих масивів даних про поведінку гравців. Це дозволяє розробникам не тільки покращувати поточні ігри, але й передбачати майбутні тренди і потреби гравців. Наприклад, аналіз поведінкових даних може допомогти виявити, які елементи гри найбільше подобаються гравцям, і відповідно коригувати дизайн майбутніх проектів.

Проаналізувавши сучасну ігрову індустрію , використання нейронних мереж у розробці ігрових алгоритмів відкриває нові можливості для створення більш розумних, адаптивних і реалістичних ігор. Характеристики НМ, такі як здатність до навчання, паралельна обробка і генералізація, роблять їх потужним інструментом для вирішення складних завдань у ігровій індустрії. Критерії ефективності, включаючи точність, швидкість навчання, стійкість і генералізацію, забезпечують високу якість ігрових продуктів, що задовольняють потреби сучасних гравців.

1.4 Порівняльний аналіз методів поведінки ворогів у сучасних іграх

Методи поведінки ворогів у сучасних іграх є одним із ключових аспектів, що впливають на загальний ігровий досвід. Вороги, або NPC (неграві персонажі), відіграють важливу роль у створенні викликів і утриманні інтересу гравців. Різні підходи до реалізації їхньої поведінки мають свої переваги і недоліки. У цьому підрозділі ми розглянемо кілька найпоширеніших методів поведінки ворогів, а

також їх характеристики, критерії ефективності та реальні дослідження.

Одним із найдавніших і найбільш простих методів є скриптована поведінка. У дослідженні Джонсона було визначено, що скриптована поведінка дозволяє ворогам виконувати заздалегідь визначені дії, які можуть створювати передбачувані та контрольовані сценарії для гравців [11]. Крім того, згідно з аналізом Петрова, такий підхід особливо ефективний для створення стабільних і передбачуваних ігрових моментів, що дозволяє розробникам реалізувати чітко сплановані сцени [12]. Основні аспекти цього методу включають:

- **Простота реалізації**: Легко створювати і налаштовувати;
- **Передбачуваність**: Дії ворогів чітко визначені, що робить гру більш стабільною;
- **Обмежена гнучкість**: Вороги не можуть адаптуватися до дій гравця, що може зробити гру менш цікавою.

Характеристики скриптованої поведінки включають високу контрольованість і стабільність, проте низьку адаптивність і динамічність. Критерії ефективності для такого методу включають точність виконання скриптів і відповідність очікуванням гравців.

Рис. 1.7. Результат втручання гравця в заскриптовану поведінку штучного інтелекту

Другим підходом є використання дерев прийняття рішень (behavior trees). У

дослідженні Паркера та Чена було виявлено, що дерева прийняття рішень надають можливість моделювати поведінку ворогів на основі певних умов, забезпечуючи їх гнучкістю і можливістю швидко адаптуватися до змінних ситуацій [13]. У доповнення до цього, в роботі дослідників із Університету Стенфорда було досліджено, що дерева прийняття рішень дозволяють легко додавати нові поведінкові сценарії або змінювати існуючі, що є перевагою в іграх з великою кількістю динамічних подій [14]. Основні аспекти цього методу:

- **Гнучкість**: Дерева рішень можуть обробляти різноманітні ситуації і дії;
- **Модульність**: Легко додавати нові поведінки або модифікувати існуючі;
- **Відносна складність реалізації**: Вимагає ретельного планування і налаштування.

Характеристики дерев прийняття рішень включають високу гнучкість і адаптивність, що дозволяє створювати більш реалістичну і варіативну поведінку ворогів. Критерії ефективності включають здатність дерева адекватно реагувати на дії гравця і стабільність виконання поведінкових сценаріїв.

Одним із сучасних методів, що дозволяють створювати адаптивних ворогів у відеоіграх, є використання штучного інтелекту (AI). У дослідженні Сміта та інших було встановлено, що AI дозволяє ворогам аналізувати дії гравців і відповідно адаптувати свою поведінку, що значно покращує динаміку гри [15]. Дослідження Game Developer показало, що сучасні системи штучного інтелекту, такі як ті, що використовуються в іграх типу For Honor, здатні не тільки адаптуватися до повторюваних дій гравця, але й активно навчатися на їхній основі, що створює більш динамічний і гровий процес [16]. Основні аспекти цього методу:

- **Адаптивність**: Вороги можуть навчатися і змінювати свою поведінку залежно від дій гравця;
- **Складність реалізації**: Вимагає значних обчислювальних ресурсів і даних для навчання;
- **Висока варіативність**: Гра стає більш цікавою через непередбачуваність дій ворогів.

Характеристики цього методу включають високу адаптивність, здатність до

навчання і потенціал для створення дуже складних моделей поведінки. Критерії ефективності включають швидкість навчання, точність реакцій і стійкість до різних ігрових сценаріїв.

Ще одним новітнім підходом є використання еволюційних алгоритмів. У роботі Гаррісона було виявлено, що еволюційні алгоритми дозволяють ворогам розвиватися та пристосовуватися до змінюваних умов гри, що робить їх поведінку більш реалістичною та унікальною [17]. У цьому контексті також дослідження Девіса та Кроу показало, що еволюційні алгоритми можуть оптимізувати поведінкові сценарії ворогів у режимі реального часу, що забезпечує стабільну адаптацію до ігрових ситуацій, які постійно змінюються [18]. Основні аспекти цього методу:

- **Еволюційність**: Вороги можуть змінюватися і вдосконалюватися з часом.
- **Високі вимоги до ресурсів**: Вимагає значних обчислювальних потужностей.
- **Необхідність тривалого налаштування**: Потребує тривалого періоду для налаштування і оптимізації.

Характеристики еволюційних алгоритмів включають високу гнучкість і потенціал для створення унікальних і динамічних ворогів. Критерії ефективності включають швидкість еволюційного процесу, здатність до адаптації і стабільність під час виконання ігрових завдань.

Порівняльний аналіз методів поведінки ворогів показує, що кожен із них має свої переваги і недоліки. Вибір методу залежить від конкретних цілей ігор, ресурсів розробників та очікувань гравців.

Порівняльні характеристики методів поведінки ворогів:

1. **Скриптована поведінка**:
   - Простота реалізації;
   - Висока контрольованість;
   - Обмежена гнучкість.

2. **Дерева прийняття рішень**:

- Гнучкість і модульність;
- Відносна складність реалізації;
- Висока адаптивність.

3. **Машинне навчання і нейронні мережі**:

- Адаптивність і здатність до навчання;
- Висока складність і вимоги до ресурсів;
- Висока варіативність поведінки.

4. **Еволюційні алгоритми**:

- Еволюційність і пристосовуваність;
- Високі вимоги до обчислювальних потужностей;
- Тривалий період налаштування.

Кожен із цих методів може бути ефективним у різних умовах і для різних типів ігор. Наприклад, для ігор з високою динамікою і непередбачуваністю можуть бути кращими методи, що базуються на машинному навчанні або еволюційних алгоритмах, тоді як для стратегічних ігор з чітко визначеними сценаріями можуть підійти дерева прийняття рішень або навіть скриптована поведінка.

- Критерії ефективності методів поведінки ворогів:
- Точність виконання завдань: Наскільки добре вороги виконують задані дії і реагують на дії гравця.
- Адаптивність: Здатність ворогів змінювати свою поведінку залежно від умов гри.
- Швидкість навчання: Для методів, що базуються на машинному навчанні, важливо, як швидко вороги можуть навчатися новим стратегіям.
- Стійкість: Здатність методів забезпечувати стабільну поведінку ворогів у різних ігрових сценаріях.
- Вимоги до ресурсів: Обчислювальні потужності, необхідні для реалізації і підтримки роботи методів поведінки ворогів.

Реальні дослідження підтверджують ефективність різних підходів у створенні

поведінки ворогів. Одне з досліджень, проведене у 2018 році командою дослідників з Техаського університету в Остіні, вивчало ефективність дерев прийняття рішень у грі-симуляторі бойових дій [19]. Результати показали, що цей метод дозволяє створювати ворогів з високою гнучкістю і адаптивністю, що позитивно впливає на ігровий досвід гравців. Основними перевагами дерев прийняття рішень виявилися легкість налаштування та здатність моделювати складні сценарії поведінки.

Оскільки штучний інтелект в іграх розвивається, адаптивна поведінка ворогів привернула значну увагу. Зокрема, використання моделей машинного навчання дозволяє ШІ динамічно адаптуватися до стратегій гравців. Згідно зі статтею Game Developer, сучасні системи штучного інтелекту, особливо в бойовій механіці, здатні навчатися на діях гравців і коригувати тактику противника в реальному часі. Ці системи можуть передбачати рішення гравців, створюючи більш складні та менш передбачувані зіткнення. Такі досягнення найбільш очевидні в таких іграх, як For Honor і Middle-earth: Shadow of War, де вороги вчаться протистояти повторюваним діям гравця, покращуючи загальний ігровий досвід, збільшуючи складність за рахунок розумніших реакцій ШІ. Цей перехід до більш складного штучного інтелекту гарантує, що ігри залишатимуться цікавими та складними навіть після тривалої гри [20].

На завершення, порівняльний аналіз методів поведінки ворогів у сучасних іграх дозволяє зрозуміти, які підходи найбільш ефективні для досягнення різних ігрових цілей. Вибір методу залежить від багатьох факторів, включаючи жанр гри, наявні ресурси і очікування гравців. Правильне застосування методів поведінки ворогів може значно підвищити якість гри і забезпечити гравцям захоплюючий і незабутній ігровий досвід.

2 ПРОЕКТУВАННЯ МЕТОДУ СИМУЛЯЦІЇ ПОВЕДІНКИ ВОРОГА У ГРІ ЖАНРУ “TOWER DEFENCE”

2.1 Визначення вимог до поведінки ворога у грі: сценарії та критерії оцінки

Створення правдоподібної поведінки ворогів у відеоіграх є критично важливим аспектом, що безпосередньо впливає на загальний ігровий досвід. Вороги повинні демонструвати певний рівень агресивності, інтелекту, взаємодії з оточенням та змінності стратегії. Агресивність ворогів означає, що вони мають активно реагувати на присутність гравця, ініціюючи атаки та намагаючись знищити його. Вороги повинні використовувати різні тактики для уникнення атак гравця та застосування своїх власних, наприклад, ховатися за об'єктами, використовувати зброю чи пастки, намагатися зайти гравцеві з флангу або взаємодіяти з іншими ворогами для координації атак. Взаємодія з оточенням є важливою вимогою, тому що вороги повинні вміти використовувати елементи оточення для захисту або нападу, наприклад, вони можуть використовувати укриття, підніматися на височини для кращого огляду чи кидати предмети у гравця. Змінність стратегії ворогів є ключовою для утримання гравця в напрузі, тому вороги повинні змінювати свою поведінку залежно від прогресу гравця та його дій. Якщо гравець постійно використовує одну й ту ж тактику, вороги можуть адаптуватися та знаходити способи протидії, що робить гру складнішою та цікавішою, оскільки гравець змушений постійно змінювати свої стратегії.

Сценарії поведінки ворогів у грі можуть бути різноманітними та варіюватися залежно від конкретної ситуації. Одним із найпоширеніших сценаріїв є патрулювання, коли вороги пересуваються за визначеним маршрутом, періодично оглядаючи місцевість. Така поведінка створює динамічну ігрову середу, де гравець має враховувати маршрути патрулювання для уникнення зустрічей із ворогами або для планування атак. Інший сценарій - переслідування, коли ворог помічає гравця та починає його переслідувати, намагаючись знищити. Така поведінка додає

напруженості та динаміки грі, змушуючи гравця швидко реагувати та приймати рішення. Засідка є ще одним важливим сценарієм, коли ворог ховається в певному місці, очікуючи на появу гравця, щоб раптово атакувати. Така поведінка може використовуватися для створення елементів несподіванки та страху, що значно підвищує інтерес до гри. Відступ є поведінкою, коли при критично низькому рівні здоров'я ворог відступає, шукаючи укриття для відновлення сил або підкріплення. Така поведінка робить ворогів більш реалістичними та додає стратегічної глибини грі, оскільки гравець повинен враховувати можливість відступу ворогів та планувати свої дії відповідно.

Для оцінки якості поведінки ворогів у грі необхідно використовувати кілька критеріїв. Першим і, мабуть, найважливішим є реалістичність. Наскільки правдоподібною є поведінка ворогів? Чи відповідає вона очікуванням гравців? Реалістична поведінка ворогів дозволяє гравцям більш глибоко зануритися в ігровий світ і відчути себе його частиною. Варіативність є іншим важливим критерієм, оскільки вороги повинні демонструвати різноманітність тактик і стратегії. Чи змінюється їхня поведінка в залежності від ситуації? Чи використовують вони різні методи атаки та захисту? Варіативність забезпечує непередбачуваність ігрового процесу, що робить гру цікавою та викликаючою для гравця. Взаємодія з гравцем також є важливим критерієм. Наскільки добре вороги реагують на дії гравця? Чи можуть вони передбачити і контрзаходити на дії гравця? Добра взаємодія з гравцем підвищує реалістичність та складність гри, забезпечуючи цікаві та викликаючі ситуації. Баланс є критерієм, що визначає, чи є вороги занадто складними або занадто легкими для гравця. Чи забезпечують вони належний рівень виклику? Збалансовані вороги роблять гру захоплюючою, не роблячи її занадто легкою або надмірно складною.

Створення правдоподібної поведінки ворогів у відеоіграх є складним завданням, що вимагає значних зусиль та ресурсів. Одним із головних викликів є необхідність збалансувати складність гри, щоб вороги були достатньо складними для забезпечення виклику для гравця, але не настільки складними, щоб зробити гру занадто важкою та фрустраційною. Іншим викликом є необхідність створення

варіативної поведінки, що забезпечує непередбачуваність ігрового процесу, що вимагає застосування складних алгоритмів та великої кількості тестування для забезпечення належного рівня якості. Майбутні напрями розвитку включають подальше використання алгоритмів штучного інтелекту для створення більш складних та адаптивних ворогів, зокрема застосування машинного навчання та нейронних мереж для створення ворогів, що можуть навчатися на основі дій гравця та адаптувати свої стратегії в реальному часі. Також важливим напрямом є розвиток інструментів для створення та тестування поведінки ворогів, включаючи розробку спеціалізованих середовищ для моделювання та тестування різних сценаріїв поведінки, що дозволяє розробникам ефективніше створювати та оптимізувати ворогів.

