МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____ Євген ДАВИДЕНКО

підпис

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІГРОВИЙ ЗАСТОСУНОК ІЗ ВИКОРИСТАННЯМ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувачка _____ **Олена ШЕРСТЮК**

підпис

«___»_____ 2025 р.

Керівник роботи

канд. техн. наук, доцент _____ **Євген ДАВИДЕНКО**

підпис

«___»_____ 2025 р.

Миколаїв – 2025

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну магістерську роботу здобувачки вищої освіти

Шерстюк Олени
(прізвище, власне ім'я)

1. Тема кваліфікаційної роботи «Ігровий застосунок із використанням алгоритмів машинного навчання» затверджена наказом в. о. ректора ЧНУ імені Петра Могили № 182 від «02» липня 2025 р.
2. Строк представлення кваліфікаційної роботи «22» грудня 2025 р.
3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом є розробка та реалізація ігрового застосунку із використанням алгоритмів машинного навчання
4. Перелік питань, що підлягають розробці:
проведення аналізу аналогічних ігрових систем та дослідження особливостей застосування алгоритмів машинного навчання в ігровому середовищі, дослідження

можливостей рушія Unity для інтеграції ML-агентів, проєктування ігрового середовища й побудова необхідних моделей, розробка та реалізація системи навчання агентів у Unity, інтеграція ігрової логіки з поведінкою ML-агентів, проведення тестування та відлагодження ігрового застосунку.

5. Перелік графічних матеріалів

Презентація

6. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Дата видачі завдання «___» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Ігровий застосунок із використанням алгоритмів машинного навчання»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КМР	01.09.2025	03.09.2025	Виконано
2	Ознайомлення з літературою та аналіз теми роботи	04.09.2025	12.09.2025	Виконано
3	Складання календарного плану КМР	15.09.2025	16.09.2025	Виконано
4	Аналіз предметної області	16.09.2025	19.09.2025	Виконано
5	Розробка проєктних рішень	22.09.2025	24.09.2025	Виконано
6	Моделювання та конструювання ПЗ	25.09.2025	01.10.2025	Виконано
7	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування	02.10.2025	21.11.2025	Виконано
8	Оформлення КМР та презентації	21.11.2025	21.11.2025	Виконано
9	Попередній захист	24.11.2025	24.11.2025	Виконано
10	Відгук керівника КМР	28.11.2025	28.11.2025	Виконано
11	Рецензування	01.12.2025	05.12.2025	Виконано
12	Завершення оформлення КМР та презентації	08.12.2025	15.12.2025	Виконано
13	Захист кваліфікаційної роботи	22.12.2025	22.12.2025	Виконано

Здобувачка _____ **Олена ШЕРСТЮК**

підпис

«___»_____ 2025 р.

Керівник роботи

канд. техн. наук, доцент _____ **Євген ДАВИДЕНКО**

підпис

«___»_____ 2025 р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

Ігровий застосунок із використанням алгоритмів машинного навчання

Здобувачка 608м гр.: Шерстюк Олена

Керівник: канд. техн. наук, доцент Давиденко Євген

Кваліфікаційна магістерська робота присвячена створенню ігрового застосунку із використанням алгоритмів машинного навчання.

Об’єктом роботи є процеси, які пов’язані з аналізом дій гравця та генерацією персоналізованих завдань ігрового застосунку.

Предметом роботи є алгоритми машинного навчання для аналізу дій гравця.

Метою кваліфікаційної роботи є забезпечення адаптивності ігрових застосунків за рахунок використання алгоритмів машинного навчання.

Кваліфікаційна магістерська робота складається із вступу, чотирьох розділів, висновків та переліку джерел посилання.

У вступі визначено актуальність, науково-практичне значення обраної теми, об’єкт та предмет дослідження, а також мета та завдання роботи.

Перший розділ містить системний аналіз обраної предметної сфери та формування постановки задачі. Здійснено огляд сучасних програмних рішень та їх аналогів у предметній області.

У другому розділі проведено моделювання об’єкта та предмету дослідження, описано методи, технології та математичний апарат, а також сформовано специфікацію вимог до програмного забезпечення.

Третій розділ містить опис виконаної роботи з моделювання та конструювання програмного забезпечення.

У четвертому розділі продемонстровано виконану роботу з кодування розробленого програмного забезпечення, тестування його функціональності.

У висновках оцінено отримані результати відносно аналогів, описано практичне значення результатів та рекомендації щодо їх впровадження у певній сфері діяльності.

Кваліфікаційна робота викладена на 92 сторінки машинного тексту, складається із вступу, 4 розділів, 39 ілюстрацій, 6 таблиць, 32 джерел в переліку посилань, 1 додатку.

Ключові слова: Action-RPG, ігровий рушій Unity, алгоритми машинного навчання, Unity ML-Agents, навчання з підкріпленням, ігровий застосунок, адаптивність.

ABSTRACT

to the qualifying master's thesis

Game application based on machine learning algorithms

Student of group 608: Sherstiuk Olena

Supervisor: Candidate of Technical Sciences, Associate Professor Davydenko Yevhen

The master's thesis is devoted to the creation of a game application using machine learning algorithms.

The object of the work is the processes related to the analysis of player actions and the generation of personalized tasks for the game application.

The subject of the work is machine learning algorithms for analyzing player actions.

The purpose of the qualification work is to provide adaptiveness of a game application using machine learning algorithms.

The master's thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction defines the relevance and scientific and practical significance of the chosen topic, the object and subject of the study, as well as the purpose and objectives of the work.

The first chapter contains a systematic analysis of the chosen subject area and the formulation of the problem. An overview of modern software solutions and their analogues in the subject area is provided.

The second chapter models the object and subject of the study, describes the methods, technologies, and mathematical apparatus, and forms the software requirements specification.

The third chapter contains a description of the work performed on modeling and designing the software.

The fourth chapter demonstrates the work done on coding the developed software and testing its functionality.

The conclusions evaluate the results obtained in relation to analogues, describe the practical significance of the results, and recommendations for their implementation in a specific field of activity.

The thesis is 92 pages long and consists of an introduction, 4 sections, 39 illustrations, 6 tables, 32 sources in the list of references, and 1 appendix.

Keywords: Action-RPG, Unity game engine, machine learning algorithms, Unity ML-Agents, reinforcement learning, game application, adaptiveness.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МН	–	Машинне навчання
ПЗ	–	Програмне забезпечення
ПК	–	Персональний комп’ютер
ШІ	–	Штучний інтелект
API	–	Application programming interface
JSON	–	JavaScript Object Notation
ML-Agents	–	Machine Learning Agents
NPC	–	Non-playable-character
PPO	–	Proximal Policy Optimization
RPG	–	Role-Playing Game
UI	–	User Interface
UML	–	Unified modeling language

ВСТУП

У наш час комп’ютерні ігри є одним із найбільш динамічних і перспективних сегментів розважальної індустрії. Їх вплив на культуру та економіку продовжує зростати, відкриваючи нові можливості для розвитку творчості, технологій та навіть способу життя. До того ж, комп’ютерні ігри стають не лише формою розваги, а й інструментом навчання, тренування навичок та розвитку креативності. Вони проникають у різні сфери нашого життя, від освіти та медицини до бізнесу та соціальної сфери. Крім того, ігрова індустрія продовжує активно впливати на формування молодіжної культури та громадської думки, надаючи значний вплив на свідомість та поведінку людей.

З огляду на це сучасні розробники активно впроваджують інноваційні технології, які роблять ігровий процес більш інтерактивним і персоналізованим. Однією з ключових тенденцій є застосування штучного інтелекту (ШІ) та машинного навчання (МН), що дозволяє іграм адаптуватися до дій гравця, створювати реалістичні світи та пропонувати унікальний досвід кожному користувачеві.

Завдяки цим технологіям ігри стають розумнішими, цікавішими та адаптивнішими, що робить процес більш захоплюючим і запам’ятовується гравцю. ШІ дозволяє створювати складні сценарії поведінки персонажів, регулювати рівень складності відповідно до навичок користувача, а також генерувати унікальні завдання та сюжетні події. Крім того, штучний інтелект керує поведінкою комп’ютерних супротивників, створює реалістичні світи та персонажів, передбачає дії гравця та пропонує унікальний ігровий досвід, що значно підвищує рівень занурення та інтерес до гри.

Щоб реалізувати всі ці можливості та продемонструвати алгоритми машинного навчання в дії, розробники використовують спеціальні ігрові рушії та програмні інструменти, такі як Unity, Unreal Engine, Godot та інші. Ці платформи дозволяють створювати інтерактивні сцени, керувати об’єктами в реальному часі та інтегрувати моделі 3D-графіки, а також адаптувати ігровий процес відповідно

до дій гравця. Використання таких технологій є ключовим для створення якісних, захоплюючих та інноваційних ігор, здатних запропонувати гравцям унікальний і персоналізований досвід.

Об’єктом роботи є процеси, які пов’язані з аналізом дій гравця та генерацією персоналізованих завдань ігрового застосунку.

Предметом роботи є алгоритми машинного навчання для аналізу дій гравця.

Метою кваліфікаційної роботи є забезпечення адаптивності ігрових застосунків за рахунок використання алгоритмів машинного навчання.

Для досягнення поставленої мети, потрібно виконати такі завдання:

– дослідити принципи роботи алгоритмів машинного навчання та їх можливості в ігрових застосунках;

– проаналізувати сучасні ігри, що використовують машинне навчання, та визначити їхні ключові особливості;

– розробити концепцію, архітектуру та механіку ігрового застосунку;

– створити інтерфейс та графічний дизайн;

– реалізувати ігровий застосунок і забезпечити його адаптивність за рахунок використання алгоритмів машинного навчання;

– перевірити якість роботи застосунку.

Область застосування: запропонований ігровий застосунок призначений для створення персоналізованого ігрового досвіду за допомогою алгоритмів машинного навчання. Він аналізує дії гравця та динамічно генерує завдання або рівні, що адаптуються під його стиль гри. Це дозволяє забезпечити унікальний ігровий процес для кожного користувача, підвищуючи рівень занурення та повторного проходження.

Апробація результатів КМР відбулася під час XXVIII Всеукраїнської науково-практичної конференції «Могилянські читання – 2025», Миколаїв, 10-14 листопада, 2025 р. (Додаток А).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Сучасні тенденції розвитку ігрових застосунків

Наразі штучний інтелект і машинне навчання стали невід'ємною частиною нашого повсякденного життя: від персональних помічників на телефонах до рекомендацій у стримінгових сервісах. Технології ШІ та МН застосовуються в різних галузях, зокрема в охороні здоров'я, фінансах і транспорті, для підвищення ефективності та точності.

Сучасні ігрові застосунки активно переходять від статичних сценаріїв до динамічних і адаптивних, де кожне рішення гравця впливає на розвиток подій. Це створює нові виклики для розробників, які повинні забезпечити реалістичну та захопливу взаємодію з користувачем.

Технології ШІ та МН відкривають нові горизонти для створення ігрових застосунків, здатних адаптуватися до поведінки гравця та пропонувати унікальні сценарії і завдання. Це забезпечує не лише персоналізований ігровий досвід, а й розширює можливості для наукових досліджень у галузі геймдизайну та розробки ігор [1].

Крім того, методи, розроблені для ігрових застосунків, можна адаптувати до інших сфер – навчальних платформ, симуляторів та інтерфейсів для користувачів, де важлива адаптивність і взаємодія з інтелектуальними системами. Це дає змогу переносити інновації, застосовувані в ігровій індустрії, в інші галузі, де ШІ та МН можуть істотно підвищити ефективність і персоналізацію користувацького досвіду.

Історія штучного інтелекту та машинного навчання відзначена численними віхами та досягненнями. Вона бере свій початок у 1940-х роках із винаходом електронного комп'ютера.

У 1943 році Воррен Маккалок і Вольтер Піттс запропонували модель штучних нейронів – першу роботу, що заклала основи ШІ. У 1947 році Алан Тюрінг виступив із публічною лекцією, у якій згадав про комп'ютерний інтелект і можливості машинного самовизначення. У 1950 році Клод Шеннон створив «Тесея» – роботизовану мишу, що стала однією з перших систем ШІ [26].

У 1956 році Джон Маккарті організував Дартмутську конференцію, яку вважають місцем народження сучасного штучного інтелекту. На ній було запропоновано ідею використання комп'ютерів для моделювання людського інтелекту, що ознаменувало нову еру обчислювальної техніки.

Машинне навчання – розділ ШІ, який ґрунтується на навчанні машин за допомогою даних. Наприкінці 1950-х Артур Самуель уперше використав термін «машинне навчання» для опису галузі, яка дозволяє комп'ютерам вчитися без явного програмування. Він також створив програму для гри в шашки, що поступово покращувала свою продуктивність із кожною новою партією.