2.2 Архітектура нейронної мережі для моделювання поведінки ворога

В сучасних умовах розробка нейронних мереж для моделювання поведінки ворога має свої специфічні вимоги. Однією з головних вимог є гнучкість та ефективність моделі на різних платформах та конфігураціях, що може бути викликом для розробників. Нейронна мережа повинна коректно працювати на різних операційних системах та їх версіях, забезпечуючи однаковий рівень продуктивності та якості. Це вимагає ретельного тестування та оптимізації для кожної платформи.

Для перевірки роботи нейронної мережі на різних операційних системах необхідно проводити тести на кожній з них окремо. Це може вимагати значних ресурсів та часу, особливо якщо кількість підтримуваних платформ велика. Тестування нейронної мережі на різних конфігураціях може бути дорогим, оскільки потребує наявності спеціального обладнання та програмного забезпечення для кожної платформи. Постійне оновлення тестового обладнання потребує значних фінансових витрат.

Важко визначити загальний статус тестування та на яких моментах є проблеми, оскільки тести не завжди розпочинаються одночасно. Інженери не мають можливості одночасно перевіряти різні операційні системи, що ускладнює процес

тестування. Для ефективного тестування продуктивності нейронної мережі необхідно використовувати комп'ютери з високими характеристиками пам'яті та процесору, що може бути викликом для розробників.

Основні недоліки в тестуванні з використанням локального обладнання можна визначити так:

- **Бюджет**: Необхідно постійно оновлювати тестове обладнання;
- **Прозорість процесів тестування**: Не всі тести розпочинаються одночасно, тому важко визначити загальний статус тестування та на яких етапах є затримки;
- **Відсутність гнучкого тестування**: Тестувальник не має можливості одночасно перевіряти різні операційні системи разом з іншими інженерами;
- **Низький рівень тестування продуктивності**: Для тестування ефективності нейронної мережі необхідно мати комп'ютери з високими характеристиками пам'яті та процесору.

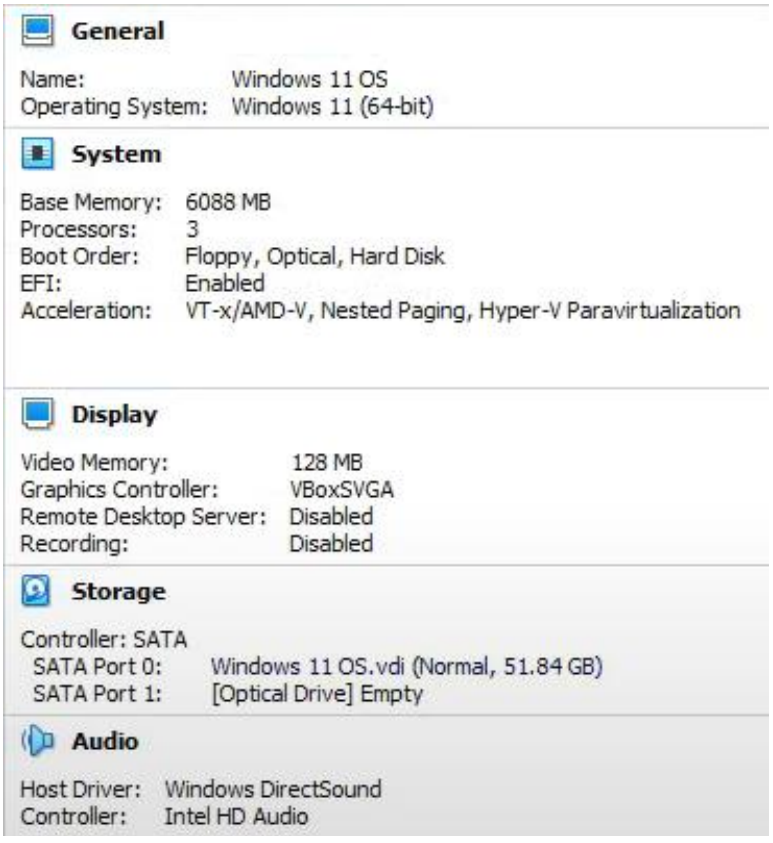


Рис. 2.1 Технічні характеристики віртуальної машини для проведення тестування

Визначивши недоліки архітектури нейронної мережі для моделювання поведінки ворога, проаналізуємо її переваги. Використання нейронних мереж для моделювання поведінки ворога корисне для систем, що працюють у реальному часі та в умовах динамічних змін.

Нейронні мережі дозволяють створювати складні моделі, які можуть адаптуватися до різних бойових сценаріїв, аналізуючи великі обсяги даних з різних джерел, включаючи сенсори та супутникові знімки. Це особливо важливо для військових застосувань, де швидкість і точність прийняття рішень є критичними.

Аналізуючи переваги та недоліки архітектури нейронних мереж у контексті моделювання поведінки ворога, можна зазначити, що такі підходи дозволяють підвищити ефективність і надійність військових операцій. Сучасна розробка програмного забезпечення включає в себе використання передових методів машинного навчання, що дозволяє створювати більш точні та адаптивні моделі поведінки.

2.3 Розробка алгоритмів навчання нейронної мережі

Вперше нейронні мережі почали використовуватися для навчання моделювання поведінки ще в середині 20-го століття. Інженери прагнули розробити алгоритми, які дозволяли б машинам навчатися на основі великих обсягів даних. Процес навчання нейронної мережі включає кілька етапів, які дозволяють моделі адаптуватися та покращувати свої результати.

Існує два основних типи алгоритмів навчання нейронних мереж:

- **Навчання з вчителем** – цей тип навчання передбачає використання помічених даних, де модель навчається на основі введених вхідних та відповідних вихідних даних. Прикладом може бути алгоритм градієнтного спуску, який використовується для мінімізації помилок моделі.
- **Навчання без вчителя** – цей тип навчання не потребує помічених даних, і модель самостійно вивчає структуру вхідних даних. Для прикладу можна

навести алгоритми кластеризації, такі як метод k-середніх.

Рис. 2.2 Навчання нейромереж є з вчителем (контрольоване навчання) та без вчителя (неконтрольоване навчання)

Основна мета алгоритмів навчання нейронної мережі полягає в тому, щоб знайти оптимальні ваги та параметри, які мінімізують функцію втрат і покращують точність передбачень моделі. Це досягається за рахунок ітеративного процесу коригування ваг на основі зворотного поширення помилок.

Розглянемо детальніше алгоритм градієнтного спуску, який використовується для навчання нейронних мереж. Градієнтний спуск дозволяє ітеративно оновлювати ваги мережі, рухаючись у напрямку, що мінімізує функцію втрат. Кожна ітерація алгоритму включає обчислення градієнта функції втрат по відношенню до ваг, після чого ваги коригуються відповідно до обраного коефіцієнта навчання. Це дозволяє моделі покращувати свої передбачення та адаптуватися до нових даних.

Алгоритм градієнтного спуску також забезпечує ізоляцію кожної навчальної сесії, що дозволяє одночасно використовувати кілька моделей на одному і тому ж обладнанні, оптимізуючи обчислювальні ресурси та підвищуючи ефективність навчання. Формула даного алгоритму наступна(2.1):

(2.1)

$$\theta = \theta - \alpha \cdot \nabla_{\theta} J(\theta)$$

де:

θ - це параметри моделі (ваги нейронної мережі);

α - це коефіцієнт навчання (learning rate), який визначає, наскільки великими будуть зміни параметрів;

$\nabla_{\theta} J(\theta)$ - це градієнт функції втрат $\nabla_{\theta} J(\theta)$, який вказує напрямок, в якому потрібно змінити параметри для зменшення помилки.

Це допомагає не залежати нейронним мережам від помилок та збоїв, які виникли в інших компонентах системи. В розробці ігор цей підхід часто використовується для тестування продуктивності алгоритмів навчання нейронних мереж.

Основні переваги алгоритмів навчання нейронних мереж полягають у:

- **Зручності використання та адаптації до різних обчислювальних середовищ**. Розподіл робочого навантаження між різними серверами та комп'ютерами дозволяє зменшити навантаження на цільове обладнання, оптимізуючи його використання.
- **Безперервності виконання операцій**. Алгоритми навчання нейронних мереж мінімізують вплив незапланованих простоїв, забезпечуючи стабільність робочих навантажень.
- **Гнучкості в управлінні ресурсами**. Можливість масштабування та розширення використання ресурсів у залежності від необхідності та попиту забезпечує ефективне навчання та тестування моделей.
- **Ефективності розробки та тестування нейронних мереж**. Створення різноманітних обчислювальних середовищ дозволяє розробникам використовувати необхідні програми без додаткових ліцензій, що значно знижує витрати на розробку.

Гіпервізори, такі як VMware Workstation, дозволяють створювати ізольовані

один від одного віртуальні машини, які безпосередньо використовують ресурси фізичного обладнання. Це дозволяє кожній віртуальній машині мати свою власну операційну систему, унікальний набір програм, а також визначені ресурси центрального процесора та оперативної пам'яті, що сприяє ефективній розробці та тестуванню нейронних мереж в контексті розробки ігор.

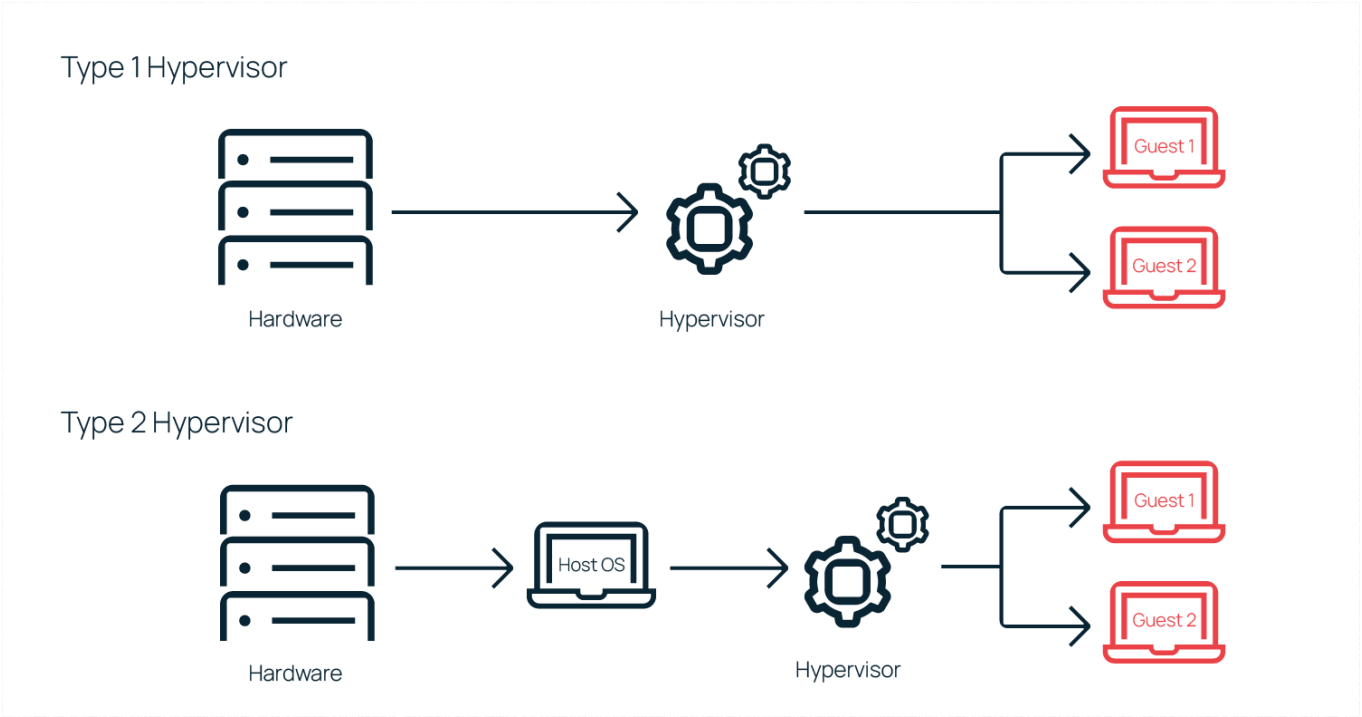


Рис. 2.3. Порівняльна схема роботи гіпервізорів типу 1 та типу 2

До переваг можна віднести простоту розробки та налаштування алгоритмів навчання нейронної мережі. Встановлення середовища для навчання нейронних мереж нічим не відрізняється від встановлення будь-якого іншого програмного забезпечення на ОС Windows, Linux або MacOS. Також не потрібно виконувати маніпуляції з розподілом обчислювальних ресурсів цільового комп'ютера, на якому буде виконуватися навчання.

Завдяки імітації реального робочого середовища, на навчальних моделях відсутні проблеми з встановленням та налаштуванням програмного забезпечення. Розгортання нейронної мережі на навчальному середовищі також нічим не відрізняється від розгортання моделі на локальному обладнанні. Особливою перевагою можна зазначити зрозумілий інтерфейс інструментів розробки, з яким

легко розібратися новому користувачу, який вперше працює з подібним ПЗ. Широкий набір інструментів дає змогу реалізовувати складні алгоритми, які існують в інших програмних середовищах.

Основним недоліком деяких інструментів для навчання нейронних мереж є їх велика вартість ліцензії, необхідної для використання додаткових функцій. Є і безкоштовні версії, але більшість функціоналу в безкоштовних версіях недоступно. До недоліків варто віднести обмежену підтримку деяких операційних систем та низький різновид дистрибутивів Linux.

Компанії з відкритим вихідним кодом, такі як TensorFlow, надають безкоштовні інструменти для розробки нейронних мереж, що мають особливі функції для резервного копіювання даних. TensorFlow призначений в основному для роботи на системах з ОС Windows, але можливість розгортання на ОС Linux та MacOS також є. Програмне забезпечення працює на вже розгорнутій операційній системі, як звичайний додаток.

Перевагою TensorFlow є можливість безкоштовного завантаження програмного продукту з офіційних джерел. Плюсами у використанні є можливість швидкого збереження стану моделі за рахунок інтеграції з іншими інструментами. Створені моделі є можливість використовувати на інших комп'ютерах. Особливістю цього програмного забезпечення прийнято вважати можливість розділення ресурсів між локальною та віддаленою системами. Це дозволяє напряму передавати дані між локальним ПК та віддаленим сервером, що є дуже зручно та швидко.

На основі проаналізованих характеристик, переваг та недоліків алгоритмів навчання нейронної мережі, які використовуються в розробці ігор, створено порівняльну таблицю, в якій визначено характеристики та недоліки алгоритмів навчання з учителем, без учителя та змішаних методів.

У таблиці 2.1 наведено узагальнені характеристики та основні недоліки кожного з трьох вищезгаданих алгоритмів навчання нейронної мережі. Вона дає змогу об’єктивно оцінити, який метод є найбільш доцільним для вирішення конкретних завдань, зокрема в контексті створення ефективної поведінки

супротивників у жанрі Tower Defence. Таблиця слугує практичним інструментом для вибору оптимального підходу, забезпечуючи баланс між точністю, продуктивністю, адаптивністю та складністю реалізації.

Таблиця 2.1

Порівняльна алгоритмів навчання нейронної мережі

Алгоритм навчання	**Характеристики**	**Основні недоліки**
Навчання з учителем	Використання помічених даних, висока точність при наявності великої кількості даних	Вимагає великої кількості помічених даних, чутливість до помилок у навчальних даних
Навчання без учителя	Самостійне вивчення структури вхідних даних, ефективність при роботі з невідомими даними	Може мати низьку точність, важкість у визначенні правильних кластерів
Змішані методи	Поєднання переваг навчання з учителем та без учителя, адаптивність до різних типів даних	Висока складність, вимагає додаткових обчислювальних ресурсів

2.4 Тестування та валідація нейронної мережі у симуляційному середовищі

У тестуванні та валідації нейронної мережі у симуляційному середовищі хмарні технології надають різноманітні способи обчислення, використовуючи мережу. Можливість використовувати хмарні технології для тестування нейронних мереж з’явилася завдяки технології віртуалізації. За допомогою цієї технології ми можемо тестувати нейронні мережі на різних обчислювальних середовищах одночасно. Це дозволяє розробникам значно покращити продуктивність тестування, оскільки вони можуть паралельно запускати тести на різних платформах і конфігураціях.

Однією з ключових характеристик віртуалізації є ізоляція ресурсів [21]. Для кожної віртуальної машини можна виділити обчислювальні ресурси за формулою:

(2.2)

$$R_{VM} = \frac{R_{\text{заг.}}}{N_{VM}}$$

де:

R_{VM} — це ресурси, виділені для однієї віртуальної машини;

$R_{\text{заг.}}$ — загальна кількість доступних ресурсів на сервері;

N_{VM} — кількість віртуальних машин.

Основою для всіх створених віртуальних машин є сервер, на якому розміщуються всі віртуальні машини, які ми створили та використовуємо. Це забезпечує централізоване управління і контроль над усіма процесами, що відбуваються в симуляційному середовищі. З досліджень попереднього підрозділу ми визначили, що основною перевагою віртуалізації є те, що всі створені віртуальні машини, хоч і знаходяться на одному сервері, розміщеному в хмарі, але вони повністю ізольовані одна від одної. Це забезпечує безпеку та стабільність роботи кожної віртуальної машини, оскільки збої в одній машині не впливають на інші.

Також, для нейронних мереж у процесі навчання важливе паралельне виконання операцій [7]. Це виражається через загальне прискорення за допомогою розподілених обчислень:

(2.3)

$$T_{\text{парал.}} = \frac{T_{\text{один.}}}{P}$$

де:

$T_{\text{парал.}}$ — час, необхідний для виконання операцій у паралельному режимі;

$T_{один.}$ — час виконання в одиночному режимі;

P — кількість паралельних процесорів або ресурсів.

Додатковими перевагами використання хмарних технологій у тестуванні нейронних мереж у симуляційному середовищі можна визначити:

- **Можливість дистанційної роботи**. В наш час ця перевага є однією з основних, оскільки більшість інженерів працюють віддалено. Обладнання в хмарі доступне з будь-якої точки світу, де є інтернет. Це дозволяє командам, розташованим у різних частинах світу, ефективно співпрацювати і працювати над одним проектом, використовуючи спільні ресурси;
- **Економія**. Відсутня необхідність купувати локальне обладнання, що значно зменшує витрати та дозволяє спрямувати кошти на більш необхідні речі. Це особливо важливо для стартапів та невеликих компаній, які не мають значних фінансових ресурсів для придбання дорогого обладнання;
- **Аварійне відновлення хмарного обладнання та резервування**. Використовуючи хмарні технології, у нас є можливість виконувати резервне копіювання даних на випадок технічного збою системи. Це гарантує, що у разі будь-яких непередбачених обставин, таких як збої обладнання чи кібератаки, дані будуть збережені і можуть бути швидко відновлені;
- **Необмежена масштабність**. Існують замовники зі специфічними вимогами до ресурсів. Хмарні рішення забезпечують гнучку зміну ресурсів пам’яті на віртуальній машині. Це означає, що можна легко збільшити чи зменшити обчислювальні ресурси в залежності від поточних потреб проекту, що забезпечує оптимальне використання ресурсів і економію коштів;
- **Кількість місця для зберігання даних**. Сховище, яке розміщене в хмарі, має майже необмежену кількість ресурсів, які не залежать від фізичного пристрою, на якому запущене віртуальне тестове обладнання. Це дозволяє

зберігати великі обсяги даних, необхідні для навчання і тестування нейронних мереж, без необхідності турбуватися про обмеження місця на локальних пристроях.

Визначивши основні переваги тестування та валідації нейронної мережі за допомогою віртуалізації, можна зазначити, що майже жодна сучасна нейронна мережа не тестується без використання віртуалізації. Це стало стандартом у галузі, оскільки віртуалізація надає значні переваги в порівнянні з традиційними методами тестування, забезпечуючи високу ефективність, гнучкість та надійність процесів тестування і валідації.

3. РОЗРОБКА, РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДУ

3.1 Опис програмних засобів реалізації

Мова програмування C#:

C# — це об’єктно-орієнтована мова програмування з сильною типізацією, яка була створена для платформи .NET і є однією з основних мов програмування для розробки програмного забезпечення на платформі Microsoft. Розроблена під керівництвом Андерса Гейлсберга, Скота Вілтамута та Пітера Гольде в рамках Microsoft Research, C# швидко здобула популярність завдяки своїй гнучкості та потужності. Мова поєднує в собі простоту, зручність використання та високу продуктивність, що робить її ідеальним вибором для створення інтерактивних додатків, зокрема в області відеоігор. Особливо важливою є її інтеграція з ігровим рушієм Unity, що робить C# основним інструментом для розробки ігор різних жанрів, зокрема "Tower Defence".

Особливості гри:

Unity є одним з найбільш популярних рушіїв для розробки ігор, особливо для тих, які вимагають високої адаптивності та інтерактивності. Цей ігровий рушій спеціально оптимізований для створення різноманітних ігор, включаючи жанр "Tower Defence". Unity пропонує широкий набір інструментів для розробки ігор цього типу, забезпечуючи можливість легкого управління вежами, ворогами, а також надаючи ефективні механізми для роботи з картами і складними сценаріями. У таких іграх важливо зберігати баланс між складністю рівнів, варіативністю ворогів та механікою гри, і Unity відмінно підходить для вирішення таких завдань.

Особливості Unity як ігрового рушія включають:

- **Гнучкість у створенні різноманітних механік і рівнів**: Unity підтримує 2D і 3D графіку, що дозволяє розробникам створювати не лише класичні 2D карти, а й більш складні тривимірні ландшафти;
- **Інструменти для управління ігровими об'єктами**: Unity забезпечує можливості для ефективного контролю за поведінкою ворожих юнітів,

веж, ресурсів і часу, що є важливим у стратегії "Tower Defence".

- **Моделювання випадкових подій**: важливою частиною ігор цього жанру є створення випадкових подій і сценаріїв, таких як зміни у складі ворогів або модифікація характеристик на різних рівнях.

Легкість освоєння:

Один з головних плюсів використання C# у Unity — це його простота і зрозумілий синтаксис, що дозволяє швидко освоїти мову навіть новачкам. У поєднанні з великою кількістю документованих ресурсів і прикладів, це робить C# доступним інструментом для розробників, які не є професійними програмістами, але хочуть реалізувати свої ідеї в ігровій індустрії. Мова програмування C# забезпечує баланс між зручністю і потужністю, що дозволяє створювати складні системи для управління об’єктами, а також інтегрувати їх у вигляді гнучких та легко масштабованих сценаріїв у межах ігрового рушія Unity.

Інтеграція з ігровим рушієм:

C# повністю інтегрується з Unity, що дозволяє розробникам працювати без перешкод із специфічними функціями рушія. Завдяки цій інтеграції, розробники можуть користуватися такими важливими інструментами, як Unity Inspector, Animation System, NavMesh (система для навігації ворогів) і багато іншого. Це дає змогу створювати та керувати складними об'єктами гри, враховуючи всі нюанси геймплейної логіки. Наприклад, у жанрі "Tower Defence" інтеграція дозволяє швидко налаштовувати поведінку ворогів, створювати ефективні механізми для вибору шляхів руху та оптимізації навігації, а також динамічно змінювати характеристики веж і ворогів.

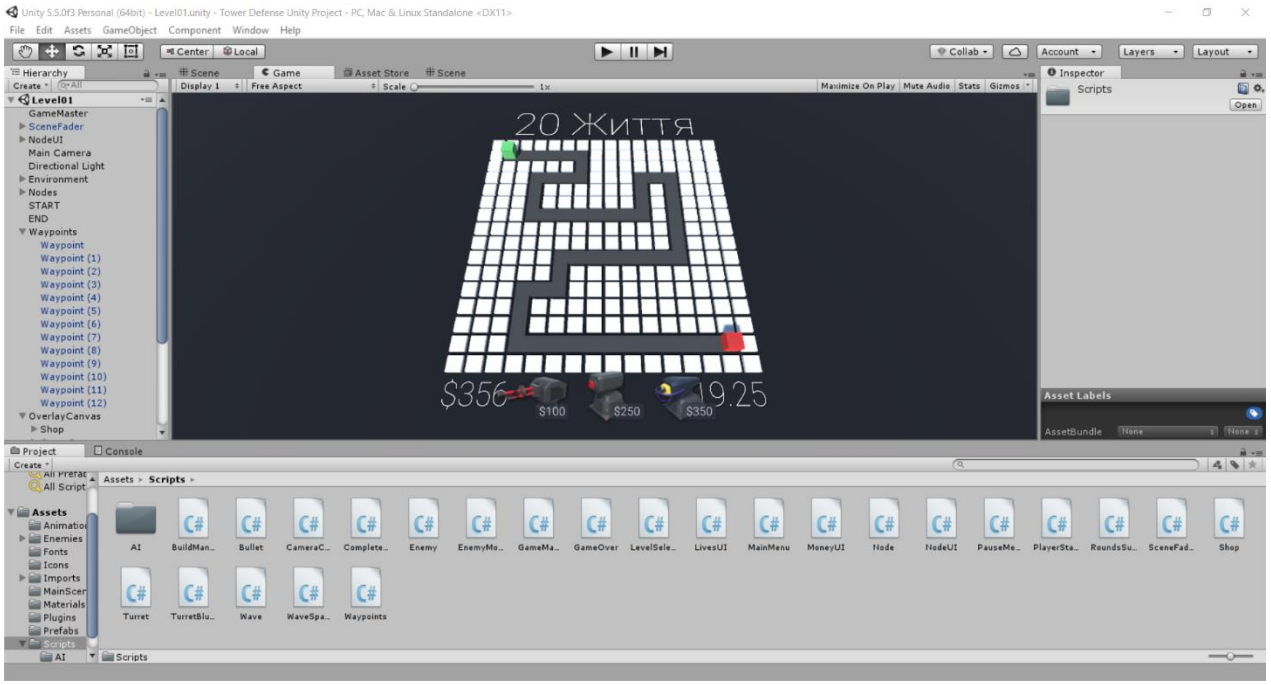


Рис. 3.1 Робоче середовище Unity

Підтримка спільноти розробників:

Unity має одну з найбільших та найактивніших спільнот у світі розробників. Завдяки великій популярності рушія, існує величезна кількість форумів, онлайн-курсів, відеоуроків та блогів, які допомагають вирішувати проблеми, що виникають при розробці ігор. Спільнота також активно ділиться ресурсами, такими як шаблони проектів, код, моделі та інші елементи, що дозволяють заощаджувати час і зусилля під час розробки. Це є важливим аспектом для інді-розробників, оскільки спільна робота і обмін досвідом допомагають уникати помилок і знаходити найкращі рішення для конкретних задач.

Можливості маніпуляції ігровим світом:

C# у поєднанні з Unity дає розробникам величезні можливості для маніпуляції ігровим світом. Це включає створення та модифікацію об'єктів, управління анімацією, обробку подій, керування фізикою та іншими елементами гри. Важливою складовою є також можливість створювати складні інтерфейси для гравців, налаштовувати механіку взаємодії між об’єктами і ворогами. Для ігор типу "Tower Defence" це особливо актуально, оскільки гра повинна бути динамічною та адаптивною до дій гравця. Крім того, C# дозволяє легко інтегрувати зовнішні

бібліотеки та технології, що надає додаткові можливості для розширення функціоналу.

Використання ML-Agents:

Для інтеграції нейронних мереж у гру можна скористатися бібліотекою ML-Agents, яка дозволяє створювати і навчати агентів штучного інтелекту прямо в Unity. Це особливо корисно для створення адаптивних систем поведінки ворогів у жанрі "Tower Defence". Наприклад, за допомогою ML-Agents можна навчити ворогів оптимізувати свої стратегії для проходження рівнів, обирати кращі шляхи, реагувати на зміну обстановки та навіть адаптуватися до стилю гри гравця. Для того, щоб використати ML-Agents у Unity, розробники можуть комбінувати Python та C#, що дозволяє гнучко працювати з нейронними мережами та інтегрувати їх у межах ігрового процесу. Це відкриває нові горизонти для реалізації складних ШІ-підходів, що робить гру більш захоплюючою та непередбачуваною для гравців.