У 1958 році з'явилася мова FORTRAN – перша для числових і наукових обчислень, що дало змогу розробникам писати складні алгоритми й обробляти великі обсяги даних. У 1959 році Джон Маккарті та Марвін Мінський заснували проєкт ШІ в Массачусетському технологічному інституті (MIT). Того ж року Маккарті запропонував використання LISP (скорочення від «LISt Processing») – першої мови програмування, спеціально створеної для досліджень у галузі ШІ [26].

У 1960–1970-х роках відбувся значний прогрес у дослідженнях ШІ та МН. У 1967 році було створено універсальний розв'язувач задач (GPS), здатний опрацьовувати широкий спектр проблем, а у 1970 році – першу систему комп'ютерного зору, що розпізнавала рукописні цифри.

Наприкінці 1970-х стало очевидно, що ручних правил і евристик недостатньо. Це спричинило розвиток нового підходу – конекціонізму, зосередженого на нейронних мережах, здатних навчатися на основі даних. У 1980-х цей напрям набув популярності завдяки алгоритму зворотного поширення помилки, який зробив навчання нейронних мереж ефективнішим.

У 1990-х дослідження продовжувалися, з'являлися нові алгоритми та методи. У 1997 році комп'ютер IBM Deep Blue переміг чемпіона світу з шахів Гаррі Каспарова – знакова подія в історії ШІ.

Сьогодні штучний інтелект і машинне навчання застосовуються у найрізноманітніших сферах – від розпізнавання мовлення та класифікації зображень до автономного водіння. Потенціал їх використання практично

безмежний, тому інвестиції та наукові дослідження в цих галузях невпинно зростають.

Особливе місце серед прикладних напрямів займають інтелектуальні агенти машинного навчання, які дозволяють використовувати ігри та симуляції як навчальне середовище для розвитку адаптивних систем. Таке навчання може здійснюватися за допомогою методів підкріплення, імітаційного навчання, нейроеволюції та інших підходів. Навчені агенти знаходять застосування у різноманітних сценаріях – від керування поведінкою неігрових персонажів (Non-Player Character, NPC) у динамічних, багатоагентних або змагальних середовищах до автоматизованого тестування ігрових збірок і попередньої оцінки рішень щодо дизайну гри [21].

Одним із ключових напрямів використання МН-агентів є навчання NPC для забезпечення динамічної адаптації ігрового процесу. Наприклад, неігровий персонаж може навчитися орієнтуватися у складному лабіринті, отримуючи винагороди за досягнення цілі та штрафи за зіткнення з перешкодами.

Схожий підхід ефективно застосовується й у робототехнічних симуляціях: віртуальна рука робота може навчитися захоплювати об’єкти методом проб і помилок, уникаючи витрат і ризиків, пов’язаних із фізичними експериментами. Для підвищення ефективності навчання часто використовується підхід навчання за навчальною програмою (curriculum learning), коли агент спочатку виконує прості завдання, які поступово ускладнюються. Наприклад, симуляція безпілотного автомобіля може спочатку навчити агента рухатися по прямій дорозі, а згодом – долати повороти або уникати перешкод.

Розробники також поєднують навчання з підкріпленням (Reinforcement Learning) та імітаційне навчання, коли агент спершу відтворює дії людини для прискореного формування базових моделей поведінки. Такий гібридний підхід дозволяє суттєво скоротити час навчання та підвищити стабільність результатів.

Одним із найпомітніших застосувань штучного інтелекту в ігрових застосунках є створення супротивників з адаптивною поведінкою. Використовуючи методи машинного навчання, ШІ може підлаштовуватися під

стиль гри користувача, пропонуючи складніші випробування, передбачаючи його дії та створюючи більш захопливий і непередбачуваний ігровий процес. У стратегічних або шутер-іграх, наприклад, система може автоматично регулювати рівень складності, що забезпечує постійний інтерес і баланс між викликом та ігровим комфортом [2].

Інший важливий аспект використання ШІ – це покращення графіки та фізики ігор. Завдяки технологіям машинного навчання, розробники можуть створювати більш реалістичні та деталізовані ігрові світи. Наприклад, ШІ може адаптувати освітлення, текстури та анімації в залежності від дій гравця чи зміни умов гри [2]. Це дозволяє зробити ігри не лише більш привабливими, а й більш реалістичними, що важливо для сучасних гравців, які очікують високу якість візуального та аудіо оформлення.

Також штучний інтелект відкриває нові можливості для розробки інноваційних ігрових механік. Завдяки методам машинного навчання можна створювати персоналізовані завдання та сюжети, які адаптуються до інтересів і стилю гри кожного користувача. Це дає змогу розробникам пропонувати унікальні рівні, що змінюються залежно від рішень гравця, забезпечуючи новий рівень взаємодії та залученості.

Зі зростанням популярності штучного інтелекту в індустрії розробки ігор він дедалі більше збагачує досвід розробників, які використовують його можливості для створення більш глибоких, динамічних і захопливих ігрових процесів.

Зростання рівня інтерактивності, поява нових технологій і розвиток інтелектуальних систем формують передумови для інтеграції інноваційних підходів у процес створення ігор. Особливої уваги заслуговує застосування алгоритмів машинного навчання, що відкривають можливості для динамічної адаптації геймплею, персоналізації досвіду гравців і моделювання реалістичної поведінки ігрових агентів.

1.2 Особливості ігрових застосунків

У сучасному цифровому світі відеоігри – це більше, ніж просто розвага; вони перетворилися на складні інтерактивні проєкти, що залучають гравців унікальним чином. Успіх гри залежить від її особливостей – елементів, які визначають ігровий досвід, формують процес гри та, зрештою, забезпечують її привабливість.

Ігрові особливості суттєво впливають на взаємодію гравців, створюючи взаємопов’язані системи механіки, динаміки та естетики. Ці елементи формують ігрові цикли, умови перемоги та системи винагород, сприяючи підтриманню зацікавленості гравців.

Поступове відкриття нових можливостей підтримує інтерес гравців, додаючи нові здібності, предмети та елементи сюжету у стратегічно важливі моменти. Такий структурований розвиток дозволяє поступово розвивати навички гравців перед введенням більш складних елементів ігрового процесу. Завдяки продуманим механікам розширюється вплив гравця, що дає змогу приймати рішення щодо розвиток персонажа, спорядження й вибору сюжетної лінії, які безпосередньо впливають на перебіг гри.

Сучасні ігри часто поєднують у собі механічні функції (наприклад, бойові системи, управління рухом, керування ресурсами) та соціальні функції (наприклад, онлайн-підбір гравців, таблиці лідерів, спілкування між користувачами). Кожна категорія функцій вносить свій внесок у загальне сприйняття гри, виконуючи певні завдання в межах її структури.

Ігрові функції – це ключові компоненти відеоігор, які визначають інтерактивну взаємодію користувача з системою. До них належать ігрова механіка, користувацький інтерфейс, системи розвитку персонажів та аудіовізуальні елементи. Саме ці функції формують залученість гравця через ігрові цикли, схеми управління та додаткові підсистеми – відстеження досягнень, соціальні можливості тощо. Із розвитком технологій нові інновації, зокрема ШІ, процедурна генерація контенту та кросплатформна сумісність, значно покращують ігровий досвід.

Інтеграція традиційних і новітніх функцій сприяє еволюції відеоігор, пропонуючи гравцям дедалі більш захопливі й динамічні сценарії взаємодії [19].

Такі елементи, як інтуїтивно зрозумілий користувацький інтерфейс, чутливе управління та постійний зворотний зв’язок, підсилюють ефект занурення, роблячи вибір і дії гравця більш осмисленими. Успішні ігри органічно поєднують ці компоненти, створюючи цілісний ігровий процес, у якому механіка, сюжет і естетика гармонійно взаємодіють і доповнюють один одного.

Ринок відеоігор нині перевищує за обсягом кіно- та музичні індустрії разом узяті, що спричинило появу великої кількості ігор на різних платформах – від мобільних пристроїв до персональних комп’ютерів, ігрових консолей і хмарних сервісів. Кожна з цих платформ має власні особливості, переваги та обмеження, а вибір середовища реалізації може суттєво впливати на особливості ігрового процесу. Водночас для багатьох користувачів пошук гри, що відповідає їхнім інтересам, залишається складним завданням. Жанр гри визначається не її тематикою, сеттингом чи цільовою аудиторією (наприклад, дітьми), а тим, що саме робить гравець у грі, тобто її основною ігровою механікою.

Сучасна ігрова індустрія відзначається значною різноманітністю стилів і підходів до побудови ігрового процесу. Протягом останніх десятиліть сформувалися окремі жанри, кожен з яких пропонує унікальний рівень інтерактивності та тип ігрового досвіду. Жанрова класифікація допомагає гравцям швидко зрозуміти, які виклики та враження пропонує конкретна гра [17].

Жанр відеоігор – це категорія ігор, об’єднаних спільними характеристиками ігрового процесу. На відміну від фільмів чи книжок, де жанри ґрунтуються переважно на тематиці або сеттингу, у відеоіграх жанр визначається типом взаємодії гравця з грою. Наприклад, шутер від першої особи (First-Person Shooter, FPS) залишається FPS незалежно від того, чи відбуваються події у науково-фантастичному світі, на історичному полі бою або у фентезійному всесвіті. Визначальними елементами тут є саме вид від першої особи та бойова механіка з використанням зброї дальнього бою.

Жанри часто поділяються на піджанри, що забезпечують точнішу класифікацію. Наприклад, до жанру екшен відносяться платформери, файтинги та шутери, кожен із яких має власну унікальну механіку. Деякі сучасні ігри поєднують елементи кількох жанрів, створюючи гібридний ігровий досвід, що дозволяє урізноманітнити геймплей і розширити аудиторію [32].

Жанрова приналежність визначається насамперед ігровим процесом, а не сюжетом, тематикою чи сеттингом – ці характеристики лише деталізують гру, але не визначають її жанр.

Жанри відіграють важливу роль у розробці ігор, оскільки допомагають розробникам зрозуміти вподобання гравців. Ця інформація дозволяє створювати продукти, що мають більшу ймовірність успіху. Знання популярних жанрів сприяє вибору відповідних ігрових механік, сюжетних підходів і функцій, які відповідають очікуванням користувачів.

Нижче наведено приклади найпопулярніших жанрів відеоігор [32]:

Екшен (Action) – ігри, що перевіряють швидкість реакції та координацію гравця. Піджанри, як-от файтинги або шутери, користуються великою популярністю. Класичні приклади – Call of Duty або Street Fighter, які поєднують динамічні бойові дії з напруженим геймплеєм.

Пригоди (Adventure) – зосереджені на дослідженні світу та розв’язанні головоломок. Вони вирізняються глибокими сюжетами та широкими ігровими просторами. Серії The Legend of Zelda або Uncharted демонструють приклади захопливого ігрового процесу з акцентом на сюжет і відкриття.

Платформери (Platformers) – двовимірні ігри з видом збоку, де гравці долають перешкоди, стрибають, бігають чи деруться складними маршрутами. Такі ігри, як Super Mario Bros. або Donkey Kong, стали основою жанру й залишаються популярними донині.

Рольові ігри (Role-Playing Game) – дозволяють гравцю керувати персонажем або групою, приймаючи рішення, що впливають на сюжет і розвиток. Приклади – Final Fantasy або The Witcher, відомі своєю глибиною, оповідальністю та системою розвитку персонажів.

Спортивні ігри (Sports) – відтворюють реальні види спорту з різним рівнем реалізму. Гравці можуть приміряти ролі професійних спортсменів або тренерів. Найвідоміші франшизи – FIFA і F1, які щороку оновлюються з урахуванням реальних змін у світі спорту.

Симулятори (Simulators) – максимально наближені до реальності, відтворюють життєві процеси або технічні системи. Наприклад, The Sims дозволяє керувати життям персонажів, а Flight Simulator – відчути себе пілотом у реалістичному середовищі.

Стратегії (Strategy) – зосереджені на плануванні, управлінні ресурсами та тактичних рішеннях. Такі ігри, як StarCraft або Civilization, вимагають стратегічного мислення та далекоглядного планування для досягнення успіху.

Ігри-пісочниці (Sandbox games) пропонують відкритий світ і високий рівень свободи дій. Гравці можуть самостійно формувати власний досвід, експериментувати з механіками й створювати унікальні сценарії. Minecraft і Grand Theft Auto є класичними прикладами цього типу, що поєднують дослідження, творчість і нелінійність.

Геймплей визначає спосіб взаємодії гравців із грою та механіку, яка формує цю взаємодію. Саме геймплей забезпечує унікальний ігровий досвід, який може суттєво відрізнятися від гри до гри, впливаючи на сприйняття, емоційне залучення та задоволення гравців. Одні ігри роблять акцент на сюжетному зануренні, інші – на технічних викликах, стратегічному мисленні або взаємодії між гравцями.