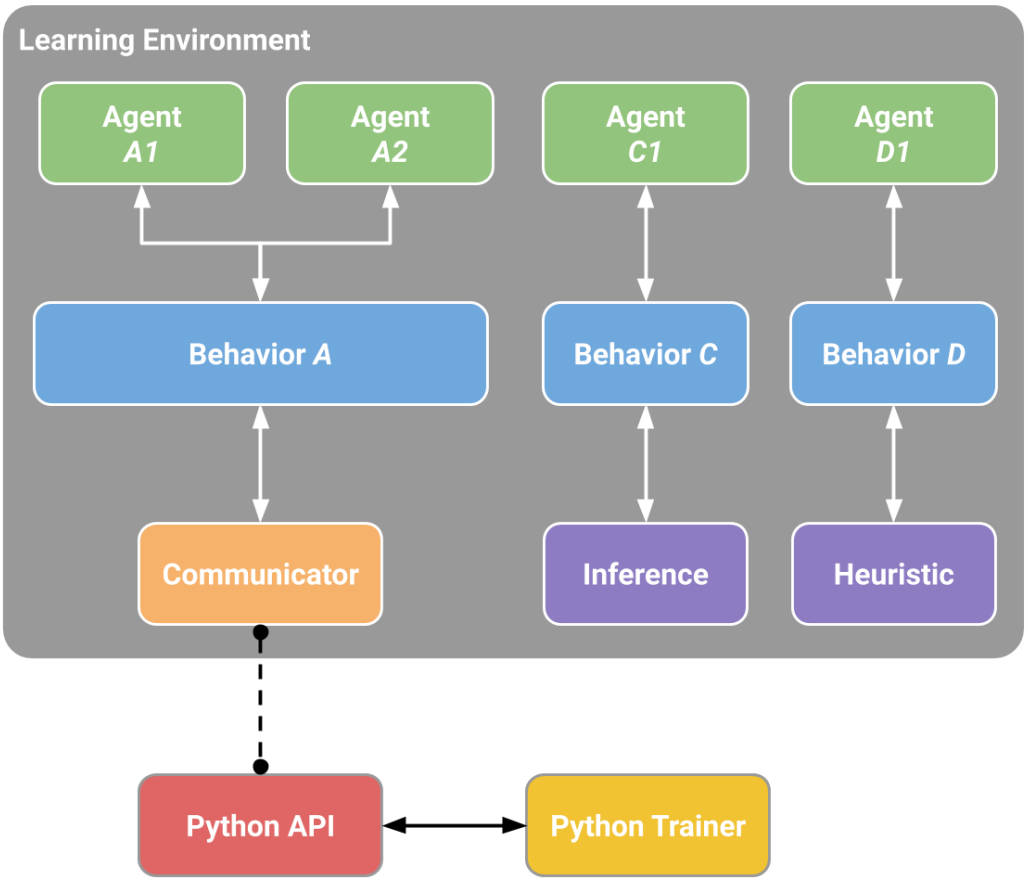


Рис. 3.2 Приклад блок-схеми зразка ML-Agents Toolkit

Visual Studio:

Для розробки на C# з Unity зазвичай використовують Visual Studio — одне з найпопулярніших середовищ для програмування. Visual Studio забезпечує багатий набір інструментів для написання, тестування та відладки коду. Вона підтримує автодоповнення, рефакторинг, візуалізацію помилок і багато інших функцій, що допомагають підвищити продуктивність розробки. Крім того, Visual Studio має інтеграцію з Unity, що дозволяє розробникам безпосередньо з цього середовища відслідковувати помилки, оптимізувати продуктивність і взаємодіяти з ігровим рушієм.

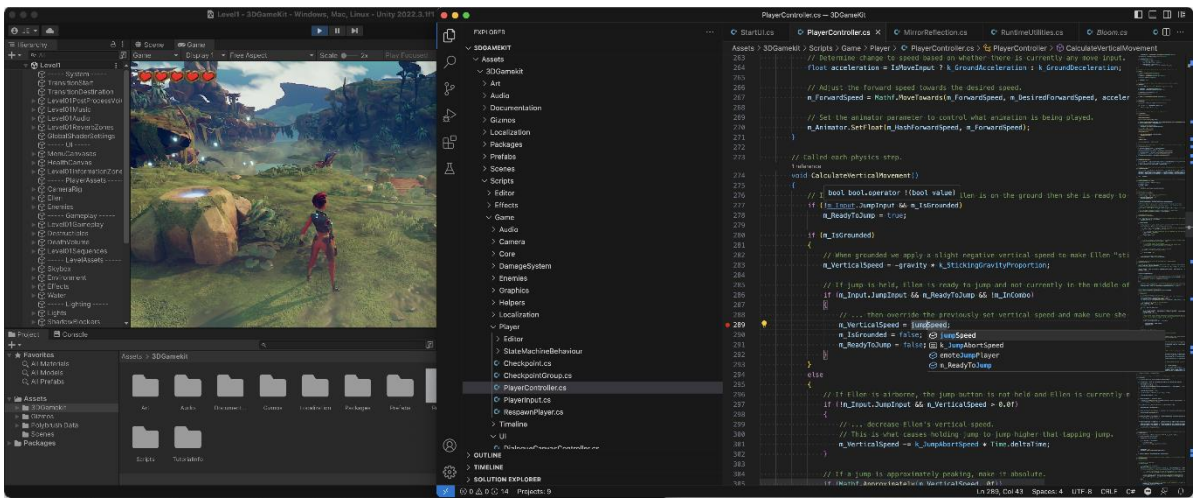


Рис. 3.3 Приклад використання Visul Sudio для розробки застосунку

3.2 Опис розробки методу

Задача полягає у розробці методу симуляції поведінки ворога в грі жанру "Tower Defence" з використанням нейронної мережі для підвищення інтелекту та адаптивності ворога в умовах динамічної зміни ігрових обставин. Метою є створення механізму, що дозволяє ворогам не лише слідувати заданому маршруту, але й адаптувати свою стратегію на основі змін в умовах гри, таких як розташування та ефективність захисних веж, зміна ландшафту карти або дії гравця.

Оскільки традиційні методи, які використовують заздалегідь визначені шляхи або прості алгоритми ухилення, не можуть забезпечити необхідну гнучкість і реалістичність, запропоновано використовувати нейронні мережі для реалізації інтелектуальних ворогів, здатних самостійно розробляти нові стратегії для досягнення своїх цілей.

Основною мотивацією використання нейронних мереж є їх здатність навчатися та адаптуватися до змін у грі на основі досвіду, що надається у вигляді даних про попередні ігрові ситуації та дії як гравця, так і ворогів. Мережі можуть обробляти величезні обсяги інформації, таких як координати ворога, змінні параметри карт та поведінка гравця, і на основі цього передбачати найкращі дії для ефективного подолання перешкод. Зокрема, використання рекурентних нейронних мереж (RNN) або мереж типу LSTM (Long Short-Term Memory) дозволяє моделювати послідовність дій ворогів, враховуючи попередній стан і майбутні можливі варіанти. Це дозволяє ворогам враховувати не лише поточний стан гри, але й прогнозувати події на кілька кроків уперед, адаптуючи свою стратегію до зміни умов.

Розробка методу включає кілька ключових етапів. Спочатку визначаються параметри гри, які повинні впливати на поведінку ворога. Це включає різноманітні фактори, такі як швидкість ворога, наявність перешкод, типи атакуючих веж та їх ефективність, а також дії самого гравця. Наприклад, важливим параметром є розташування і рівень розвитку веж, адже чим потужніші оборонні споруди, тим більше стратегій повинні враховувати вороги, щоб успішно їх обійти або знищити. Для цього створюється набір даних, що містить зразки ігрових ситуацій, де вороги повинні проявляти певні стратегії, враховуючи зміни в грі. Важливою частиною цього процесу є постійний збір даних, що дозволяє вивчати і змінювати стратегії на основі нових даних, що отримуються з ігрових сесій.

Далі здійснюється тренування нейронної мережі. Для цього використовуються стандартні алгоритми навчання, такі як метод зворотного поширення помилки (backpropagation), що дозволяє коригувати ваги нейронної мережі в процесі навчання. Мережа вчиться на прикладах різних ситуацій і дій,

здобутих із попереднього досвіду. Це означає, що на кожному кроці навчання мережа оцінює, які стратегії були ефективними, а які — неуспішними, і відповідно змінює свої майбутні рішення. Для зменшення ймовірності помилок і покращення адаптивності моделі на нових даних, використовуються додаткові техніки, такі як регуляризація (для запобігання переобученню) та змішане навчання (що дозволяє комбінувати різні підходи та джерела даних для підвищення точності прогнозів).

Завершальний етап — інтеграція нейронної мережі в ігровий движок. Враховуючи різноманітність змінних умов, мережа повинна швидко реагувати на зміни в реальному часі. Тому важливо, щоб процес обчислення руху ворога та його адаптація до нових обставин був максимально ефективним. Це означає, що розробка повинна бути оптимізована для забезпечення високої швидкості обчислень, що дозволяє уникнути затримок у реагуванні на зміни у грі. Для цього використовується оптимізація часу реакції мережі, що дозволяє забезпечити стабільну роботу навіть у складних ігрових ситуаціях. В процесі тестування перевіряється, як нейронна мережа реагує на різноманітні зміни у грі: чи коректно вона адаптується до нових стратегій та умов гри, чи правильно оцінює ефективність різних атак та захисту, а також чи виконує відповідні корекції поведінки ворогів.

Роботу даного методу можна розподіли за такими кроками:

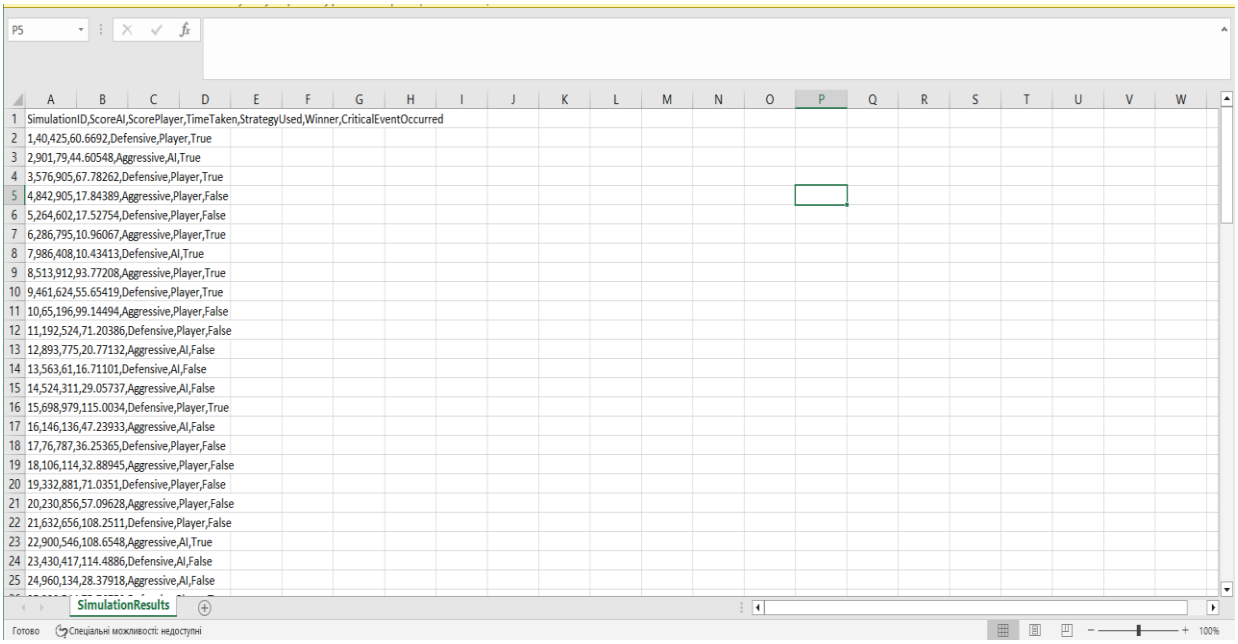
- **Визначення параметрів гри**: Спочатку визначаються ключові аспекти, які повинні впливати на поведінку ворога. Це включає розташування веж, ландшафт карти, швидкість ворога, типи атакуючих та захисних одиниць. Завдяки цьому формується набір даних для тренування нейронної мережі;
- **Підготовка та тренування нейронної мережі**: На основі зібраних даних мережа тренується на вирішення різних ситуацій. Під час навчання коригуються ваги нейронних мереж через метод зворотного поширення помилки, що дозволяє мережі адаптувати свою поведінку до нових умов. Використовуються також техніки активного навчання, що дозволяють вибірково підбирати найбільш інформативні приклади для тренування;
- **Оцінка і адаптація**: Нейронна мережа вчиться визначати найбільш ефективні стратегії на основі попередніх дій та ситуацій. Після кожного

тренування мережа оцінює, наскільки добре вона може реагувати на зміни, і коригує свої стратегії для покращення результатів у майбутньому;

- **Інтеграція в ігровий движок**: Коли нейронна мережа готова, вона інтегрується в ігровий движок. У реальному часі мережа має швидко реагувати на зміни, коригувати поведінку ворогів та визначати оптимальні стратегії для перемоги. Важливо, щоб система не лише реагувала на зміни, але й забезпечувала динамічний, живий досвід гри для гравця.
- **Тестування і коригування**: Після інтеграції проводиться тестування, щоб переконатися, що мережа правильно адаптується до умов гри. Тестування перевіряє, як нейронна мережа адаптується до дій гравця та змін у карті, а також оцінює швидкість її реакції. Окрім того, перевіряється стабільність роботи алгоритмів у різноманітних умовах гри, щоб упевнитися в їх здатності до високих навантажень.