Нижче наведено основні типи ігрового процесу:

Одиночна гра (Single-player) призначена для одного гравця й зазвичай відзначається глибокими сюжетами та ретельно розробленими персонажами. Вона дає змогу повністю зануритися у світ гри без зовнішнього втручання. Серії Uncharted або Life is Strange є яскравими прикладами одиночних проєктів із виразним наративом.

Багатокористувацькі ігри (Multiplayer) дозволяють одночасну участь кількох гравців у кооперативному або змагальному форматі. Такий тип геймплею стимулює соціальну взаємодію, командну співпрацю або суперництво. Unravel Two

і A Way Out демонструють кооперативний підхід, тоді як Fortnite і League of Legends представляють конкурентний багатокористувацький досвід.

Онлайн-ігри (Online games) функціонують через Інтернет і об'єднують гравців по всьому світу. Вони можуть включати як кооперативні, так і PvP-режими, часто доповнюються регулярними оновленнями, сезонними подіями та інтерактивними активностями в реальному часі, що підтримує активність спільноти.

Офлайн-ігри (Offline games) не потребують підключення до мережі, що робить їх зручними для гри будь-де й будь-коли. Такий формат ідеальний для користувачів, які віддають перевагу автономному ігровому процесу без взаємодії з іншими гравцями.

Технологічна еволюція постійно стимулює розвиток інтерактивних розваг, удосконалюючи геймплей за рахунок нових технічних рішень. Ключові інновації включають динамічні середовища, системи штучного інтелекту, гіперреалістичний рендеринг, кросплатформну сумісність і хмарні сервіси. Ці досягнення покращують процес розробки, користувацький досвід і довгострокову взаємодію, забезпечуючи ітеративне оновлення контенту та створення дедалі реалістичніших і захопливіших віртуальних світів.

Таким чином, сучасна ігрова індустрія характеризується великою різноманітністю жанрів, стилів геймплею та технічних підходів до розробки. Зростання рівня інтерактивності, поява нових технологій і розвиток інтелектуальних систем створюють умови для інтеграції інноваційних рішень у процес створення ігор. Особливої уваги заслуговує застосування алгоритмів машинного навчання, що відкривають можливості для динамічної адаптації ігрового процесу, персоналізації досвіду гравців та створення більш реалістичної поведінки ігрових агентів.

1.3 Огляд сучасних підходів та аналогів

У межах аналізу розглянуто низку сучасних ігор, у яких застосовуються методи штучного інтелекту та машинного навчання. Хоча ступінь і форма

інтеграції цих технологій у кожному випадку різняться – від генеративних моделей і динамічних наративів до поведінкових дерев, систем адаптації ворогів чи складних агентних моделей – отримані результати дозволили визначити широкий спектр можливостей, що відкриваються завдяки інтелектуальним підходам. Досліджувані приклади охоплюють як проєкти з глибокою МН-складовою, так і традиційні ігри з розвиненими системами ШІ, що забезпечує всебічне розуміння технік, застосовних для проєктування ігрового застосунку.

Розглянуті приклади охоплюють такі ігри, як AI Dungeon (табл. 1.1), Dragon’s Dogma 2 (табл. 1.2), Middle-earth: Shadow of Mordor (табл. 1.3), The Elder Scrolls IV: Oblivion (табл. 1.4), The Witcher 3: Wild Hunt (табл. 1.5), які демонструють різні підходи до використання ШІ та методів МН – від генерації сюжетів і поведінки персонажів до симуляції суперників та емоційних моделей.

Рисунок 1.1 – Аналоги створюваного ігрового застосунку

Таблиця 1.1 – Основні характеристики AI Dungeon

Назва	AI Dungeon
Розробник	Latitude (Nick Walton)
Архітектура	Web / Mobile / Desktop application
Мова реалізації	Python, JavaScript
Функції	– генерація текстових пригод за допомогою ШІ; – можливість створення власних сюжетів та персонажів; – мультиплеєрний режим для спільної гри; – інтеграція з різними платформами для збереження прогресу; – спільнота користувачів для обміну створеними історіями.
Переваги	– безмежні можливості для творчості та експериментів; – легка доступність через веб-браузери та мобільні пристрої; – активна спільнота та регулярні оновлення.
Недоліки	– іноді непередбачувані або нелогічні відповіді від ШІ; – обмежена графіка та візуальні ефекти; – потребує стабільного інтернет-з’єднання для оптимальної роботи.
Посилання	https://aidungeon.com/

AI Dungeon є прикладом застосунку, що реалізує генерацію інтерактивних сюжетів за допомогою моделей штучного інтелекту. Гра демонструє можливості текстових нейромереж у створенні нелінійних історій та динамічної взаємодії з користувачем, що робить її показовою для аналізу систем, побудованих на генеративних мовних моделях.

Таблиця 1.2 – Основні характеристики Dragon’s Dogma 2

Назва	Dragon’s Dogma 2
Розробник	Capcom
Архітектура	ПК / PlayStation / Xbox / Switch
Мова реалізації	C++
Функції	– ШІ-напарники, що адаптуються до стилю гри користувача; – різноманіття ворогів з унікальною поведінкою; – відкритий світ із циклом дня та ночі; – онлайн-взаємодія для обміну Pawn-помічниками між гравцями; – Pawn-помічники, які дають поради, знаходять маршрути, активують події та реагують на дії гравця.
Переваги	– глибока й тактична бойова система; – висока якість анімацій і бойових ефектів; – значна реіграбельність завдяки класам, поведінці Pawn та випадковим подіям у світі.
Недоліки	– непередбачувана поведінка Pawn у складних бойових ситуаціях; – обмежена кількість точок швидкого переміщення; – нерівномірний баланс складності, особливо вночі та під час боїв із великими ворогами.
Посилання	https://www.dragonsdogma.com/

Dragon’s Dogma 2 демонструє роботу розширеної поведінкової AI-системи Pawn, яка забезпечує адаптацію напарників до стилю гри користувача та впливає на дослідження світу і бойові взаємодії. Завдяки поєднанню відкритого середовища, складної бойової моделі та адаптивних ШІ-механік ця гра є релевантним прикладом для вивчення поведінкових агентів у 3D-action-RPG.

Таблиця 1.3 – Основні характеристики Middle-earth: Shadow of Mordor

Назва	Middle-earth: Shadow of Mordor
Розробник	Monolith Productions
Архітектура	ПК / PlayStation / Xbox
Мова реалізації	C++
Функції	– система Nemesis, що відстежує результати сутичок і оновлює структуру ворожої ієрархії; – відкритий світ із поділом на регіони та доступом до різних типів завдань; – система розвитку героя з окремими гілками бойових, стелс- і Wraith-умінь; – цикл дня і ночі, що впливає на активність ворогів; – різні типи ворогів із власними поведінковими моделями та характерними слабкостями.
Переваги	– якісна реалізація паркуру та переміщення; – висока реіграбельність; – автономний розвиток фракцій ворогів.
Недоліки	– повторювані побічні завдання; – обмежений вибір локацій та однотипність середовища; – надмірна сила деяких навичок у пізній грі, що спрощує проходження.
Посилання	http://www.shadowofmordor.com/

Shadow of Mordor відзначається інноваційною системою Nemesis, яка формує унікальні взаємодії між гравцем та ворожими капітанами. Механізм автономного розвитку ворогів та зміни їхнього статусу робить гру важливим прикладом застосування ШІ-алгоритмів у формуванні динамічної ієрархії персонажів.

Таблиця 1.4 – Основні характеристики Oblivion

Назва	The Elder Scrolls IV: Oblivion
Розробник	Bethesda Game Studios
Архітектура	ПК / Xbox / PlayStation
Мова реалізації	C++
Функції	– NPC мають цілі та рутини, виконують завдання відповідно до власного розкладу; – відкритий світ із великим простором для досліджень (міста, села, підземелля); – гнучка система квестів та можливість приєднатися до різних фракцій; – свобода розвитку персонажа через систему навичок і спеціалізацій; – динамічний цикл дня та ночі.
Переваги	– можливість обрати різні стилі проходження (рольова гра, бойова, дослідження); – різноманітність NPC та побічних активностей; – можливість повторного проходження.
Недоліки	– нерівномірна деталізація локацій і персонажів; – обмежена глибина бойової системи та логіки ШІ ворогів; – emergent-ситуації трапляються не завжди передбачувано.
Посилання	https://elderscrolls.bethesda.net/en-EU/oblivion-remastered

Oblivion демонструє класичну ШІ-систему для відкритого світу: NPC «живуть своїм життям», мають власні цілі та розпорядок дня, що формує відчуття інтерактивного середовища та забезпечує реалістичну поведінку персонажів у світі гри.

Таблиця 1.5 – Основні характеристики The Witcher 3: Wild Hunt

Назва	The Witcher 3: Wild Hunt
Розробник	CD Projekt Red
Архітектура	ПК / PlayStation / Xbox / Nintendo Switch
Мова реалізації	C++
Функції	– NPC і вороги мають унікальну поведінку та тактики бою; – гнучка система квестів, сюжетних ліній та побічних завдань; – динамічний, деталізований відкритий світ із багатьма локаціями; – бойова система з комбінаціями атак, магії та тактичного ухилення; – різноманіття ворогів із унікальною поведінкою та тактикою бою.
Переваги	– живий, атмосферний світ; – велика свобода дій; – варіативність геймплею.
Недоліки	– складність бою і механік для новачків; – нерівномірна деталізація другорядних локацій; – деяка повторюваність контенту.
Посилання	https://www.thewitcher.com/

Аналіз The Witcher 3: Wild Hunt підтверджує ефективність поєднання масштабного відкритого світу, нелінійної системи квестів та гнучкого розвитку персонажа. Використання таких підходів дозволяє створювати ігровий досвід, у якому вибори гравця мають значущі наслідки, що може бути застосовано у розробці ігрового застосунку з адаптивною поведінкою NPC.

Висновки до розділу 1

У результаті проведеного аналізу предметної області встановлено, що сучасні ігрові застосунки потребують інтеграції інтелектуальних алгоритмів та методів машинного навчання для забезпечення більш реалістичної, адаптивної та варіативної поведінки персонажів та ворогів. Особливо це актуально для 3D-ігор у жанрі Action-RPG, де гравець взаємодіє з відкритим світом, приймає стратегічні та тактичні рішення, формує власний стиль проходження та очікує високого рівня занурення.

Аналіз сучасних тенденцій розвитку ігрових застосунків показав, що технології ШІ та МН використовуються не лише для поведінки NPC і супротивників, а й для генерації динамічних сюжетів, процедурного створення контенту, адаптації складності гри, покращення графіки та фізики середовища, а також персоналізації ігрового досвіду. Розглянуто історичний розвиток ШІ та МН, ключові методи навчання агентів, підходи підкріплення, імітаційне навчання та нейроеволюцію, які стали основою для сучасних адаптивних ігрових механік.

Аналіз жанрових особливостей і типів геймплею підтвердив, що успішний ігровий процес потребує врахування відкритих світів, нелінійних сюжетних структур, багатопланових систем квестів, розвитку персонажа та інтерактивності NPC. Такі підходи формують динамічний ігровий процес, що надає гравцеві свободу дій і можливість самостійно визначати розвиток подій.

Огляд сучасних підходів і програмних рішень, а також вивчення аналогів показав широкий спектр способів реалізації адаптивної поведінки, динамічного світу та персоналізованого досвіду гравця. Це дозволяє виділити ключові принципи та підходити, які можна використати при проєктуванні ігрового застосунку.

2 МОДЕЛЮВАННЯ ТА МЕТОДИ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Методи та алгоритми машинного навчання, що використовуються в іграх

Однією з головних переваг використання нейромереж у відеоіграх є створення інтелектуальних NPC-персонажів. Традиційні методи програмування ШІ обмежені у своїх можливостях, і часто NPCs виявляють передбачувану поведінку. Натомість нейромережі дозволяють створювати персонажів, здатних навчатися та адаптуватися до різноманітних ситуацій у грі [25]. Це робить поведінку NPC більш непередбачуваною і реалістичною, що суттєво покращує ігровий досвід.

Ще одним важливим аспектом застосування нейромереж в іграх є можливість покращення графіки. Нейромережі використовують для покращення роздільної здатності текстур, поліпшення освітлення та анімації. Завдяки цьому віртуальні світи більш реалістичними й візуально привабливими.

Машинне навчання є сукупністю технологій і методів штучного інтелекту, головною особливістю яких є здатність системи навчатися в процесі пошуку рішень. Це дозволяє у майбутньому виконувати завдання більш точно й оперативно.

В основі машинного навчання лежить використання статистичних методів для виявлення зв’язків і кореляцій у великих обсягах даних, що дозволяє будувати прогностичні моделі за принципом «причина–наслідок».

У контексті геймплею машинне навчання дозволяє адаптувати гру до особливостей сприйняття конкретного гравця, оптимізуючи ігровий процес. Зокрема, воно допомагає вирішувати такі завдання, як налаштування ігрового балансу, адаптивне керування складністю, вдосконалення рендерингу та фізичних моделей, а також поліпшення системи діалогів [12].

Завдяки машинному навчанню діалоги головного героя з іншими персонажами стають глибшими, природнішими, що сприяє формуванню емоційного зв’язку між гравцем і грою. Для ще більшої персоналізації ігрового

досвіду можна впровадити систему аватарів, які імітуватимуть не лише зовнішність користувача, а й особливості його поведінки й реакцій.

ШІ та МН також широко застосовуються для динамічного налаштування складності гри відповідно до рівня гравця. Наприклад, система може автоматично підвищувати або знижувати складність гри залежно від навичок і досвіду користувача, забезпечуючи при цьому оптимальний баланс виклику та задоволення від гри.

Крім того, ці технології сприяють створенню більш реалістичних та інтерактивних ігрових світів. Алгоритми машинного здатні покращувати анімацію, фізичну взаємодію, поведінку персонажів, що робить віртуальне середовище живим і переконливим.

2.1.1 Загальна характеристика алгоритмів машинного навчання у геймінгу

Машинне навчання охоплює кілька типів або підходів, кожен з яких найкраще підходить для вирішення різних видів проблем. Розуміння цих типів, а також характеру й структури вхідних даних, має вирішальне значення для вибору ефективної моделі.

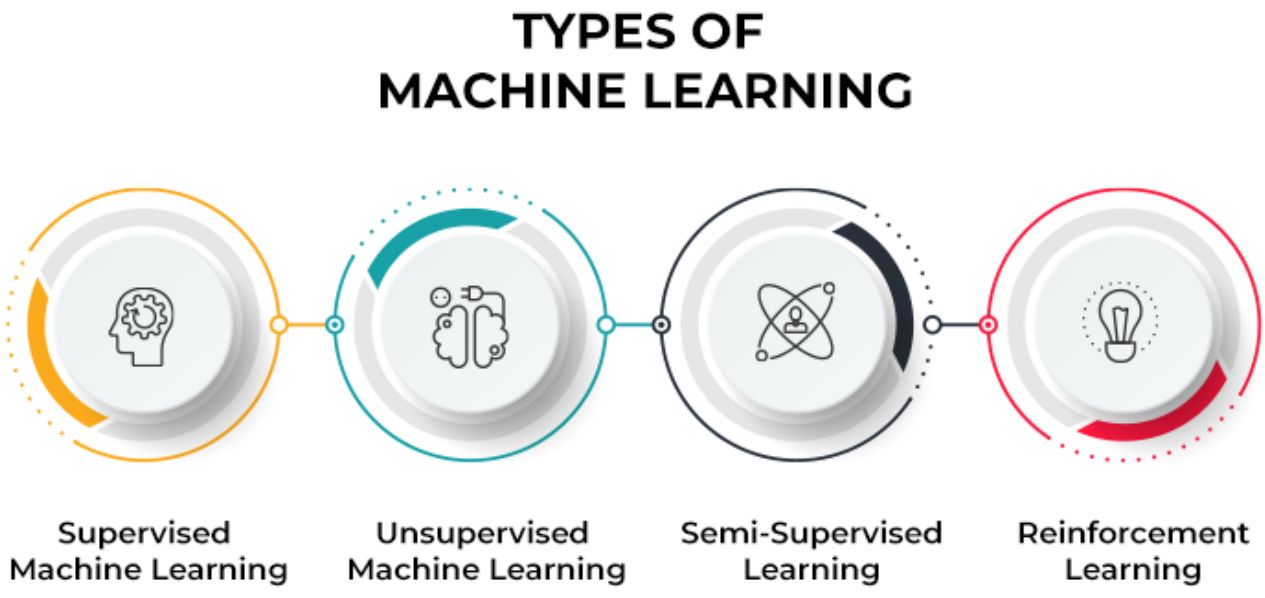


Рисунок 2.1 – Типи машинного навчання

Контрольоване навчання (Supervised Machine Learning) – це тип машинного навчання, при якому модель навчається на маркованих даних: кожному прикладу входу відповідає правильний вихід (мітка або ціль). Мета такого навчання – навчити модель робити точні прогнози на нових, раніше небачених даних, спираючись на закономірності вивчені під час навчання [2].

Модель навчається шляхом порівняння своїх прогнозів із правильними відповідями та поступового налаштування параметрів, щоб мінімізувати помилку.

Наприклад, якщо модель навчається розпізнавати рукописні цифри, їй надаються зображення з відповідними мітками (0–9), після чого вона здатна класифікувати нові зображення цифр.

Етапи навчання моделі:

1) збір і підготовка даних:
   - вхідні ознаки (features) + мітки (labels/targets);
   - обробка пропущених значень, масштабування ознак, очищення даних.
2) розподіл на навчальну та тестову вибірки (зазвичай 80% для навчання, 20% – для тестування);
3) вибір відповідного алгоритму (залежить від типу задачі: класифікація або регресія);
4) навчання моделі (модель оптимізує параметри, щоб краще відповідати навчальним міткам);
5) оцінка якості на тестовій вибірці (використання метрик точності, F1-міри, RMSE тощо);
6) оптимізація моделі (пошук найкращих гіперпараметрів: Grid Search, крос-валідація);
7) фінальне перенавчання та розгортання моделі.

Основні задачі контрольованого навчання:

Класифікація – передбачення категорії. Наприклад: «кішка» або «собака», «спам» чи «не спам».

Алгоритми:

- логістична регресія (Logistic Regression);
- дерева рішень (Decision Trees);
- випадковий ліс (Random Forest);
- машина опорних векторів (SVM);
- k-найближчих сусідів (k-NN);
- наївний Байєс (Naive Bayes);
- нейронні мережі (Neural Networks).

Регресія – передбачення числового значення. Наприклад: прогнозування цін на житло або акції.

Алгоритми:

- лінійна регресія (Linear Regression);
- Lasso / Ridge регресія;
- Support Vector Regression (SVR).

Таблиця 2.1 – Популярні алгоритми контрольованого навчання

Алгоритм	Тип задачі	Короткий опис
Лінійна регресія	Регресія	Прогнозує числові значення на основі лінійної залежності
Логістична регресія	Класифікація	Підходить для задач з двома класами (бінарна класифікація)
Дерева рішень	Обидва	Моделюють рішення у вигляді дерева зі шляхами та вузлами
Random Forest	Обидва	Комбінація дерев рішень для кращої точності
SVM	Обидва	Створює гіперплощини для розділення класів у багатовимірному просторі
k-NN	Обидва	Класифікує новий приклад за найближчими сусідами
Наївний Байєс	Класифікація	Використовує теорему Байєса з припущенням незалежності ознак
Gradient Boosting	Обидва	Поєднує кілька слабких моделей у сильну, коригуючи помилки попередніх моделей

Неконтрольоване навчання (Unsupervised Learning) – це підхід у машинному навчанні, за якого алгоритми працюють із немаркованими даними, тобто без наперед визначених вхідних даних або класів. Основна мета – виявити приховані закономірності, структури або взаємозв'язки в даних без участі людини.

Модель отримує лише вхідні дані і самостійно аналізує їх, знаходячи подібності, відмінності, угруповання чи залежності [2].

Основні задачі неконтрольованого навчання:

Кластеризація – процес групування об'єктів у кластери за схожістю між ними. Широко застосовується для сегментації користувачів, аналізу зображень, рекомендаційних систем.

Поширені алгоритми кластеризації:

– *K-Means* ділить дані на K кластерів на основі близькості до центрів;

– *DBSCAN* виявляє кластери в щільних регіонах, ігноруючи розсіяні точки (як «шум»);

– *ієрархічна кластеризація* створює дерево (дендрограму) схожості між об'єктами;

– *Mean Shift* ітеративно зміщує центри до найбільш щільних регіонів;

– *спектральна кластеризація* використовує теорію графів для виявлення структур у даних.

Навчання правил асоціацій має на меті виявлення частих зв'язків або правил типу «якщо X – тоді Y». Часто використовується для аналізу споживчого кошика в e-commerce.

Основні алгоритми:

– *Apriori* виявляє часті набори елементів шляхом ітеративного аналізу;

– *FP-Growth* швидший за Apriori, уникає генерації всіх можливих комбінацій;

– *Eclat* працює на основі перетинів множин.

Зменшення розмірності скорочує кількість ознак у даних при збереженні найбільш інформативних характеристик. Використовується для візуалізації, очищення від шуму, прискорення обчислень.

Основні методи:

– *PCA (Principal Component Analysis)* проєктує дані на нові осі з максимальною дисперсією;

– *LDA (Linear Discriminant Analysis)* враховує мітки (напівконтрольований метод) і підходить для задач класифікації;

– *t-SNE* – нелінійний метод для візуалізації багатовимірних даних;

– *Autoencoders* – нейронні мережі, що навчаються стискати та відновлювати дані;

– *Isomap, LLE (Locally Linear Embedding)* зберігають локальні або глобальні структури простору даних.

Виявлення аномалій (Anomaly Detection) – пошук даних, що суттєво відрізняються від загального розподілу. Він застосовується у фінансовому моніторингу, кібербезпеці, контролі якості тощо.

Поширені методи:

– Isolation Forest;

– One-Class SVM;

– LOF (Local Outlier Factor).

Представлені алгоритми можна використовувати для сегментації клієнтів у маркетингу, рекомендаційних систем (наприклад, Amazon), аналізу зображень та відео, біоінформатики (групування генів, клітин) та виявлення шахрайства.

Навчання з підкріпленням (Reinforcement Learning, RL) – це розділ машинного навчання, який зосереджується на тому, як агенти можуть навчитися приймати рішення шляхом проб і помилок для максимізації сукупної винагороди. RL дозволяє агентам навчатися, взаємодіючи з навколишнім середовищем, та

отримувати зворотний зв’язок у вигляді нагород (позитивного підкріплення) або штрафів (негативного підкріплення) [16].

Основна ідея полягає в тому, що агент перебуває в певному середовищі, де виконує дії (action), отримує стан середовища (state), отримує нагороду або покарання (reward), оновлює свою стратегію поведінки (policy) на основі отриманого досвіду.

З часом агент вчиться вибирати такі дії, які забезпечують найбільший довгостроковий виграш, навіть якщо короткочасно це може не приносити нагороди. Це вимагає балансу між дослідженням нового (exploration) і використанням вже вивченого (exploitation).

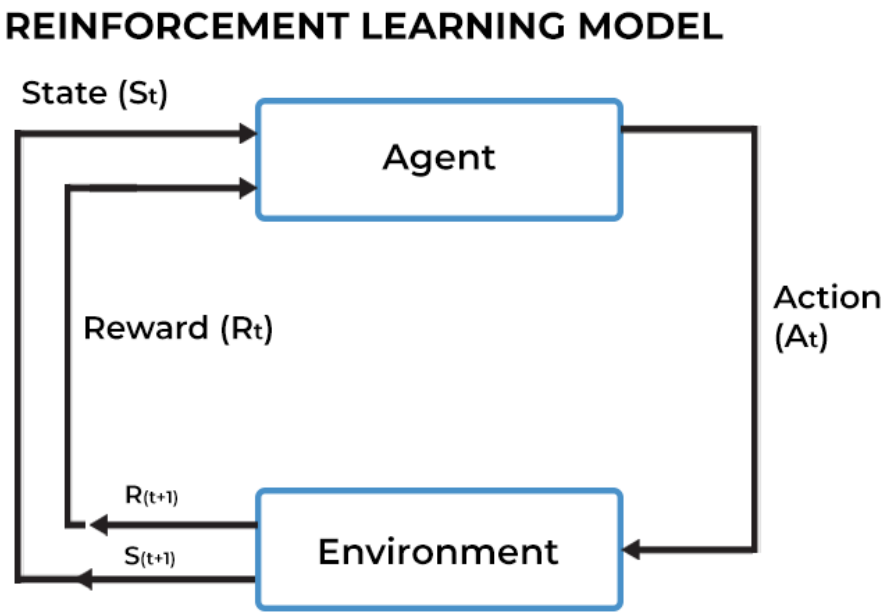


Рисунок 2.2 – Схема взаємодії у Reinforced Learning

Опис схеми взаємодії (рис. 2.2):

Агент → виконує дію в середовищі.

Середовище → змінюється, повертає новий стан і винагороду.

Агент → оновлює свою стратегію відповідно до отриманого досвіду.

Приклад застосування навчання з підкріпленням:

Компанія DeepMind (підрозділ Google) використала навчання з підкріпленням для створення агента, який навчився грати в гру Quake III Arena на

рівні, близькому до людського. Середовище гри з чіткими, але складними правилами ідеально підходить для тестування RL–алгоритмів.