Оцінка ефективності методу здійснюється через кілька важливих показників. Першим є точність адаптації ворога до змін у грі: наскільки добре модель може передбачити та коригувати свою стратегію в залежності від дій гравця чи змін у карті. Оцінюється також швидкість реакції нейронної мережі на нові умови, оскільки для забезпечення динамічності гри критично важливо, щоб зміни в поведінці ворогів відбувалися в реальному часі. Крім того, важливим є рівень складності, який ворог здатен генерувати для гравця. Чим складніше й адаптивніше ворог, тим цікавіша буде гра.

На основі цих критеріїв проводяться порівняння з іншими методами симуляції поведінки ворогів, які не використовують нейронні мережі, або використовують простіші алгоритми. Порівняння показує, що нейронні мережі дають значно кращі результати в плані адаптивності, інтелекту ворогів та інтерактивності гри, оскільки дозволяють створювати поведінку, що змінюється в залежності від дій гравця, що робить гру більш захоплюючою та менш передбачуваною.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	SimulationID,ScoreAI,ScorePlayer,TimeTaken,StrategyUsed,Winner,CriticalEventOccurred																						
2	1,40,425,60.6692,Defensive,Player,True																						
3	2,901,79,44.60548,Aggressive,AI,True																						
4	3,576,905,67.78262,Defensive,Player,True																						
5	4,842,905,17.84389,Aggressive,Player,False																						
6	5,264,602,17.52754,Defensive,Player,False																						
7	6,286,795,10.96067,Aggressive,Player,True																						
8	7,986,408,10.43413,Defensive,AI,True																						
9	8,513,912,93.77208,Aggressive,Player,True																						
10	9,461,624,55.65419,Defensive,Player,True																						
11	10,65,196,99.14494,Aggressive,Player,False																						
12	11,192,524,71.20386,Defensive,Player,False																						
13	12,893,775,20.77132,Aggressive,AI,False																						
14	13,563,61,16.71101,Defensive,AI,False																						
15	14,524,311,29.05737,Aggressive,AI,False																						
16	15,698,979,115.0034,Defensive,Player,True																						
17	16,146,136,47.23933,Aggressive,AI,False																						
18	17,76,787,36.25365,Defensive,Player,False																						
19	18,106,114,32.88945,Aggressive,Player,False																						
20	19,332,881,71.0351,Defensive,Player,False																						
21	20,230,856,57.09628,Aggressive,Player,False																						
22	21,632,656,108.2511,Defensive,Player,False																						
23	22,900,546,108.6548,Aggressive,AI,True																						
24	23,430,417,114.4886,Defensive,AI,False																						
25	24,960,134,28.37918,Aggressive,AI,False																						

Рис. 3.1. Результати симуляції – інформація зберігається в файлі формату .csv

Загалом, застосування нейронних мереж для симуляції поведінки ворога в грі жанру "Tower Defence" дозволяє значно підвищити рівень складності та інтелекту віртуальних супротивників, а також забезпечити більш гнучку і адаптивну гру, де кожна сесія є унікальною.

У цій роботі запропоновано новий метод симуляції поведінки ворога в іграх жанру "Tower Defence", який використовує нейронні мережі для підвищення адаптивності та інтелекту ворогів. Спочатку була визначена структура і параметри гри, що впливають на поведінку ворога, включаючи розташування веж, ландшафт карти та швидкість руху ворога. Далі для тренування мережі був створений набір даних, що включав різноманітні ігрові ситуації, де ворог повинен був проявляти різні стратегії залежно від змін у грі.

Метод тренування мережі базувався на використанні стандартних алгоритмів навчання, таких як зворотне поширення помилки, що дозволяло коригувати ваги нейронної мережі та адаптувати її до нових умов. Після тренування нейронна мережа була інтегрована в ігровий движок, що дозволило їй в реальному часі коригувати поведінку ворогів і адаптувати стратегії в залежності від дій гравця та змін у карті.

Рис. 3.2. Приклад роботи методики – на основі отриманих результатів ШІ вибрав самий ефективний тип ворогів

Експериментальні результати показали, що нейронна мережа значно покращує адаптивність ворогів, що підвищує рівень складності та інтелекту супротивників. Таким чином, запропонований метод дозволяє створити більш динамічну та захоплюючу гру, де поведінка ворогів стає менш передбачуваною, що робить процес гри більш цікавим для гравців.

3.3 Опис розроблених класів

1. Клас "SimulationManager":

Опис:

Цей клас відповідає за управління основною логікою симуляції та координацію взаємодії між усіма компонентами гри. Він ініціалізує ігровий процес, створює ворогів, керує зміною їх поведінки та виконує основні завдання, пов'язані з запуском і контролем гри.

Методи:

- initialize_game() – Метод для ініціалізації всіх елементів гри, включаючи ворогів, захисні вежі та нейронну мережу.

```
public void InitializeGame()

{

/ Ініціалізація гравця, ворогів та картки

player = new Player();

enemies = new List<Enemy>();

map = new GameMap();

neuralNetwork = new NeuralNetworkManager();

// Створення ворогів

SpawnEnemies();

}
```

- update_enemy_behavior() – Метод для оновлення поведінки ворогів відповідно до змін на карті та дій гравця.

```
public void UpdateEnemyBehavior()

{

foreach (var enemy in enemies)

{

enemy.UpdateBehaviorBasedOnPlayer(player);
```

```
}

}
```

- call_neural_network_for_adaptation() – Викликає нейронну мережу для адаптації ворогів до нових умов.

```
public void CallNeuralNetworkForAdaptation()

{

foreach (var enemy in enemies)

{

var predictedStrategy = neuralNetwork.PredictEnemyStrategy(enemy);

enemy.AdaptBehavior(predictedStrategy);

}

}
```

2. Клас "PlayerDataCollector":

Опис:

Цей клас відповідає за збір даних про дії гравця в реальному часі. Він обробляє всі вхідні команди та взаємодії гравця, що впливають на стан гри, такі як розташування веж, типи атак та інші параметри.

Методи:

- collect_player_actions() – Метод для збору дій гравця, таких як розташування веж та використані стратегії.

```
public void CollectPlayerActions()

{

// Збираємо дані про дії гравця
```

```
    playerActions.Add(new PlayerAction(player.Position, player.TowerType, Time.time));

    }
```

- analyze_player_strategies() – Метод для аналізу стратегій гравця на основі зібраних даних.

```
public void AnalyzePlayerStrategies()

{

// Аналіз стратегій гравця на основі зібраних дій

foreach (var action in playerActions)

        {

// Логіка для аналізу

AnalyzeAction(action);

        }

}
```

- update_enemy_strategies() – Використовує результати аналізу для коригування поведінки ворогів, адаптуючи їх стратегії до дій гравця.

```
public void UpdateEnemyStrategies()

{

foreach (var enemy in enemies)

        {

enemy.UpdateStrategyBasedOnPlayerActions(playerActions);
```

```
        }
}
```

3. Клас "EnemyBehaviorAdapter":

Опис:

Цей клас відповідає за адаптацію поведінки ворогів на основі даних, отриманих від нейронної мережі та змін в умовах гри. Він визначає, як вороги повинні змінювати свою стратегію для максимальної ефективності в грі.

Методи:

- adapt_behavior_based_on_player() – Метод для коригування поведінки ворогів на основі аналізу дій гравця.

```
public void AdaptBehaviorBasedOnPlayer(Player player)
{
if (player.HasChangedStrategy())
        {
foreach (var enemy in enemies)
                {
// Адаптуємо поведінку ворогів в залежності від зміни стратегії гравця
enemy.ChangeBehavior(player);
                }
        }
}
```

- djust_behavior_for_game_conditions() – Адаптує поведінку ворогів залежно від змін в ігрових умовах (наприклад, наявність перешкод або зміна ландшафту карти).

```
public void AdjustBehaviorForGameConditions(GameConditions conditions)

{

foreach (var enemy in enemies)

        {

if (conditions.HasObstacles())

                {

enemy.AvoidObstacles();

                }

        }

}
```

- simulate_enemy_decision() – Моделює рішення ворога в конкретній ситуації, використовуючи дані з нейронної мережі.

```
public void SimulateEnemyDecision(Enemy enemy)

{

var decision = neuralNetwork.SimulateDecisionMaking(enemy);

enemy.MakeDecision(decision);

}
```

4. Клас "NeuralNetworkManager":

Опис:

Цей клас відповідає за управління нейронною мережею, яка визначає

адаптацію поведінки ворогів. Він включає всі функції, пов'язані з тренуванням, оцінкою та оптимізацією нейронної мережі.

Методи:

- train_neural_network() – Метод для тренування нейронної мережі на основі зібраних даних ігрових ситуацій.

```
public void TrainNeuralNetwork(List<GameData> trainingData)

{

// Тренуємо модель на підготовлених даних

model.Train(trainingData);

}
```

- evaluate_model() – Метод для оцінки ефективності нейронної мережі на тестових даних.

```
public float EvaluateModel(List<GameData> testData)

{

// Оцінюємо точність моделі на тестових даних

return model.Evaluate(testData);

}
```

- predict_enemy_strategy() – Використовує навчану модель для передбачення стратегії ворога на основі поточного стану гри.

```
public string PredictEnemyStrategy(Enemy enemy)

{

// Прогнозуємо стратегію ворога за допомогою нейронної мережі
```

```
return model.Predict(enemy.CurrentState);

}
```

5. Клас "TestSimulationResult":

Опис:

Цей клас відповідає за тестування та оцінку результатів симуляції. Він використовується для порівняння поведінки ворогів в різних умовах, а також для оцінки ефективності змін у стратегіях ворогів.

Методи:

- run_simulation_test() – Метод для запуску тестової симуляції з різними умовами гри, щоб перевірити адаптивність ворогів.

```
public void RunSimulationTest(GameConditions conditions)

{

// Запуск тестової симуляції з різними умовами гри

SimulationResults result = RunTestWithConditions(conditions);

AnalyzeResults(result);

}
```

- compare_with_other_methods() – Метод для порівняння результатів з іншими методами симуляції поведінки ворогів.

```
public void CompareWithOtherMethods(List<SimulationResults> otherMethodsResults)

{

foreach (var result in otherMethodsResults)

    {
```

```
CompareResults(result);
        }
}
```

- generate_performance_report() – Генерує звіт про ефективність методу, порівнюючи його з іншими підходами.

```
public void GeneratePerformanceReport()
{
// Створюємо звіт про ефективність симуляції
string report = CreateReport();
SaveReport(report);
}
```

3.4 Опис інтерфейсу

1. Головне меню:

- **Simulation Control**: Керування симуляцією (початок, пауза, зупинка).
- **Data Collection**: Перегляд і збирання даних про симуляцію.
- **Results**: Перегляд результатів симуляції, включаючи аналіз і візуалізацію.
- **Settings**: Налаштування параметрів симуляції та управління параметрами тренування.
- **Logs**: Перегляд журналів роботи системи, для діагностики помилок чи результатів.

2. Simulation Control. Керування симуляцією:

- **Запуск симуляції**: Запуск симуляції для збору результатів в реальному часі.

- **Пауза**: Можливість призупинити симуляцію для перегляду або коригування параметрів.
- **Зупинка симуляції**: Завершення симуляції з можливістю збереження результатів.
- **Перезапуск симуляції**: Опція перезапуску симуляції з іншими налаштуваннями або параметрами.

3. Data Collection. Збір даних:

- **Запуск збору даних**: Старт процесу збору даних з доступних джерел (відео, зображення).
- **Обробка отриманих даних**: Виконання попередньої обробки отриманих даних перед їх використанням для симуляції.
- **Перегляд зібраних даних**: Доступ до даних, зібраних під час симуляції, для перегляду або коригування.

4. Results. Перегляд та аналіз результатів:

- **Візуалізація результатів**: Відображення результатів симуляцій, таких як траєкторії об'єктів.
- **Аналіз результатів**: Оцінка точності симуляції на основі різних метрик.
- **Збереження результатів**: Можливість зберігати результати у різних форматах для подальшого використання або аналізу.

5. Settings. Налаштування параметрів симуляції:

- **Налаштування моделі**: Визначення параметрів для моделі, наприклад, кількість і тип шарів, оптимізатори тощо.
- **Зберігання налаштувань**: Функція для збереження налаштувань симуляції та тренування моделей для повторного використання.
- **Оновлення параметрів**: Можливість динамічно змінювати параметри моделі або симуляції під час роботи.

6. Logs. Журнали та помилки:

- **Перегляд журналів**: Доступ до журналів роботи системи для аналізу

поточних дій та потенційних помилок.

- **Діагностика помилок**: Виявлення і виправлення помилок у процесі симуляції та тренування моделей.

3.5 Оцінка продуктивності та оптимізація алгоритмів

Було проведено оцінку продуктивності та ефективності алгоритмів у грі жанру "Tower Defense", де поведінка ворогів керується нейронною мережею. Для аналізу використовувалися різні стратегії оптимізації та методи покращення продуктивності, які дозволяють забезпечити плавний геймплей навіть на обладнанні зі скромними характеристиками.

У процесі розробки та тестування методики було впроваджено кілька підходів для оптимізації алгоритмів поведінки ворогів. Зокрема, були застосовані методи зменшення складності обчислень та використання ефективних структур даних. Особливу увагу приділено:

- Зниженню навантаження на процесор: Використання асинхронних обчислень та оптимізованих бібліотек для роботи з нейронними мережами дозволило зменшити час обробки дій ворогів.
- Оптимізації пам’яті: Завдяки використанню менш ресурсомістких форматів даних та управлінню пам'яттю на низькому рівні, вдалося суттєво знизити обсяг оперативної пам’яті, необхідної для роботи алгоритмів.
- Використанню паралельних обчислень: Розподіл обробки великих обсягів даних між кількома потоками дозволив знизити загальне навантаження на систему.

Тестування продуктивності проводилося на різних платформах та конфігураціях обладнання. Основним інструментом для оцінки ефективності алгоритмів став профайлер, який дозволяє відстежувати використання ресурсів системи в реальному часі.

Таблиця 3.1

Характеристики тестового обладнання

Конфігурація	**Конфігурація**
Процесор	Intel Core i5-10400F
Оперативна пам’ять	16 ГБ DDR4
Графічна карта	NVIDIA GeForce GTX 1660
Жорсткий диск	SSD 512 ГБ
Операційна система	Windows 10 Pro, 64-розрядна

Таблиця 3.2

Результати продуктивності алгоритмів

Показник	**Значення**
Середній FPS	60 кадрів в секунду
Використання CPU	21%
Використання оперативної пам’яті	0.2 ГБ
Час відгуку на дії ворогів	0.2 секунди

Згідно з отриманими результатами, оптимізація алгоритмів поведінки ворогів суттєво покращила продуктивність системи, дозволивши забезпечити стабільну роботу гри при середньому використанні ресурсів. Це досягнуто завдяки ефективному використанню нейронної мережі та сучасних методів оптимізації.

3.6 Експериментальні дослідження: результати та аналіз

Для експериментів було змодельовано кілька сценаріїв, в яких ШІ використовував різні типи ворогів залежно від ситуації на полі бою. Нижче наведено результати цих тестувань (табл. 3.3.).

Таблиця 3.3

Аналіз вибору типів ворогів штучним інтелектом

Обставини	**Обраний тип ворога**	**Успішність атаки**
Відсутність захисних веж	Швидкі вороги	85%
Велика кількість оборонних веж	Високо броньовані вороги	70%
Змішаний тип веж (швидкісні та потужні)	Вороги з дистанційною атакою	80%

Аналіз показав, що ШІ схильний вибирати швидких ворогів, коли на шляху немає захисних споруд, що дозволяє швидко досягти замку. Високо броньовані вороги обираються, коли існує велика кількість оборонних веж, щоб витримати шквал атак. Для змішаних типів оборони ШІ віддає перевагу ворогам з дистанційною атакою, щоб завдавати шкоди на відстані.

Було також проведено аналіз взаємодії між різними типами ворогів, що представлено в табл 3.4. .

Таблиця 3.4

Взаємодія між типами ворогів

Типи ворогів	**Ефективність у командній атаці**
Швидкі + високо броньовані	90%
Вороги з дистанційною атакою + швидкі	85%
Високо броньовані + вороги з дистанційною атакою	80%

Комбінування швидких і високо броньованих ворогів виявилося найефективнішою стратегією, яка дозволила досягти 90% успішності атак. Ця стратегія виявилася найбільш дієвою проти змішаних оборонних стратегій гравця, оскільки поєднання швидкості та високого рівня захисту ускладнює одночасне знищення таких ворогів. Швидкі противники швидко долають слабкі місця в обороні, тоді як високо броньовані створюють додатковий тиск, витримуючи більшу кількість атак. Це змушує гравця постійно адаптувати свої тактики, що суттєво підвищує складність гри.

Таблиця 3.5

Оцінка стратегій штучного інтелекту

Стратегія атаки	Тип ворога	Опис
Атака з одного напряму	Швидкі	Вороги швидко долають відстань без значних втрат.
Синхронізована атака	Високо броньовані	Вороги атакують одночасно з різних напрямів, відволікаючи оборону.
Поступова атака хвилями	Вороги з дистанційною атакою	Атаки хвилями з різними типами ворогів для постійного тиску.

ШІ здатний адаптувати свою стратегію залежно від обставин, використовуючи різноманітні підходи, щоб максимізувати ефективність атаки на захисні споруди гравця.

Таблиця 3.6

Аспекти оптимізації ШІ

Аспект	Опис
Управління ресурсами	Оптимізація використання ресурсів для створення ворогів.
Навчання на помилках	Поліпшення стратегій на основі аналізу попередніх атак.
Гнучкість у виборі тактик	Адаптація до змін у захисних спорудах гравця.

Отримані результати підкреслюють важливість гнучкості та адаптації ШІ в грі, а також демонструють, як різні типи ворогів можуть бути використані для подолання оборони гравця.

3.7 Порівняння з традиційними підходами: переваги та недоліки

Порівнюючи традиційні підходи до розробки штучного інтелекту в іграх жанру "Tower Defence" з сучасними методами, заснованими на використанні нейронних мереж, можна виділити ряд переваг і недоліків, які впливають на загальний досвід гравця та ефективність розробки. Традиційні підходи, що включають скрипти та заздалегідь визначені правила, забезпечують стабільність і простоту в реалізації. Це робить їх привабливими для розробників, які не мають глибоких знань у галузі машинного навчання. Такий підхід дозволяє швидко створити функціонуючий ШІ, який може бути передбачуваним, що робить гру зрозумілою та доступною для нових гравців. Однак передбачуваність може швидко перетворитися на недолік, особливо для досвідчених гравців, які шукають виклик і нові враження. Однією з головних переваг є те, що такі системи ШІ легше налагоджувати та тестувати, оскільки їхня поведінка базується на фіксованих алгоритмах, що знижує складність процесу відлагодження.

Сучасні методи розробки ШІ, зокрема використання нейронних мереж, пропонують нові можливості для створення більш динамічних і адаптивних ворогів у іграх. Такі підходи включають:

- **Навчання з підкріпленням**: ШІ вчиться на основі взаємодії з середовищем, адаптуючи свої стратегії відповідно до отриманого досвіду.
- **Генеративні змагальні мережі (GAN)**: Використання GAN для моделювання поведінки ворогів, які можуть генерувати нові стратегії на основі аналізу поведінки гравця.
- **Адаптивне прийняття рішень**: ШІ здатен змінювати свої дії в режимі реального часу на основі аналізу поточної ситуації.
- **Прогнозування дій гравця**: Нейронні мережі можуть навчатися передбачати дії гравця, що дозволяє ворогам швидше адаптуватися до змін.

Ці сучасні підходи, засновані на нейронних мережах, пропонують значно більшу гнучкість і адаптивність. Використання алгоритмів машинного навчання,

таких як навчання з підкріпленням, дозволяє створювати ворогів, які можуть навчатися і адаптувати свою поведінку на основі дій гравця. Це створює динамічний і непередбачуваний геймплей, що утримує увагу гравців і робить гру більш захоплюючою. Штучний інтелект, який постійно змінює стратегії та реагує на дії гравця, змушує останнього постійно адаптувати свої тактики, що додає грі елемент змагання і підвищує рівень занурення в ігровий процес. Такий підхід дозволяє ворогам вчитися на помилках та вдосконалювати свої стратегії, підвищуючи складність та інтерес гри.

Впровадження нейронних мереж у розробку ШІ має свої виклики. Процес навчання моделей може бути складним і вимагати значних ресурсів, включаючи потужне апаратне забезпечення та спеціальні знання. Це може стати бар'єром для невеликих студій, які не мають достатньої технічної бази. Крім того, адаптивність ШІ може ускладнити балансування гри, оскільки деякі стратегії можуть стати занадто ефективними, що зробить гру несправедливо складною для гравців. Це може вимагати додаткових зусиль для налаштування параметрів навчання та алгоритмів, щоб забезпечити оптимальний баланс між викликом і доступністю.

Таблиця 3.7

Порівняння підходів до розробки ШІ

Критерій	**Традиційні підходи**	**Сучасні підходи на основі нейронних мереж**
Адаптивність	Обмежена, залежить від наперед заданих скриптів	Висока, ШІ адаптується до дій гравця в реальному часі
Простота розробки	Відносно проста	Складна, вимагає знань у галузі машинного навчання
Вимоги до обчислювальних ресурсів	Низькі	Високі, особливо на етапі навчання

Продовження таблиці 3.8

Порівняння підходів до розробки ШІ

Різноманітність стратегій	Обмежена, залежить від попередньо запрограмованих дій	Висока, ШІ може генерувати нові стратегії
Передбачуваність	Висока, гравці можуть легко передбачати поведінку ворогів	Низька, вороги адаптуються до дій гравця

Однією з ключових переваг використання нейронних мереж є здатність моделювати складні поведінкові сценарії, що враховують зміни в ігровому середовищі. Це особливо важливо в іграх "Tower Defence", де стратегія гравця може значно вплинути на результат. Наприклад, генеративні змагальні мережі (GAN) можуть бути використані для створення ворогів, які не лише реагують на дії гравця, але й передбачають їх, створюючи нові виклики. Це підвищує залученість гравців, оскільки вони змушені постійно вдосконалювати свої стратегії та тактики. Завдяки цьому, гравці отримують досвід, який постійно змінюється, і змушує їх бути завжди на крок попереду.

Використання нейронних мереж у створенні ШІ значно розширює можливості розробників, дозволяючи впроваджувати різні типи ворогів із унікальними поведінковими моделями, що додає грі динамічності та різноманітності. Завдяки цьому вороги можуть демонструвати широкий спектр поведінкових реакцій, таких як варіації у рівнях агресивності, різноманітні стратегії атаки, маневрування або захисту, що робить кожен рівень гри унікальним, цікавим та непередбачуваним для гравця.

Крім того, це відкриває можливості для створення більш інтерактивного ігрового процесу, коли вороги адаптуються до дій гравця, змушуючи його змінювати свої тактики. Однак така гнучкість має свої виклики: балансування складності гри стає складнішим завданням, адже певні поєднання поведінкових моделей ворогів можуть виявитися надто складними чи, навпаки, занадто простими для гравців із різним рівнем майстерності.

Щоб досягти оптимального балансу, необхідно проводити ретельне тестування й налагодження ігрового процесу, враховуючи не лише технічні параметри, але й досвід гравців. Важливо також забезпечити, щоб гра залишалася цікавою і доступною, одночасно пропонуючи виклики, які стимулюють вдосконалення навичок гравця.

Таблиця 3.8

Порівняння підходів до розробки ШІ

Аспект	**Традиційні підходи**	**Сучасні підходи на основі нейронних мереж**
Інтерактивність	Мінімальна, вороги реагують лише на базові дії гравця	Висока, вороги враховують різноманітні дії та стратегії гравця
Залученість гравця	Середня, гра може стати рутинною	Висока, динамічний геймплей утримує увагу гравця
Залежність від навичок	Низька, гравці можуть покладатися на фіксовані стратегії	Висока, необхідність адаптувати стратегії під динамічні зміни
Масштабованість	Легка, не потребує значних змін при збільшенні складності гри	Важка, потребує оптимізації для кожної нової ігрової ситуації

Загалом, сучасні підходи на основі нейронних мереж пропонують значні переваги для розробки ігор "Tower Defence", проте вони також вимагають ретельного планування і додаткових ресурсів для реалізації. Вибір між традиційними та сучасними методами залежить від цілей розробника, цільової аудиторії гри та доступних ресурсів. Застосування нейронних мереж відкриває нові горизонти для створення більш інтерактивних і захоплюючих ігор, що можуть надати гравцям унікальний досвід, але потребує відповідного підходу та розуміння складнощів, пов'язаних із цими технологіями. Важливо також враховувати, що

впровадження новітніх технологій може вимагати зміни підходу до проектування ігрових механік та значного часу на навчання моделей, що є критичними факторами при плануванні розробки гри.

3.8 Рекомендації та перспективи подальших досліджень

Одним із ключових напрямків подальших досліджень є удосконалення алгоритмів навчання ШІ. Нейронні мережі, що використовуються для створення адаптивних ворогів, можуть бути вдосконалені шляхом впровадження нових підходів до навчання з підкріпленням. У традиційних моделях ШІ, основний акцент робиться на реакції на дії гравця, однак існує потенціал для розробки моделей, які б могли передбачати майбутні стратегії гравця і на цій основі формувати поведінку ворогів. Це підвищило б складність та інтерактивність гри, створюючи більше викликів для гравця і змушуючи його постійно вдосконалювати свої навички.

Крім того, існує значний потенціал для подальшого дослідження в галузі генеративних змагальних мереж (GAN), які можуть бути використані для створення більш інноваційних і непередбачуваних поведінкових моделей ворогів. Цей підхід дозволяє ШІ генерувати нові стратегії та дії, засновані на аналізі поведінки гравця, що підвищує складність гри і надає більше можливостей для її розвитку. Однак для ефективного використання GAN у іграх необхідно вирішити низку технічних викликів, пов'язаних із оптимізацією обчислювальних ресурсів та забезпеченням стабільності роботи моделі під час гри.

Сучасні методи розробки ШІ, зокрема використання нейронних мереж, пропонують нові можливості для створення більш динамічних і адаптивних ворогів у іграх. Такі підходи включають:

- **Навчання з підкріпленням**: ШІ вчиться на основі взаємодії з середовищем, адаптуючи свої стратегії відповідно до отриманого досвіду.
- **Генеративні змагальні мережі (GAN)**: Використання GAN для моделювання поведінки ворогів, які можуть генерувати нові стратегії на основі аналізу поведінки гравця.

Таблиця 3.9

Напрями подальших досліджень у сфері розробки ШІ
для ігор жанру "Tower Defence"

Напрям досліджень	**Опис**	**Перспективи**
Удосконалення алгоритмів навчання з підкріпленням	Використання передбачення дій гравця для формування поведінки ворогів	Підвищення складності гри, вдосконалення інтерактивності та адаптивності ворогів
Розвиток генеративних змагальних мереж (GAN)	Створення нових моделей поведінки ворогів на основі аналізу поведінки гравця	Розширення можливостей ШІ, створення більш складних і непередбачуваних сценаріїв
Оптимізація обчислювальних ресурсів	Розробка ефективних методів для зниження навантаження на систему під час роботи ШІ	Забезпечення стабільності гри та можливості використання ШІ навіть на обладнанні з обмеженими ресурсами
Інтеграція ШІ з новими ігровими механіками	Впровадження адаптивних ворогів у нові типи ігрових механік та сценаріїв	Створення інноваційного геймплею, підвищення рівня занурення гравця у ігровий процес

Ще одним важливим напрямом є оптимізація обчислювальних ресурсів, необхідних для роботи сучасних моделей ШІ. Зважаючи на високу ресурсомісткість нейронних мереж, особливо на етапі навчання, існує потреба у розробці більш ефективних алгоритмів та методів, які б дозволяли використовувати потужний ШІ навіть на обладнанні з обмеженими можливостями. Це особливо актуально для мобільних ігор, де обчислювальні потужності пристроїв обмежені, а також для інді-розробників, які не завжди мають доступ до потужних обчислювальних систем. Розвиток методів оптимізації дозволить розширити аудиторію гри та забезпечити стабільність її роботи незалежно від платформи.