Популярні алгоритми:

а) Q–learning – табличне навчання дій у кожному стані;

б) SARSA – схоже на Q–learning, але враховує обрану дію;

в) Deep Q–Networks (DQN) – поєднання Q–learning з нейронними мережами;

г) Policy Gradient – безпосереднє навчання стратегії (policy);

д) Actor–Critic, A3C, PPO – сучасні методи, що поєднують переваги різних підходів (широко використовуються в іграх, робототехніці, автономних агентах).

Напівконтрольоване навчання (Semi-Supervised Learning) – це підхід у машинному навчанні, який поєднує елементи контрольованого (supervised) та неконтрольованого (unsupervised) навчання. Він використовує невелику кількість мічених (маркованих) даних разом із великою кількістю немічених для навчання моделі [2].

Метою є навчання моделі передбачати вихідні змінні на основі вхідних, подібно до контрольованого навчання, але при цьому ефективно використовувати немарковані дані, щоб зменшити потребу в ручному маркуванні, яке може бути дорогим чи складним.

Переваги:

– економить ресурси, адже маркування великої кількості даних часто трудомістке або дорого коштує;

– дозволяє покращити точність моделі, використовуючи доступні великі масиви немаркованих даних.

Приклади застосування:

Навчання моделі за допомогою невеликої кількості мічених текстів і великої кількості немічених – наприклад, для фільтрації спаму чи категоризації новин.

Використання частини мічених зображень (наприклад, медичних знімків) разом із великою кількістю немічених для покращення точності розпізнавання.

Навчання моделі виявляти незвичні або відхилені від норми спостереження, маючи лише кілька прикладів «аномалій».

У розробці ігрового застосунку було застосовано методи машинного навчання для реалізації **динамічного налаштування складності гри**. Основна мета – забезпечити адаптивність ігрового процесу до індивідуального рівня навичок гравця, покращуючи загальне враження та залученість.

Для навчання алгоритму використовувалися такі ігрові показники:

- середній час реакції гравця на атаку супротивника;
- відсоток влучань (accuracy) у ворогів;
- кількість втрат здоров’я на рівень;
- частота використання спеціальних умінь;
- час проходження рівня;
- кількість поразок за останні 3 бої.

Ці дані накопичувались протягом проходження гравцем рівнів і зберігалися локально.

Було реалізовано модель класифікації з використанням алгоритму k-Nearest Neighbors (k-NN), який аналізує поведінку гравця та відносить її до одного з заздалегідь визначених класів складності:

- «Новачок» (низький рівень складності);
- «Середній гравець»;
- «Просунутий гравець».

Після кожного рівня модель переобчислює клас гравця і відповідно коригує параметри гри.

Оскільки гра виконується локально, навчання реалізоване у вигляді offline–моделі з фіксованими класами і відстанями до центроїдів кластерів. Модель не оновлюється під час гри, але виконує класифікацію кожного разу на основі нових зібраних даних.

У внутрішньому тестуванні моделі було використано:

а) точність класифікації (accuracy) – відношення правильно класифікованих рівнів складності до загальної кількості;

б) середній час проходження рівня після адаптації (мета – стабілізація на середньому значенні);

в) рівень відтоку гравців після першого рівня (менше 15%);

г) суб’єктивна оцінка гравців (через внутрішнє опитування) про адекватність рівня складності.

Результат класифікації безпосередньо впливає на такі параметри:

– кількість супротивників на рівні;

– їхня швидкість, рівень здоров’я та шкода;

– тривалість таймерів для ухилення чи контратак;

– доступність підказок (відображаються лише для «Новачка»).

Адаптація відбувається між рівнями, щоб уникнути непередбачуваності під час активного бою.

Ще одним напрямком застосування машинного навчання в ігровому застосунку стала реалізація **адаптивної поведінки неігрових персонажів (NPCs)**, які виступають у ролі супротивників. Мета – створити враження «розумних» ворогів, які реагують на дії гравця не за фіксованими шаблонами, а гнучко змінюють стратегію залежно від обставин.

Для навчання поведінки супротивників було обрано алгоритм Q-навчання (Q-learning) – один із найпростіших, але ефективних алгоритмів машинного навчання з підкріпленням (reinforcement learning).

NPC сприймає ігрову сцену як середовище, в якому він

– обирає дію (атакувати, захищатись, відступити, викликати підмогу);

– отримує за неї нагороду або штраф (наприклад, за влучний удар – +10, за отримання шкоди – -5);

– оновлює таблицю Q-значень (Q-table), щоб у майбутньому приймати ефективніші рішення.

Станові змінні середовища включали:

– відстань до гравця;

– поточний рівень здоров’я NPC;

– наявність укриттів;

– час від останньої атаки;

– агресивність поведінки гравця.

Навчання NPC проводилось у симуляціях на внутрішньому рівні, із запуском 1000+ боїв у прискореному режимі. Кожен NPC мав свою Q-таблицю, яка поступово збільшувала ефективність прийняття рішень.

У фінальній реалізації таблиця не оновлюється під час гри – модель попередньо навчена, а значення ретельно закодовані. Це дозволяє уникнути додаткових обчислювальних витрат під час активного геймплею.

Метрики оцінювання:

– Win rate NPC проти гравця (у межах 30–60%, щоб уникнути занадто слабких або занадто сильних ворогів);

– кількість повторень одних і тих самих дій – менше 20% (оцінка «непрограмної» поведінки);

– інтенсивність і різноманіття бою, оцінена тестерами.

NPC приймає рішення кожні 1–2 секунди бою на основі свого поточного стану. ***Наприклад:***

– якщо здоров’я низьке – відступає до союзників;

– якщо гравець часто ухиляється – атакує в затримку;

– якщо гравець у кутку – активізує агресивний режим.

У грі реалізовано **систему інтелектуальної підтримки гравця союзником–компаньйоном**, який адаптує свою поведінку до стилю гри користувача. Метою є не просто створити автоматичну бойову одиницю, а сформувати помічника, який відчуває ритм та пріоритети гравця.

Було використано алгоритм кластеризації K-means, що дозволив аналізувати стиль гри користувача та визначати оптимальні шаблони дій для союзника.

Вхідні параметри:

– частота атак гравця (агресивність);

– частота використання умінь і їх тип (наприклад, підтримка/атака);

– середня дистанція до ворога під час бою (ближній/дальній бій);
– пріоритет цілей (бос, слабкий ворог, маг, тощо);
– кількість використаних лікувальних предметів;
– тривалість боїв.

Ці параметри регулярно оновлюються протягом гри та фіксуються після завершення кількох боїв.

Гравці класифікуються на 3 основні стилі:

– агресивний – активна атака, мало захисту;
– тактичний – баланс атаки, дистанційної боротьби й ухилень;
– оборонний – часте лікування, переважно захист.

Союзник обирає шаблон поведінки, що відповідає поточному кластеру:

а) для агресивного гравця – союзник прикриває в тилу, допомагає блокувати атаки;
б) для оборонного – бере на себе удар;
в) для тактичного – атакує паралельно з гравцем.

Після кожних трьох боїв проводиться оновлення кластера. Якщо стиль гравця змінився, союзник автоматично переходить на інший шаблон.

Метрики ефективності:

– середній час виживання союзника (має корелювати з гравцем, а не бути постійно мертвим або невразливим);
– суб’єктивна користь для гравця (опитування: чи корисний союзник?);
– синергія дій (наприклад, атакують одну ціль чи працюють злагоджено).

Союзник діє автономно, але його поведінка виглядає так, ніби він підлаштовується під гравця. Це підвищує емоційну залученість та дозволяє уникнути ситуацій, коли союзники діють неприродньо або неефективно.

Цей метод чудово доповнює попередні два, що стосуються динамічної складності та адаптивної поведінки неігрового персонажа, і вони разом створюють відчуття «живої» гри, що реагує на гравця.

Загалом, впровадження ШІ та машинного навчання в розробку ігор відкриває нові горизонти для індустрії, дозволяючи створювати більш глибокий, захопливий і якісний ігровий контент. Завдяки стрімкому розвитку цих технологій та зростанню інтересу до них, можна очікувати, що вплив ШІ та МН на геймдев не лише зростатиме, трансформуючи індустрію в майбутньому.

Unity Machine Learning Agents (ML-Agents) – це набір інструментів з відкритим вихідним кодом, який дозволяє розробникам створювати середовища, у яких штучні агенти навчаються складній поведінці за допомогою методів навчання з підкріпленням. Такий підхід ідеально підходить для моделювання реалістичної поведінки NPC, симуляцій або адаптивних систем складності гри [9].

Цей фреймворк долає розрив між машинним навчанням і розробкою ігор, забезпечуючи Python API, який взаємодіє з рушієм Unity, що дозволяє тренувати інтелектуальних агентів за допомогою сучасних алгоритмів навчання з підкріпленням без потреби у глибоких знаннях штучного інтелекту.

ML-Agents підтримує кілька підходів до навчання, серед яких [14]:

– навчання наслідування (imitation learning) – тренування агентів на прикладах поведінки;

– навчання за навчальною програмою (curriculum learning) – поступове підвищення складності завдань;

– багатоагентне навчання (multi-agent learning) – конкурентна або кооперативна взаємодія між кількома сутностями.

Типові сфери застосування ML-Agents включають навчання NPC реалістичній поведінці, оптимізацію контролерів персонажів, створення автономних транспортних засобів, адаптивних систем складності гри та інтелектуальних агентів для промислових симуляцій.

2.1.2 Математичний апарат

Одним із найпоширеніших сучасних методів RL, реалізованих у ML-Agents, є Proximal Policy Optimization (PPO). Його основу становить архітектура «актор–критик», запропонована Р. Сатоном та співавторами ще у 1980-х роках [15].

У цій архітектурі:

– актор (actor) приймає рішення щодо дій і формує політику π(a|s);

– критик (critic) оцінює якість цих дій, обчислюючи функцію цінності V(s), тобто очікувану суму майбутніх винагород із певного стану.

Таке розділення дозволяє збалансувати дослідження нових стратегій і експлуатацію вже вивчених, що забезпечує стабільність процесу навчання.

Припустимо, що агент знаходиться у певному стані. Цінність цього стану розраховується як сума всіх майбутніх винагород, отриманих після виконання дії з цього стану. При такому сумуванні зазвичай застосовується коефіцієнт дисконтування $\gamma<1$, який відображає більшу цінність негайних винагород порівняно з віддаленими.

Замість безпосереднього використання дисконтованих винагород для оновлення політики актора, критик оцінює значення стану V(s) і використовує його для розрахунку переваги A (2.1):

$$A = \sum_{k=0}^{N} \gamma^k * R_{t+k} - V(s), \quad (2.1)$$

де **A** – міра того, наскільки конкретна дія в цьому стані була кращою або гіршою, ніж середнє очікування (value);

$\boldsymbol{R_{t+k}}$ – нагорода, отримана через k кроків після часу t;

$\boldsymbol{\gamma^k}$ – коефіцієнт зниження важливості майбутніх нагород ($0 < \gamma \leq 1$);

V(s) – оцінка середньої очікуваної винагороди з цього стану.

Використання переваги для оновлення політики зменшує дисперсію оцінок і робить навчання більш стабільним.

Однією з ключових частин алгоритму PPO є сурогатна функція втрат, яка визначає, як оновлюються параметри нейронних мереж актора та критика. Обидві

мережі мають власні функції втрат, оскільки виконують різні ролі у процесі навчання [15].

Метою навчання є оновлення параметрів мережі θ таким чином, щоб збільшити ймовірність вибору дій, які приносять вищу винагороду. Це здійснюється шляхом мінімізації функції втрат за допомогою методу градієнтного спуску.

Функція втрат критика оцінює, наскільки точно функція цінності V(s) передбачає очікувану суму дисконтованих винагород.

Функція втрат політики складається з двох компонентів:

– усіченої (clipped) цільової функції;

– функції ентропії, що сприяє збереженню різноманітності дій.

Термін clip (обрізання) обмежує надмірні зміни політики, що забезпечує стабільність навчання. Основна усічена цільова функція задається як (2.2):

$$L_{CLIP}(\theta) = E[\min(r_t * A_t, clip(r_t, 1-\varepsilon, 1+\varepsilon)A_t)] \; with \; r_t = \frac{\pi(a_t|S_t)}{\pi_{old}(a_t|S_t)}, \quad (2.2)$$

де $\boldsymbol{r_t} = \frac{\boldsymbol{\pi(a_t|S_t)}}{\boldsymbol{\pi_{old}(a_t|S_t)}}$ – відношення нової політики до старої;

$\boldsymbol{A_t}$ – оцінка переваги (advantage);

$\boldsymbol{\varepsilon}$ – коефіцієнт обмеження (зазвичай 0.1–0.3), що визначає, наскільки сильно може змінюватися політика між ітераціями.