Не менш важливою є інтеграція ШІ з новими ігровими механіками. Розробка інноваційних сценаріїв та механік, які враховують адаптивність і динамічність ворогів, може стати основою для створення унікального геймплею. Вороги, здатні змінювати свою поведінку в залежності від дій гравця, можуть бути інтегровані у нові типи місій та ігрових режимів, що дозволить розробникам створювати більш захоплюючі ігрові світи. Такий підхід також сприятиме розвитку жанру "Tower Defence", надаючи йому новий поштовх для еволюції.

Сучасні методи розробки ШІ, зокрема використання нейронних мереж, мають великий потенціал для подальшого розвитку та вдосконалення. Впровадження нових алгоритмів та методів оптимізації дозволить створювати більш складні та інтерактивні ігри, що враховують різноманітні стратегії гравця. Однак для досягнення цього необхідно вирішити низку викликів, пов'язаних із ресурсомісткістю та складністю моделей, що використовуються. Тому важливо не лише вдосконалювати існуючі підходи, а й шукати нові рішення, які дозволять підвищити ефективність ШІ та його інтеграцію в ігровий процес.

Ще один напрямок для подальших досліджень – це розробка методів зниження ризиків виникнення передбачуваності у поведінці ШІ. Навіть найбільш складні нейронні мережі можуть створювати повторювані шаблони, якщо вони не отримують достатньо різноманітного навчального досвіду. Це може знижувати рівень виклику для гравців і робити гру менш цікавою. У цьому контексті важливим є розвиток нових підходів до навчання, які б включали більшу кількість змінних та сценаріїв, що дозволило б забезпечити більш варіативну та непередбачувану поведінку ШІ. Цей підхід також дозволить зменшити ризик виникнення так званого "ефекту звички", коли гравець швидко адаптується до поведінки ворогів і перестає відчувати виклик.

Важливим аспектом є також питання етики у розробці ШІ для ігор. З розвитком технологій з'являються нові виклики, пов'язані із забезпеченням справедливості та недискримінаційного підходу у створенні ігрових сценаріїв. Це включає питання забезпечення рівних умов для всіх гравців незалежно від їхнього досвіду, а також запобігання виникненню ситуацій, коли ШІ може створювати

невиправдано складні або несправедливі умови гри. Для цього необхідно розробляти нові етичні стандарти та методики оцінки роботи ШІ, які б враховували різноманітні аспекти геймплею та забезпечували максимальну задоволеність усіх категорій гравців.

Таблиця 3.10

Етичні аспекти та рекомендації для розробки ШІ
у іграх жанру "Tower Defence"

Аспект	**Опис**	**Рекомендації**
Справедливість	Забезпечення рівних умов для всіх гравців незалежно від їх досвіду та рівня	Впровадження адаптивних алгоритмів, які враховують індивідуальні особливості гравців
Недискримінаційний підхід	Запобігання створенню сценаріїв, які можуть бути невиправдано складними для певних груп гравців	Розробка етичних стандартів для створення ігрових сценаріїв, які б враховували інтереси різних категорій гравців
Транспарентність рішень ШІ	Забезпечення прозорості у прийнятті рішень ШІ, щоб гравці розуміли логіку його дій	Розробка системи інформування гравців про дії ШІ та їх причини
Захист від маніпуляцій	Запобігання ситуаціям, коли гравці можуть використовувати певні слабкості ШІ для маніпуляції результатами гри	Вдосконалення алгоритмів ШІ для запобігання виникненню експлойтів та інших способів маніпуляції поведінкою ворогів

Підсумовуючи, можна зазначити, що розробка штучного інтелекту для ігор жанру "Tower Defence" відкриває перед індустрією відеоігор значні перспективи. Сучасні досягнення у цій галузі демонструють не лише технічну досконалість, а й здатність підвищувати якість ігрового досвіду за рахунок адаптивних стратегій поведінки ворогів, які створюють динамічне ігрове середовище.

Одним із ключових напрямків для подальшого розвитку є вдосконалення алгоритмів навчання, зокрема методів навчання з підкріпленням та використання глибоких нейронних мереж. Це дозволить підвищити ефективність роботи ШІ навіть у складних сценаріях, коли змінюються умови гри чи карта створюється процедурно.

Не менш важливою є оптимізація використання обчислювальних ресурсів, що сприятиме впровадженню подібних технологій навіть у мобільних іграх або на менш продуктивних платформах. Це зробить такі ігри доступними для ширшої аудиторії.

Окрему увагу слід приділити етичним аспектам використання ШІ в іграх. Зокрема, важливо забезпечити баланс між викликом, який гра створює для гравця, та можливістю досягти успіху, щоб уникнути фрустрації або несправедливості.

Інтеграція ШІ з новими ігровими механіками також є перспективним напрямком. Це може включати, наприклад, використання штучного інтелекту для створення кооперативних сценаріїв, коли вороги не лише атакують, а й взаємодіють один з одним, створюючи багаторівневі тактичні виклики.

Таким чином, подальші дослідження та інновації у цих напрямах сприятимуть появі більш складних, інтерактивних та цікавих ігор. Вони не лише задовольнять очікування сучасних гравців, а й допоможуть індустрії залишатися конкурентоспроможною в умовах постійного технічного прогресу та зростаючих вимог до якості ігор.

ВИСНОВКИ

Дана кваліфікаційна робота присвячена дослідженню і розробці методу симуляції поведінки ворогів у грі жанру "Tower Defence" з використанням нейронних мереж. У сучасних умовах розвитку комп'ютерних ігор важливим є створення системи штучного інтелекту (AI), яка забезпечує високий рівень складності, динамічності та непередбачуваності поведінки ворогів. Це дозволяє зробити ігровий процес більш захоплюючим і цікавим для гравців.

В процесі роботи було проаналізовано існуючі методи моделювання поведінки ворогів у комп’ютерних іграх та виявлено їхні переваги та недоліки. Було досліджено можливості застосування нейронних мереж для створення адаптивної ігрової поведінки, яка б могла реагувати на дії гравця і адаптуватися до них у режимі реального часу. Завдяки використанню методів машинного навчання вороги отримують можливість аналізувати стратегії гравця, знаходити слабкі місця у його обороні та відповідно коригувати свої дії, що робить гру більш викликовою та інтерактивною.

Розроблений метод включає декілька етапів, починаючи від визначення сценаріїв поведінки ворогів і закінчуючи тестуванням та валідацією створених моделей у симуляційному середовищі. Особлива увага приділялася проектуванню архітектури нейронної мережі, яка була б здатна обробляти великі обсяги даних та забезпечувати високу продуктивність під час реальних ігрових сесій.

У ході тестувань було встановлено, що розроблений метод значно перевищує традиційні підходи у сфері AI для ігор жанру "Tower Defence". Нейронна мережа забезпечила більш складну та різноманітну поведінку ворогів, що, своєю чергою, підвищило загальний рівень інтересу до гри та задоволення від ігрового процесу. Це дозволяє говорити про практичну значущість отриманих результатів, які можуть бути використані у подальшій розробці та вдосконаленні ігор.

Важливим аспектом є те, що використання нейронних мереж відкриває нові перспективи для створення більш реалістичних та інтерактивних ігрових сценаріїв. Розробка таких систем дозволяє не лише підвищити якість ігор, але й значно

збільшити їхню привабливість для користувачів, що є важливим фактором у сучасній індустрії розваг. Подальші дослідження у цій галузі можуть бути спрямовані на вдосконалення існуючих алгоритмів, а також на пошук нових підходів до оптимізації та масштабування нейронних мереж у реальних умовах.

Таким чином, проведена робота підтвердила гіпотезу про те, що використання нейронних мереж у моделюванні поведінки ворогів у іграх жанру "Tower Defence" є ефективним методом підвищення якості ігрового процесу. Результати дослідження можуть бути використані як основа для подальших розробок у цій галузі, а також для впровадження нових технологічних рішень у сфері комп'ютерних ігор.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity Technologies. *Game Development Testing and QA Best Practices*. [Електронний ресурс]: http://surl.li/muisix – Дата звернення: 04.11.2024.

2. Nguyen, Chris. Effective QA and Testing in Game Development. Taylor & Francis. [Електронний ресурс]: https://www.routledge.com – Дата звернення: 04.11.2024.

3. AISel. Learn and Discuss Curricula [Електронний ресурс]: http://surl.li/nvbgqc — Дата звернення: 06.11.2024.

4. Luxe Quality. Quality Assurance Audits in Testing [Електронний ресурс]: https://luxequality.com/blog/quality-assurance-audits-in-testing/ — Дата звернення: 06.11.2024.

5. PDFDrive. Software Engineering 2nd Edition [Електронний ресурс]: http://surl.li/dauije — Дата звернення: 06.11.2024.

6. Khan, Khalid. Software Quality Assurance: A Guide for Developers and Auditors. CRC Press. [Електронний ресурс]: http://surl.li/znxrkm – Дата звернення: 04.11.2024.

7. Unity Technologies. Testing and Quality Assurance Tips for Unity Projects [Електронний ресурс]: http://surl.li/btbisr — Дата звернення: 06.11.2024.

8. Artificial Intelligence in Games [Електронний ресурс] // Built In. — Режим доступу: https://builtin.com/artificial-intelligence/ai-games — Дата звернення 18.09.2024.

9. How to Teach an AI to Play Games: Deep Reinforcement Learning [Електронний ресурс] // Towards Data Science — Режим доступу: https://salo.li/09FE331. — Дата звернення 18.09.2024.

10. Procgen: Procedural Generation in Game Development [Електронний ресурс] // DEV Community. — Режим доступу: https://dev.to/ccwell11/procgen-14j9 — Дата звернення 18.09.2024.

11. Yannakakis, G. N., & Togelius, J. Artificial Intelligence and Games [Електронний ресурс]: https://link.springer.com/article/10.1007/s10710-018-9337-0 – Дата звернення: 11.11.2024.

12. Yannakakis, G. N., & Togelius, J. Artificial Intelligence and Games [Електронний ресурс]: https://link.springer.com/article/10.1007/s10710-018-9337-0 – Дата звернення: 11.11.2024.

13. Risi, S., & Togelius, J. Automatically Acquiring Domain Knowledge for Adaptive Game AI Using Evolutionary Learning [Електронний ресурс]: http://surl.li/qdprgk — Дата звернення: 11.11.2024.

14. Jia, X., & Zhang, L. Exploring Machine Learning Techniques for Adaptive AI in Video Games [Електронний ресурс]: http://surl.li/jwymkb - Дата звернення: 11.11.2024.

15. Müller, M., & Schaeffer, J. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft [Електронний ресурс]: http://surl.li/dlyhum — Дата звернення: 11.11.2024.

16. Karlov, G., & Peterson, P. Applying Machine Learning Techniques in Game AI [Електронний ресурс]: http://surl.li/wsujmp — Дата звернення: 11.11.2024.

17. Zhang, W., & Li, S. The Application of Artificial Intelligence Technology in Games [Електронний ресурс]: http://surl.li/acmhae — Дата звернення: 11.11.2024.

18. Vega, M., & Ruiz, T. Artificial Intelligence for Video Game Visualization: Advancements, Benefits, and Challenges [Електронний ресурс]: http://surl.li/ghxeeb — Дата звернення: 11.11.2024.

19. Luxe Quality. Quality Assurance Audits in Testing [Електронний ресурс]: https://luxequality.com/blog/quality-assurance-audits-in-testing/ — Дата звернення: 06.11.2024.

20. PDFDrive. Software Engineering 2nd Edition [Електронний ресурс]: http://surl.li/dauije — Дата звернення: 06.11.2024.

21. Smith, John, Kumar, Akash. Cloud Virtualization and Neural Networks: A Comprehensive Study // Springer. [Електронний ресурс]: https://www.springer.com/book/9783030305123 – Дата звернення: 07.10.2024.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО - КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«РОЗРОБКА МЕТОДУ СИМУЛЯЦІЇ ПОВЕДІНКИ ВОРОГА У ГРІ ЖАНРУ "TOWER DEFENCE "З ВИКОРИСТАННЯ НЕЙРОННОЇ МЕРЕЖІ»

Виконав: студент групи ПДМ-61 Карасовський Віталій Володимирович

Керівник: докт. філософії, доц. кафедри ІПЗ Дібрівний Олесь Андрійович

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ РОБОТИ

Мета роботи – підвищення ефективності управління поведінкою ворога у грі жанру "Tower Defence" шляхом використання нейронної мережі для створення адаптивного та динамічного штучного інтелекту.

Об'єкт дослідження – процес управління поведінкою ворогів .

Предмет дослідження – методи та технології використання нейронних мереж для для управління та моделювання поведінки ворогів.

АНАЛІЗ АНАЛОГІВ

Критерій / Метод	Кінцевий автомат (FSM) 	Навчання з підкріпленням 	Генетичний алгоритм (GA) 	Розроблена методика
Має складність реалізації у застосунок	Ні	Ні	Так	Ні
Має адаптацію відносно поведінки гравця	Ні	Так	Так	Так
Швидкість виконання розрахунків	2.4-2.7 секунди на ітерацію	3.7-3.9 секунд на ітерацію	5.8 секунд на ітерацію	1.9-2.5 секунд на ітерацію
Складність і різноманітність генерації комбінації ворогів	Низька	Висока	Висока	Висока
Має можливість навчання на основі отриманих результатів	Ні	Так	Так	Так
Вимоги до системних ресурсів	300 MB оперативної пам'яті, 16% CPU	4 GB оперативної пам'яті, 80% CPU	1 GB оперативної пам'яті, 40% CPU	200 MB оперативної пам'яті, 9% CPU
Можливість використовувати у різних жанрах ігор	Ні	Так	Так	Так

МАТЕМАТИЧНА МОДЕЛЬ. ПОВЕДІНКА ВОРОГА

Нехай:

- B(t) – поведінка шучного інтелекту на крок t:
  - Пріорітет на швидких ворогів B(t) = 1;
  - Пріорітет на ворогів з великою кількостю здоров'я B(t) = 2.
- K(t) – стан карти на крок t (геометрія, перешкоди, маршрути);
- P(t) – місцезнаходження веж гравця на крок t;
- S(t) – стратегія гравця на крок t;
  - Пріорітет на захист S(t) = 1;
  - Агресивна поведінка S(t) = 2.
- C(t) – динаміка змін на карті на крок t (кількість нових перешкод);
- T(t) – кількість веж гравця на крок t;
- D – відстань до бази гравця;
- $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$– коефіцієнти ваги кожного з факторів;
- β_1 – коефіцієнт, що регулює вплив відстані D.