Додатково, у PPO використовується ентропійна складова, яка стимулює агента підтримувати баланс між дослідженням нових стратегій та використанням уже відомих. Вона запобігає передчасній детермінованості політики та підвищує ефективність навчання [15].

Ентропія політики вимірює рівень невизначеності у виборі дій агентом. Чим вища ентропія, тим більш різноманітними є дії агента, що сприяє дослідженню нових стратегій і стабілізує процес навчання. Максимізація ентропії допомагає PPO підтримувати оптимальний баланс між дослідженням та експлуатацією, роблячи навчання більш надійним і ефективним.

Функція ентропії політики визначається як (2.3):

$$L_{ENTROPY}(\pi) = \sum_a \pi(a|s_t) * \log(\pi(a|s_t)), \quad (2.3)$$

де $\boldsymbol{\pi(a|s_t)}$ – ймовірність вибору дії a у стані s_t.

Таким чином, сурогатна функція втрат у PPO поєднує елементи цінності стану, усіченої функції політики та ентропії, забезпечуючи стабільне й ефективне оновлення параметрів нейронних мереж актора й критика.

Завдяки цій структурі PPO дозволяє агентам навчатися адаптивній поведінці в складних середовищах, ефективно балансуючи між дослідженням нових стратегій і використанням вже відомих. Це робить алгоритм одним із найпопулярніших підходів для реалізації інтелектуальної поведінки NPC у сучасних ігрових застосунках і симуляціях.

2.2 Специфікація вимог до програмного забезпечення

Призначення системи: створення багатого та динамічного ігрового досвіду, який поєднує захоплюючі дії з глибокими елементами рольової гри та адаптивним штучним інтелектом. Система дозволяє гравцям поринати в епічні пригоди, взаємодіяти зі світом, приймати осмислені рішення та формувати власну долю в ігровому світі. Адаптивна поведінка ворогів реалізується за допомогою алгоритмів машинного навчання, що дозволяє їм навчатися на поведінці гравця та підлаштовувати стратегії у реальному часі.

Область застосування: індустрія розваг і комп’ютерних ігор, розробка інтерактивних симуляцій і тренажерів, тестування ігрових сценаріїв та ШІ-поведінки персонажів.

Характеристики користувачів: користувачі віком від 18 років з ПК та доступом до мережі Інтернет.

Функції системи:

– наявність двох ігрових протагоністів із відмінними стилями проходження;

– реалізація розгалуженої сюжетної структури та відкритого ігрового світу для дослідження;

– адаптивна поведінка супротивників із використанням механізмів навчання на основі дій гравця;

– розвинена система крафту та прогресії персонажів;

– широкий набір навичок і можливість формування комбінованих здібностей;

– процедурна генерація квестів і випадкових ігрових подій;

– інтеграція різноманітних головоломок у ігровий процес;

– динамічна система діалогів і взаємодії з неігровими персонажами;

– механізми формування зв’язків між персонажами та ігровими фракціями;

– реалізація торговельної системи для купівлі та продажу озброєння й ресурсів;

– наявність різних класів і підкласів ворогів, включно з босами;

– підтримка повторного проходження гри з використанням адаптивного штучного інтелекту;

– можливість динамічного налаштування рівня складності;

– реалізація системи зміни погодних умов і циклів дня та ночі;

– взаємодія з елементами навколишнього середовища, зокрема руйнування об’єктів і подолання перешкод.

Вимоги до технічного забезпечення:

– операційна система: Windows 10 або вище;

– процесор: Intel Core i5-6600K або AMD Ryzen 5 1600X;

– відеокарта: NVIDIA GeForce GTX 970 або AMD Radeon R9 390X;

– оперативна пам’ять: не менше 8 ГБ;

– місце на жорсткому диску: не менше 50 ГБ.

Архітектура програмної системи: складається з клієнтської частини.

Системне програмне забезпечення: для розробки ігрового застосунку з використанням алгоритмів машинного навчання необхідно встановити такі системні програмні засоби, як Windows 10 або вище, Unity Engine (версія 2022.1 або новіше), Visual Studio 2019 або новіше для програмування на C#, Blender 2.80

або новіше для створення 3D-моделей та анімацій, а також DirectX 12 або вище для рендерингу та обробки графіки.

Мова та технології розробки програмного забезпечення: для реалізації ігрового застосунку рекомендовано використовувати мову програмування C# у середовищі Unity. Для навчання та керування поведінкою агентів штучного інтелекту доцільним є застосування інструментарію ML-Agents Toolkit. Створення 3D-моделей і анімацій передбачається виконувати за допомогою Blender, тоді як для реалізації фізичних взаємодій та навігації неігрових персонажів рекомендується використовувати Unity Physics і систему NavMesh.

Інтерфейс користувача: користувацький інтерфейс системи повинен бути інтуїтивно зрозумілим і зручним у використанні. Він має забезпечувати ефективне керування персонажем, підтримувати взаємодію з ігровим середовищем та надавати користувачеві зручний доступ до налаштувань ігрового процесу.

2.3 Огляд інструментарію та технологій

Ефективність розробки значною мірою залежить від правильно обраного набору інструментів: від потужних ігрових рушіїв до мов програмування та редакторів контенту. Відповідний технологічний стек забезпечує зручність роботи, оптимізує процеси та дозволяє якісно втілити задум у готовий ігровий застосунок.

У межах даного проєкту було використано такий стек технологій: ігровий рушій Unity, мова програмування C#, інструменти машинного навчання Python, засоби 3D-моделювання та анімації Blender, а також графічний редактор Adobe Photoshop. Сукупність цих інструментів дозволяє ефективно реалізувати ігровий процес, розробити графічні активи та впровадити штучний інтелект для персонажів.

Сучасні відеоігри пройшли значний шлях розвитку – від простих піксельних сцен до складних і деталізованих віртуальних світів. Вирішальну роль у цьому процесі відіграють ігрові рушії, які виступають технологічною основою для створення інтерактивних цифрових продуктів. Ігровий рушій являє собою програмне забезпечення, що надає розробникам комплекс інструментів для

реалізації графіки, фізичних процесів, анімації, обробки користувацького введення, роботи зі звуком і керування поведінкою ігрових персонажів [17].

Unity

Одним із найпоширеніших ігрових рушіїв є Unity, який підтримує розробку ігрових застосунків для більш ніж двадцяти платформ, зокрема персональних комп'ютерів, мобільних пристроїв, ігрових консолей, XR-середовищ і WebGL. Завдяки цьому Unity виступає універсальним рішенням для створення ігор різної складності та спрямування [30].

Важливою складовою екосистеми Unity є ML-Agents Toolkit – проєкт з відкритим вихідним кодом, що дозволяє використовувати ігрові сцени та симуляції як середовище для навчання інтелектуальних агентів. Навчання агентів здійснюється переважно з використанням методів навчання з підкріпленням, які ґрунтуються на взаємодії агента з динамічним середовищем і оптимізації його поведінки на основі системи винагород. Одним із ключових алгоритмів, реалізованих у ML-Agents, є Proximal Policy Optimization (PPO) – сучасний алгоритм глибинного навчання на основі нейронних мереж. Процес навчання виконується у середовищі Python з використанням бібліотек TensorFlow або PyTorch і взаємодіє із запущеним застосунком Unity через сокетне з'єднання, що забезпечує обмін даними в реальному часі [24].

Окрім PPO, інструментарій ML-Agents підтримує імітаційне навчання, нейроеволюційні підходи та інші методи машинного навчання, що значно розширює можливості моделювання поведінки агентів. Навчені агенти можуть застосовуватися для керування поведінкою неігрових персонажів у різних умовах, включаючи багатоагентні та змагальні середовища, а також для автоматизованого тестування ігрових збірок і попередньої оцінки дизайнерських рішень. Це робить ML-Agents ефективним інструментом як для розробників ігор, так і для дослідників у галузі штучного інтелекту, оскільки платформа дозволяє аналізувати поведінку агентів у насичених ігрових середовищах та поширювати отримані результати серед наукової й професійної спільноти.

Крім того, Unity забезпечує тісну інтеграцію зі сторонніми інструментами для створення контенту, зокрема Blender та Adobe Photoshop, що спрощує процес імпорту 3D-моделей і графічних ресурсів без втрати якості. Хмарні сервіси Unity, такі як Unity Collaborate, Unity Teams і Photon Unity Networking, надають можливості для організації командної роботи та синхронної розробки проєктів у режимі реального часу.

C# (C-Sharp)

Об’єктно-орієнтована мова програмування загального призначення, що використовується для розробки широкого спектру програм, включаючи корпоративне програмне забезпечення, відеоігри та мобільні застосунки. Мова була представлена компанією Microsoft у 2000 році, а у складі Visual Studio .NET з’явилася у 2002 році. C# належить до сімейства мов C, до якого також входять C та C++, і має Сі-подібний синтаксис, близький до C++ та Java [3].

Мова надає зрозумілу та читабельну базу для побудови логіки застосунків, приховуючи більшу частину складності внутрішньої реалізації. Стандартизована згідно специфікації ISO/IEC 23270: Інформаційні технології. Мови програмування. C#. C# спочатку створювалась для роботи з фреймворком .NET, який забезпечує кросплатформність, підтримку декількох мова, розвинену бібліотеку класів, різноманітність технологій та високу продуктивність.

C# легко інтегрується з Unity, де використовується як основна мова для створення скриптової логіки ігрових об’єктів. Програми можуть складатися з декількох вихідних файлів, які компілюються разом, забезпечуючи взаємне посилання між ними. Завдяки своїй універсальності, сучасним можливостям, великій спільноті та підтримці Microsoft, C# залишається доступним і для початківців, і для досвідчених розробників [3].

Python

Мова програмування загального призначення, яка широко застосовується в галузі машинного навчання та штучного інтелекту. У рамках розробки ігор Python використовується для навчання ML-агентів у середовищі Unity ML-Agents,

забезпечуючи автоматизацію процесів створення, тренування та оцінки моделей поведінки NPC та ворогів.

Мова надає багатий набір бібліотек і фреймворків для роботи з даними, математичними та статистичними обчисленнями, що дозволяє аналізувати результати тренувань агентів та оптимізувати їх поведінку в ігровому середовищі. Через API бібліотеки ML-Agents Python дозволяє налаштовувати параметри тренування агентів, проводити навчання в реальному часі та інтегрувати готові моделі у Unity [18].

Ключові можливості Python у розробці ігор з ML-агентами включають створення і збереження моделей, генерацію та обробку тренувальних даних, реалізацію алгоритмів навчання з підкріпленням і взаємодію з компонентами ігрового середовища через API. Мова відрізняється відкритою архітектурою, кросплатформністю та гнучкістю, що робить її універсальним і надійним інструментом для наукових і комерційних проєктів, зокрема в ігровому дизайні та симуляціях [23].

Кожна нова версія Python включає вдосконалення продуктивності, оптимізацію роботи з великими обсягами даних та розширення можливостей стандартної бібліотеки, сприяє його широкому застосуванню у сучасних проєктах.

Blender

Програмне забезпечення для 3D-анімації, яке використовується для створення анімаційних фільмів, візуальних ефектів, інтерактивних 3D-додатків та відеоігор. Програма застосовується як у кіностудіях, так і серед розробників ігор [22].

Blender надає розширені функції, що зазвичай зустрічаються у професійних пакетах, таких як Maya чи Autodesk, при цьому він безкоштовний і може працювати практично на будь-якому комп’ютері завдяки відкритій архітектурі.

Ключові можливості Blender включають моделювання, текстурування, анімацію, рендеринг, композитинг та відстеження руху. Програма дозволяє створювати 3D-об’єкти різними методами: видавлювання, редагування сітки, розподіл поверхонь тощо [22].

Особливістю інтерфейсу Blender є його візуальна орієнтованість: під час роботи з моделлю відображаються лише необхідні інструменти, що робить навігацію сценою швидкою і зручною. Blender підтримує як 3D-, так і 2D-анімацію, дозволяючи працювати з різними напрямками без додаткового ПЗ.

Blender забезпечує легку інтеграцію з ігровими рушіями, такими як Unity. Об’єкти можна імпортувати двома способами: безпосередньо через файл .blend або через експорт у формат .fbx. Це дозволяє ефективно використовувати кожен інструмент там, де він найкраще підходить у процесі розробки 3D-ігор [13].

Кожне нове оновлення Blender розширює функціональні можливості програми за рахунок додавання інструментів і утиліт для роботи з різними типами проєктів, зокрема засобів симуляції рідинних потоків, фізичних процесів, систем частинок та спеціалізованих фільтрів для рендерингу. Завдяки цьому Blender виступає універсальним і повнофункціональним середовищем для створення тривимірного контенту.