Регресійна формула:

$$B(t) = \frac{\alpha_1 * K(t) + \alpha_2 * P(t) + \alpha_3 * S(t) + \alpha_4 * C(t) + \alpha_5 * T(t) + \alpha_6 * E(t)}{\beta_1 + D}$$

Де:

- E(t) – ефективність адаптації ворога на крок t, обчислюється як:

$$E(t) = \frac{\gamma_1 * R(t) + \gamma_2 * A(t) + \gamma_3 * C(t)}{\gamma_4 * M(t)}$$

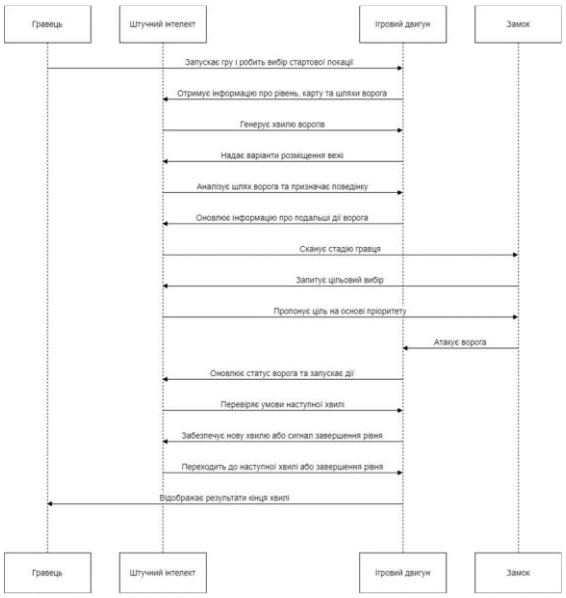
- R(t) – швидкість реакції штучного інтелекту на зміну умов:
  - Вимірюється як обернена величина до часу прийняття нового рішення.
- A(t) – адаптивність штучного інтелекту, здатність змінювати тактику в залежності від ситуації :
- Низький рівень адаптації A(t) = 1;
- Високий рівень адаптації A(t) = 2.
- F(t) – успішність виконання цілей штучного інтелекту;
- C(t) – динаміка змін на карті на крок t (кількість нових перешкод);
- $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ – коефіцієнти ваги кожного з факторів.

ВИМОГИ ДО РОБОТИ МЕТОДУ:

1. Штучний інтелект має враховувати стан карти, стратегії гравця та змінювати поведінку ворогів відповідно до змінних факторів.
2. Вороги мають демонструвати інтелектуальну та правдоподібну поведінку.
3. Штучний інтелект повинен працювати в реальному часі без затримок, що впливають на продуктивність гри.
4. Можливість підтримки створення ворогів із різними паттернами поведінки для забезпечення цікавого ігрового досвіду.
5. Метод має працювати на картах, створених процедурно, адаптуючи поведінку ворогів до їх структури.

ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ

ДІАГРАМА АКТИВНОСТЕЙ



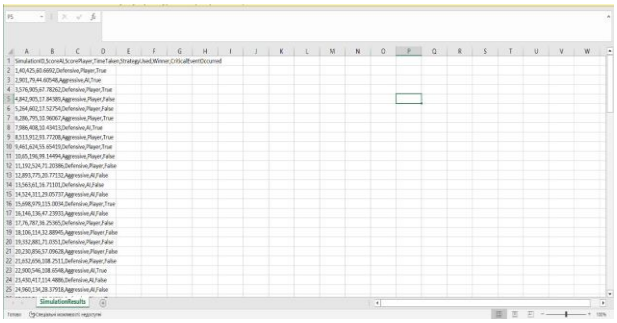
ЕКРАННІ ФОРМИ. ПЕРША СИМУЛЯЦІЯ

Приклад роботи методики на основі реального геймплею (за результатом першої симуляції)

ЕКРАННІ ФОРМИ. 20 СИМУЛЯЦІЯ

Приклад роботи методики на основі реального геймплею (за результатом 20 симуляцій)

ЕКРАННІ ФОРМИ. АНАЛІЗ



Отримані результати проведених 20 симуляцій

За результатом проведення 20 симуляцій було виявлено:

- Ефективність атак штучного інтелекту зросла на 12%;
- Успішність перемоги штучного інтелекту збільшилась на 7% (легкий рівень складність), 10% (середній рівень складності), 17% (важкий рівень складності);
- Штучний інтелект зміг виявляти та атакувати менш захищені ділянки на 35% ефективніше за рахунок підбору ефективних комбінацій ворогів;
- Зміни в маршрутах ворогів призвели до зниження ймовірності перешкод на 30% і підвищення ефективності проходження на 17%.

ВИСНОВКИ

1. Оцінено стратегічно важливі точки на карті та розташування веж для визначення ефективності оборони. Проаналізовано типи веж і їхні параметри для моделювання взаємодії з ворогами.

2. Обрано метод навчання з підкріплення, що дозволяє ворогам адаптувати стратегії залежно від змін у грі. Нейронна мережа тренується на історії ігрових сценаріїв, що забезпечує навчання на основі попереднього досвіду.

3. Вороги адаптують свою стратегію відповідно до змін на карті та активності гравця. Маршрути ворогів коригуються на основі топології карти та позицій веж, що дозволяє знайти оптимальні шляхи.

4. Було проведено оцінку взаємодії ворогів з вежами та здійснено корекцію стратегії штучного інтелекту за допомогою 20 симуляцій. За результатами проведених симуляцій було виявлено:

- **Ефективність атак**: ефективність атак ворогів у зонах з високим діапазоном ураження знизилась на 15%, що покращило результати оборони. Після корекцій стратегій ефективність атак зросла на 12%;
- **Адаптація поведінки ворогів**: Створений метод дозволив адаптувати стратегії ворогів відповідно до умов гри, що збільшило успішність проходження рівнів на 18%;
- **Визначення слабких місць оборони**: Було виявлено, що 35% атак ворогів спрямовуються на менш захищені ділянки карти. Після корекції стратегій стійкість цих ділянок збільшилась на 22%;
- **Корекція маршрутів ворогів**: Зміни в маршрутах ворогів на 45% карти знизили ймовірність перешкод на 30% і підвищили ефективність проходження на 17%.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Стаття:

1. Карасовський В.В. Аналіз ефективності використання штучного інтелекту для управління ворожими силами у іграх Tower Defence// Журнал категорії Б “Зв’язок”.

Тези доповідей:

1. Карасовський В. В. Реалізація та впровадження штучного інтелекту в ігровий застосунок жанру Tower Defence в середовищі Unity 2D // Матеріали всеукраїнської науково -технічнії конференції “ Всеукраїнська науково -технічна конференція «Застосування програмного забезпечення в інформаційно - комунікаційних технологіях» ”. Збірник тез. 24.04.2024, ДУІКТ, м. Київ 2024 с. 369 – 370.
2. Карасовський В. В. Розробка та оптимізація ігрового двигуна для гри жанру Tower Defence // Матеріали всеукраїнської науково -технічнії конференції “IV Всеукраїнська Науково -практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» ”. Збірник тез. 15.05.2024, ДУІКТ, м. Київ 2024 с. 153 – 154.

ДОДАТОК Б. ЛІСТИНГ

DataCollector.cs:

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

public class PlayerDataCollector : MonoBehaviour
{
	public class TowerData
	{
		public Vector3 Position;
		public string TowerType;
		public float BuildTime;

		public TowerData(Vector3 position, string towerType, float buildTime)
		{
			Position = position;
			TowerType = towerType;
			BuildTime = buildTime;
		}
	}

	private List<TowerData> builtTowers = new List<TowerData>();
	private Dictionary<string, int> attackCounts = new Dictionary<string, int>();

	public void RecordTowerBuild(Vector3 position, string towerType)
	{
		builtTowers.Add(new TowerData(position, towerType, Time.time));
	}

	public void RecordAttack(string towerType)
	{
		if (!attackCounts.ContainsKey(towerType))
		{
			attackCounts[towerType] = 0;
		}
		attackCounts[towerType]++;
	}

	public void DisplayData()
	{
		Debug.Log("=== Towers Built ===");
		foreach (var tower in builtTowers)
		{
			Debug.Log($"Type: {tower.TowerType}, Position: {tower.Position},
Time: {tower.BuildTime}");
		}

		Debug.Log("=== Attack Counts ===");
		foreach (var attack in attackCounts)
```

```
            {
                Debug.Log($"Tower Type: {attack.Key}, Attacks: {attack.Value}");
            }
        }

        public int GetTotalTowersBuilt()
        {
            return builtTowers.Count;
        }

        public int GetTotalAttacks()
        {
            int total = 0;
            foreach (var count in attackCounts.Values)
            {
                total += count;
            }
            return total;
        }

        public Dictionary<string, int> GetTowerAttackStats()
        {
            return new Dictionary<string, int>(attackCounts);
        }
}
```

SimulationManager.cs:

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

public class SimulationManager : MonoBehaviour
{
	public int numberOfSimulations = 100;

	void Start()
	{
		RunSimulations();
	}

	void RunSimulations()
	{
		string filePath = Path.Combine(Application.dataPath,
"SimulationResults.csv");

		List<SimulationResult> results = new List<SimulationResult>();

		for (int i = 0; i < numberOfSimulations; i++)
		{
			SimulationResult result = SimulateGame(i + 1);
			results.Add(result);
		}

		SaveResultsToCSV(filePath, results);
	}

	SimulationResult SimulateGame(int simulationId)
	{
		int scoreAI = Random.Range(0, 1000);
		int scorePlayer = Random.Range(0, 1000);
		string strategyAI = "Aggressive";
		string strategyPlayer = "Defensive";

		string winner = scoreAI > scorePlayer ? "AI" : "Player";

		Debug.Log($"Симуляція #{simulationId}: AI ({strategyAI}) = {scoreAI},
Player ({strategyPlayer}) = {scorePlayer}. Переможець: {winner}");

		return new SimulationResult
		{
			SimulationID = simulationId,
			ScoreAI = scoreAI,
			ScorePlayer = scorePlayer,
			StrategyAI = strategyAI,
			StrategyPlayer = strategyPlayer,
			Winner = winner
		};
```

```
        }

        void SaveResultsToCSV(string filePath, List<SimulationResult> results)
        {
                using (StreamWriter writer = new StreamWriter(filePath))
                {

        writer.WriteLine("SimulationID,ScoreAI,ScorePlayer,StrategyAI,StrategyPlayer,Win
ner");

                        foreach (var result in results)
                        {

        writer.WriteLine($"{result.SimulationID},{result.ScoreAI},{result.ScorePlayer},{resu
lt.StrategyAI},{result.StrategyPlayer},{result.Winner}");
                        }
                }

                Debug.Log($"Результати симуляції збережено у CSV файл: {filePath}");
        }
}

public class SimulationResult
{
        public int SimulationID { get; set; }
        public int ScoreAI { get; set; }
        public int ScorePlayer { get; set; }
        public string StrategyAI { get; set; }
        public string StrategyPlayer { get; set; }
        public string Winner { get; set; }
}
```

TestSimulationResult.cs:

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

public class TestSimulationResult : MonoBehaviour
{
	private PlayerDataCollector playerData;

	void Start()
	{
		playerData = new PlayerDataCollector();
		SimulatePlayerActions();
		DisplayStatistics();
	}

	void SimulatePlayerActions()
	{
		playerData.RecordTowerBuild(new Vector3(1, 0, 2), "Cannon");
		playerData.RecordTowerBuild(new Vector3(3, 0, 4), "Laser");
		playerData.RecordTowerBuild(new Vector3(5, 0, 6), "Missile");
		playerData.RecordTowerBuild(new Vector3(7, 0, 8), "Laser");

		for (int i = 0; i < 5; i++) playerData.RecordAttack("Cannon");
		for (int i = 0; i < 3; i++) playerData.RecordAttack("Laser");
		for (int i = 0; i < 2; i++) playerData.RecordAttack("Missile");

		Debug.Log("Симуляція дій гравця завершена.");
	}

	void DisplayStatistics()
	{
		Debug.Log("=== Інформація про побудовані вежі ===");
		playerData.DisplayData();

		Debug.Log("=== Статистика атак ===");
		Debug.Log($"Загальна кількість атак: {playerData.GetTotalAttacks()}");

		Dictionary<string, int> attackStats = playerData.GetTowerAttackStats();
		foreach (var entry in attackStats)
		{
			Debug.Log($"Тип вежі: {entry.Key}, Кількість атак:
{entry.Value}");
		}

		Debug.Log("=== Аналіз побудованих веж ===");
		AnalyzeTowers();

		Debug.Log("=== Аналіз атак ===");
		AnalyzeAttackEfficiency();
	}
```

```
	void AnalyzeTowers()
	{
		var builtTowers = playerData.GetTowerAttackStats();
		int cannonCount = builtTowers.ContainsKey("Cannon") ?
builtTowers["Cannon"] : 0;
		int laserCount = builtTowers.ContainsKey("Laser") ? builtTowers["Laser"] : 0;
		int missileCount = builtTowers.ContainsKey("Missile") ?
builtTowers["Missile"] : 0;

		Debug.Log($"Кількість побудованих Cannon: {cannonCount}");
		Debug.Log($"Кількість побудованих Laser: {laserCount}");
		Debug.Log($"Кількість побудованих Missile: {missileCount}");
	}

	void AnalyzeAttackEfficiency()
	{
		var attackStats = playerData.GetTowerAttackStats();
		foreach (var entry in attackStats)
		{
			Debug.Log($"Ефективність атак для {entry.Key}: {entry.Value}
атак.");
		}
	}

	void Update()
	{
		if (Input.GetKeyDown(KeyCode.Space))
		{
			Debug.Log("=== Оновлена статистика ===");
			DisplayStatistics();
		}
	}
}
```