Adobe Photoshop

Програмне забезпечення для створення та обробки зображень, графічного дизайну й редагування фотографій, розроблене компанією Adobe у 1988 році братами Томасом і Джоном Ноллами. Спочатку програма була орієнтована на комп’ютери платформи Macintosh, однак згодом стала доступною і для операційних систем Windows та macOS.

Photoshop входить до складу пакета Adobe Creative Cloud, який об’єднує низку професійних інструментів для роботи з графічним контентом, зокрема Adobe Illustrator, Photoshop Lightroom та Adobe Dreamweaver, що забезпечує можливість комплексної обробки візуальних матеріалів на різних пристроях.

Програмний продукт призначений передусім для роботи з растровою графікою та підтримує багатошарову структуру з використанням прозорості, масок і фільтрів, які дозволяють модифікувати базове зображення без його руйнування. До шарів можуть застосовуватися різні колірні моделі, зокрема RGB, CMYK, Lab, Duotone та Spot Color. Крім того, Photoshop надає засоби для створення тіней,

візуальних ефектів і реалізації альфа-композитингу, що забезпечує коректне поєднання шарів із прозорістю.

Photoshop підходить для створення графіки та макетів для друкованих матеріалів (газети, журнали, плакати), веб-дизайну, логотипів та іншого цифрового мистецтва. Зображення можна зберігати у різних форматах, включаючи JPEG, PNG, GIF (для Інтернету) та TIFF (для друку).

Завдяки своїй потужності та гнучкості, Photoshop широко використовується в розробці графічних ресурсів для ігор, включаючи текстури, спрайти та інші елементи для Unity та інших ігрових рушіїв.

Висновки до розділу 2

У цьому розділі виконано комплексний аналіз методів та алгоритмів машинного навчання, які застосовуються в ігрових системах. Особлива увага приділена підкріплювальному навчанні та архітектурі «актор-критик» на прикладі алгоритму PPO. Він забезпечує адаптивну поведінку агентів та стабільне оновлення нейронних мереж у складних сценаріях взаємодії в ігровому середовищі. Розглянуто математичний апарат, який лежить в основі алгоритмів, що дозволяє моделювати динамічну поведінку NPC та прогнозувати результати їх дій в різних умовах.

На основі проведеного аналізу алгоритмів були сформульовані вимоги до програмного забезпечення, що включають функціональні та нефункціональні характеристики системи, інтеграцію алгоритмів машинного навчання, реалізацію адаптивної поведінки NPC та забезпечення інтерактивної взаємодії користувача з грою.

Додатково описано інструментарій та технології, необхідні для реалізації поставлених вимог. Розглянуто рушій Unity для інтеграції графіки, фізики, анімації та ШІ; C# для ігрової логіки; Python для навчання ML-агентів та аналізу даних; Blender для створення 3D-моделей та анімацій; Adobe Photoshop для підготовки візуальних елементів.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

3.1 Побудова сценаріїв використання

Короткий use case

Користувач запускає ігровий застосунок на своєму пристрої та переходить до головного меню. У меню він має можливість обрати доступні опції, зокрема створення нової гри або продовження раніше збереженого проходження. У разі вибору опції «Нова гра» користувач переходить до етапу вибору персонажа, яким керуватиме під час гри. Після підтвердження вибору ініціалізується процес генерації ігрового світу, після чого гравець отримує можливість взаємодіяти з навколишнім середовищем та елементами ігрового процесу.

Поверхневий use case

Головний сценарій (успішний)

Користувач запускає ігровий застосунок і в головному меню обирає опцію «Продовжити гру». Система ініціалізує генерацію ігрового світу, завантажує збережені дані гравця та активує адаптивні моделі поведінки неігрових персонажів. Після завершення завантаження гравець може обирати доступні завдання та квести, що передбачають бойові взаємодії, збирання ресурсів, дослідження локацій і спілкування з персонажами.

Поведінка ворогів і компаньйонів динамічно адаптується до стилю гри користувача на основі алгоритмів машинного навчання. Отримані ресурси використовуються для крафту предметів і розвитку персонажа. Досягнувши визначених рівнів прогресії, гравець отримує можливість обирати та покращувати навички, які впливають на подальшу взаємодію з ігровим світом і його адаптивними елементами.

Альтернативні сценарії

Створення нового персонажа

Гравець у головному меню обирає опцію «Нова гра», створює нового персонажа та зберігає прогрес у доступних слотах профілю. Система підтримує використання різних персонажів і комбінацій для подальшого проходження гри.

Зміна налаштувань гри

Гравець обирає опцію «Налаштування» та змінює параметри графіки, звуку, керування та інші системні налаштування відповідно до власних уподобань..

Вихід із гри

Гравець обирає опцію «Вийти з гри». Система зберігає поточний прогрес і коректно завершує роботу застосунку.

Помилка завантаження

У разі вибору опції «Продовжити гру» система може повідомити про помилку завантаження. Гравцеві надається можливість повторити спробу, перезапустити застосунок або, за необхідності, звернутися до технічної підтримки.

Повний use case

Use Case Name: Розпочати нову гру

Scope: System

Level: Ігровий застосунок із використанням алгоритмів машинного навчання

Primary Actor: Гравець

Зацікавлені сторони та їх інтереси

Гравець: зацікавлений в успішному запуску нової гри, комфортному ігровому процесі та динамічно адаптованій поведінці неігрових персонажів і ворогів.

Розробники: зацікавлені в забезпеченні стабільного запуску застосунку, коректної генерації ігрового світу, правильної роботи алгоритмів машинного навчання та ефективної взаємодії системи з користувачем.

Preconditions:

– користувач має доступ до персонального комп’ютера або іншого підтримуваного пристрою;

– на пристрої встановлено ігровий застосунок із підтримкою алгоритмів машинного навчання.

Success Guarantee: гравець розпочинає нову гру на своєму пристрої та отримує доступ до ігрового світу з динамічно адаптованою поведінкою персонажів і середовища.

Main Success Scenario:

1) гравець запускає ігровий застосунок на своєму пристрої;
2) у головному меню гри обирає опцію «Нова гра»;
3) вибирає головного персонажа, яким буде керувати;
4) система створює нову гру з обраним персонажем та генерує світ, враховуючи адаптивну поведінку NPC і ворогів на основі алгоритмів машинного навчання;
5) гравець починає взаємодію з ігровим світом, виконуючи завдання та квести, де поведінка ворогів та NPC динамічно підлаштовується під дії гравця.

Extensions:

а) неможливість розпочати гру внаслідок технічних проблем:

– система відображає повідомлення про помилку та надає рекомендації щодо її усунення;

– гравець повторно запускає ігровий застосунок;

– у разі, якщо проблема не усунута, гравець звертається до служби технічної підтримки.

б) гравець обирає опцію «Налаштування» для зміни графічних параметрів, звукових ефектів, чутливості контролера з інших системних параметрів гри;

в) гравець обирає опцію «Вийти з гри» після збереження прогресу та виходить з гри.

Special Requirements:

– гра повинна мати систему автоматичного збереження прогресу гравця;

– гра повинна включати систему навчання, яка допомагає гравцю ознайомитися з основними елементами гри та керування;

– інтерфейс користувача повинен бути інтуїтивно зрозумілим та підтримувати зручну взаємодію з ігровим світом;

– алгоритми машинного навчання повинні забезпечувати адаптивну поведінку NPC, ворогів та інших елементів ігрового світу, підвищуючи динамічність та персоналізацію досвіду гравця.

Technology and Data Variations List: рекомендується використовувати Blender версії 2.8 та новіше для створення 3D-моделей та анімацій; Unity Engine версії 2021+ для розробки ігрового застосунку; для поведінки NPC/ворогів необхідна інтеграція Unity ML-Agents та ONNX-моделей.

Frequency of Occurrence: частота виконання даного варіанта використання визначається кількістю гравців, які ініціюють запуск нової гри в ігровому застосунку.

Miscellaneous: у подальшому функціональність системи може бути розширена шляхом додавання багатокористувацького режиму та вдосконаленої системи квестів.

3.2 UML-діаграми

UML (Unified Modeling Language) є стандартною мовою моделювання, що використовується для візуального представлення структури та поведінки програмних систем. За допомогою UML-діаграм розробники аналізують архітектуру програмного забезпечення, проєктні рішення та логіку реалізації складних систем. Крім того, UML застосовується для моделювання робочих і бізнес-процесів, що дозволяє формалізувати взаємодію між різними компонентами системи на концептуальному рівні.

У межах UML розрізняють дві основні групи діаграм: структурні та поведінкові. Структурні діаграми відображають статичні елементи системи та зв’язки між ними, тобто описують її внутрішню організацію. Поведінкові діаграми, у свою чергу, зосереджені на динаміці системи та демонструють процеси взаємодії між об’єктами під час виконання певних сценаріїв.

3.2.1 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) призначена для опису функціональних можливостей системи на високому рівні та визначення меж її застосування. Вона демонструє взаємодію між системою та зовнішніми

учасниками (акторами), фокусуючись на тому, які дії система надає користувачам, без деталізації внутрішніх механізмів реалізації [31].

На рис. 3.1 представлено діаграму варіантів використання для ігрового застосунку.

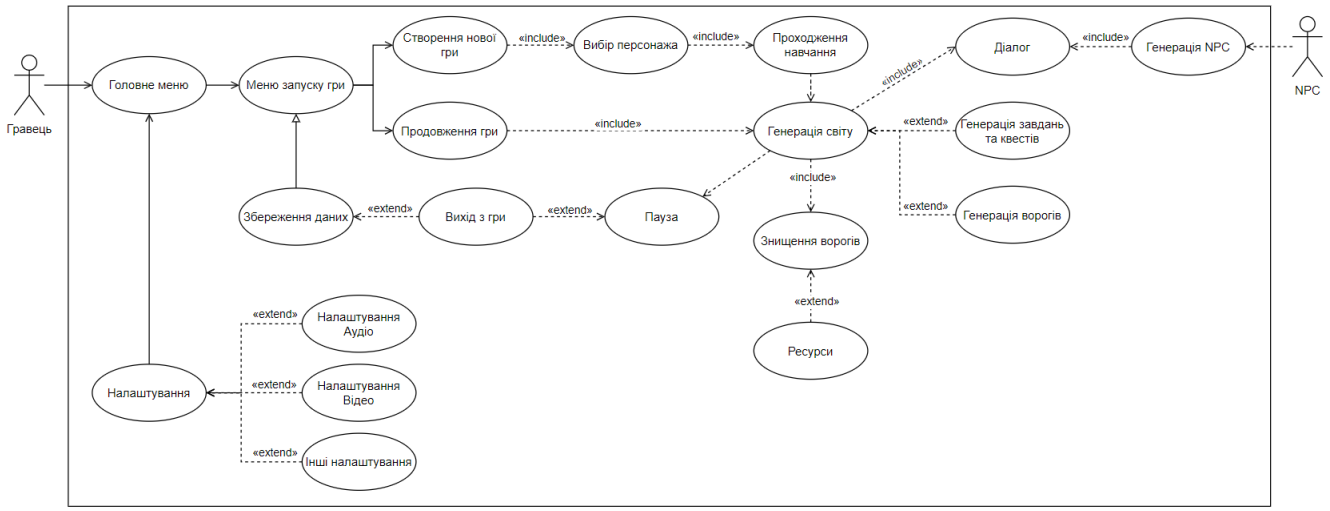


Рисунок 3.1 – Діаграма варіантів використання

Згідно з діаграмою, основним актором є гравець, який має можливість змінювати налаштування звуку та відео, розпочинати нову гру або продовжувати збережене проходження. Після вибору персонажа гравець проходить навчальний етап, необхідний для ознайомлення з базовими механіками гри. У процесі ігрового проходження гравець взаємодіє з неігровими персонажами (NPC), виконує квести та завдання, вступає в бойові сутички з ворогами й отримує ресурси. Система також підтримує збереження прогресу під час виходу з гри.

3.2.2 Діаграма класів

Діаграма класів UML є ключовим елементом об’єктно-орієнтованого проєктування та використовується для опису класів системи, їх атрибутів, методів і взаємозв’язків між ними. Вона дозволяє формалізувати структуру програмної системи та визначити ролі окремих компонентів у її реалізації [5].

Кожен клас на діаграмі подається у вигляді прямокутника, розділеного на три секції: верхня містить назву класу, середня – його атрибути, а нижня – операції або методи, що визначають поведінку об’єктів даного класу.

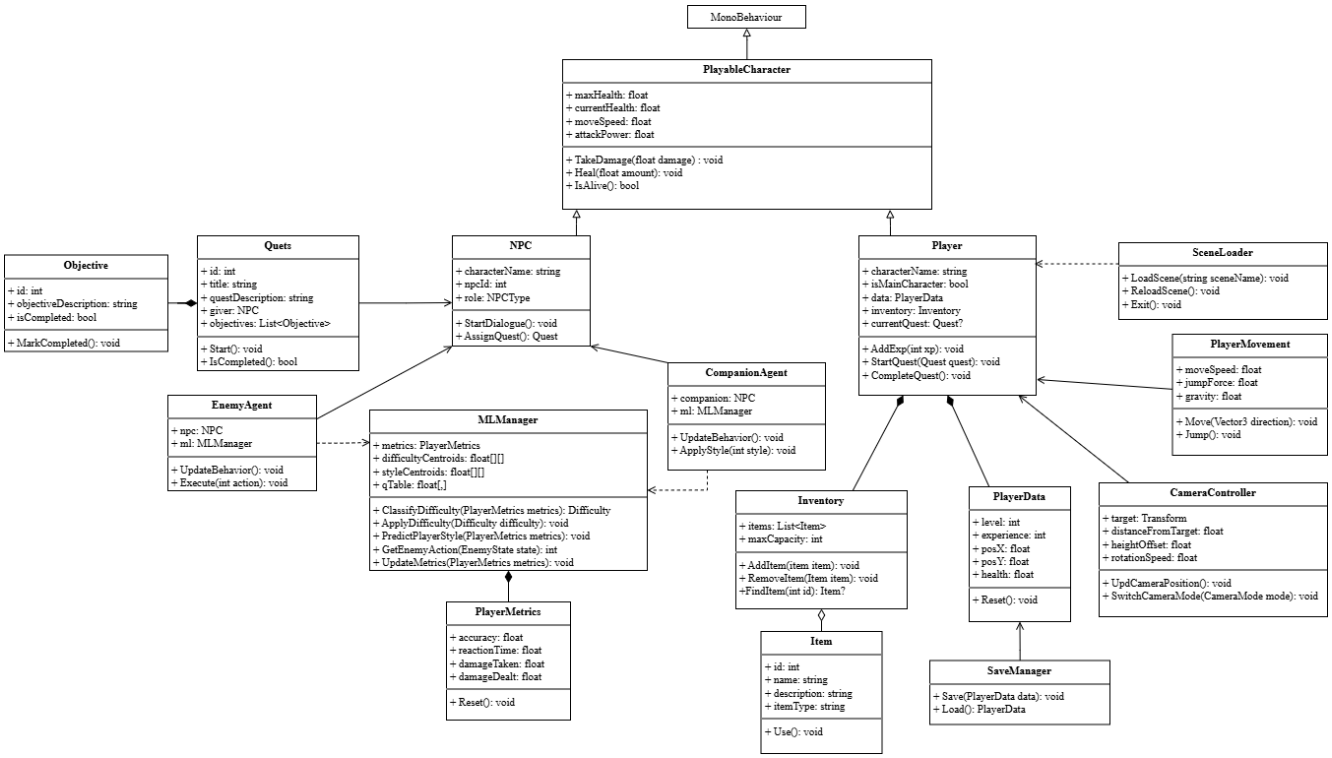


Рисунок 3.2 – Діаграма класів

На діаграмі класів представлені основні компоненти ігрового застосунку, зокрема:

Клас PlayableCharacter представляє базову модель ігрового персонажа з основними характеристиками.

Клас Player описує гравця та його взаємодію з ігровим світом.

Клас PlayerMovement керує рухом персонажа (швидкість, стрибки тощо).

Клас CameraController відповідає за керування камерою, що слідує за гравцем.

Клас SceneLoader здійснює завантаження та перезавантаження ігрових сцен.

Клас EnemyAgent реалізує поведінку ворогів.

Клас BossAgent описує поведінку унікальних босів гри.

Клас CompanionAgent описує союзників гравця.

Клас MLManager забезпечує роботу алгоритмів машинного навчання та адаптацію поведінки NPC.

Клас PlayerMetrics накопичує статистичні дані про дії гравця, необхідні для ML-алгоритмів.

Клас NPC представляє неігрового персонажа, що може видавати квести або взаємодіяти з гравцем.

Клас Inventory керує інвентарем гравця.

Клас Item описує ігровий предмет та його властивості.

Клас Quest представляє квест із цілями та описом.

Клас Objective є окремою ціллю квесту.

Клас SaveManager відповідає за збереження та завантаження прогресу.

Клас PlayerData містить інформацію про стан гравця та його прогрес у грі.

3.2.3 Побудова діаграм взаємодії (послідовності та кооперації)

У UML взаємодія визначається як процес обміну повідомленнями між об’єктами або акторами системи, що відображає динамічну поведінку програмного забезпечення під час виконання певних сценаріїв. Діаграми послідовності (Sequence Diagram) застосовуються для моделювання порядку передачі повідомлень між учасниками системи в межах конкретного варіанта використання, фіксуючи послідовність викликів методів і реакцій об’єктів у часі. Такі діаграми дозволяють детально проаналізувати логіку виконання функцій, уточнити взаємодію між компонентами та доповнити діаграми варіантів використання на рівні реалізації [27].

Діаграма на рис. 3.3 відображає процес бойової взаємодії між гравцем і ворогом: ініціалізацію супротивника, атаку, отримання шкоди, використання предметів лікування, отримання досвіду, підвищення рівня та збір трофеїв.

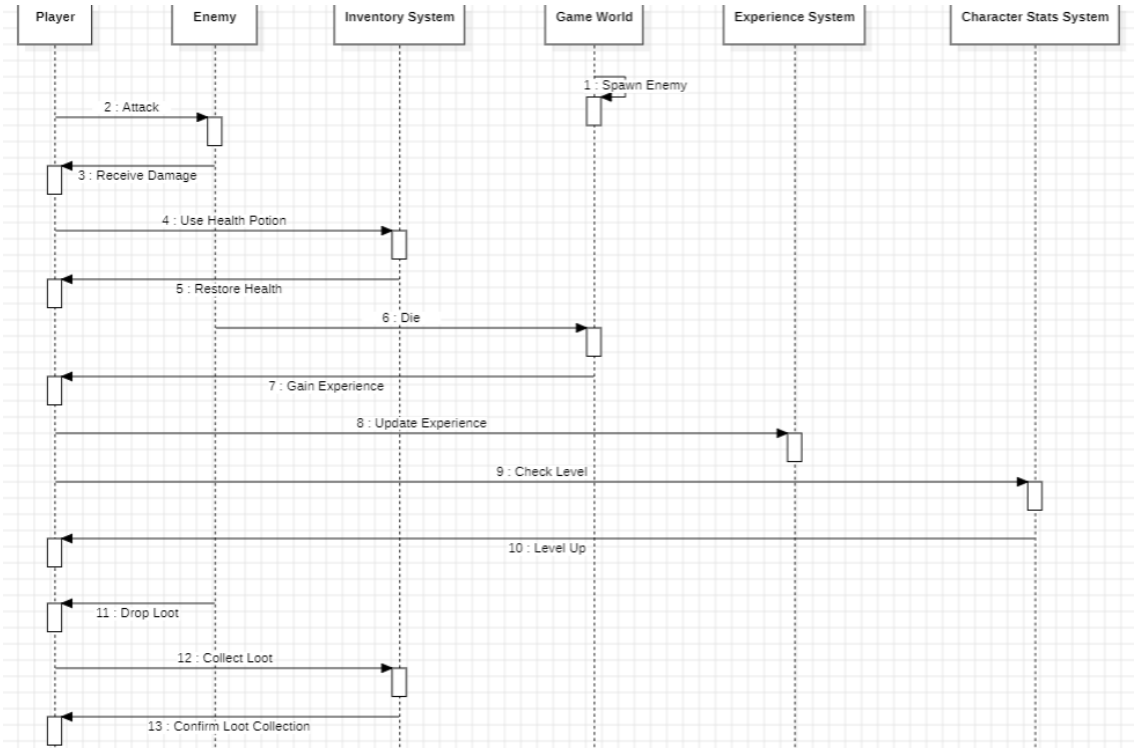


Рисунок 3.3 – Діаграма послідовності для варіанту використання «Combat»

У сценарії на рис. 3.4 показано взаємодію гравця з журналом квестів і NPC, включаючи перегляд активних завдань, прийняття квесту, виконання цілей та отримання винагороди.

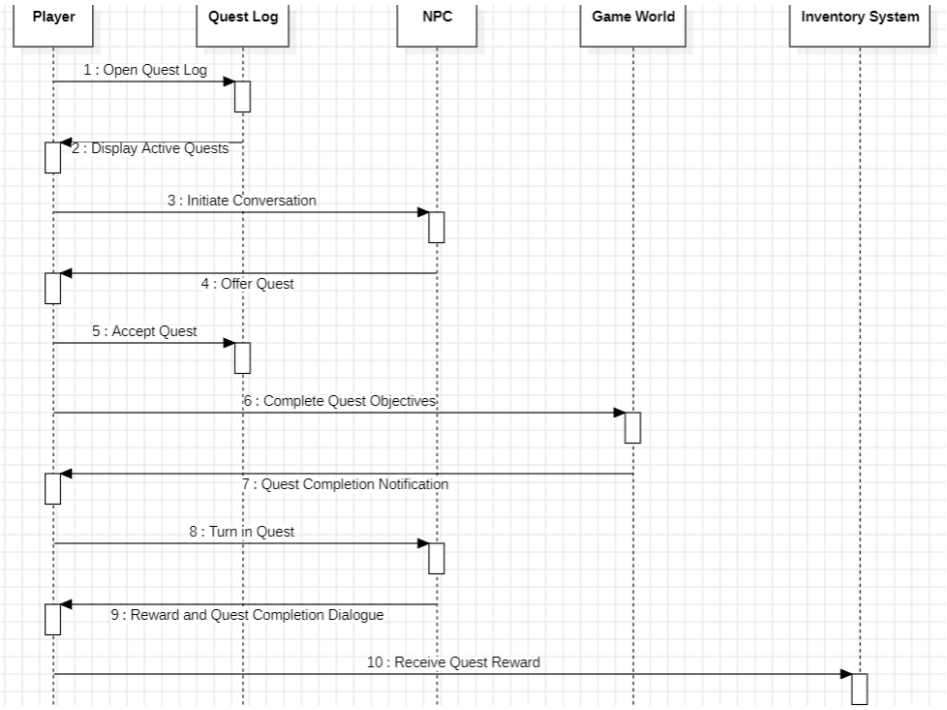


Рисунок 3.4 – Діаграма послідовності для варіанту використання «Quest Completion»

Діаграма, що зображена на рис. 3.5 демонструє процес купівлі предметів у торговця: відкриття інтерфейсу магазину, перевірку інформації про предмет і валютний баланс, підтвердження покупки та додавання предмета до інвентарю.

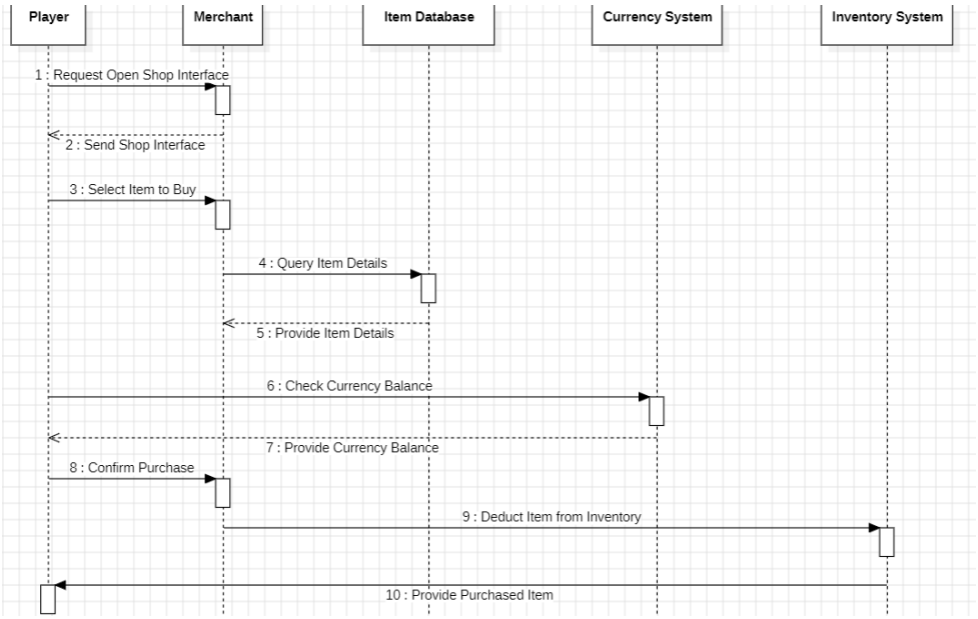


Рисунок 3.5 – Діаграма послідовності «Buying Items from a Merchant»

Сценарій крафту на рис. 3.6 включає взаємодію гравця з майстернею, перевірку рецептів і наявності матеріалів, списання ресурсів та отримання створеного предмета.

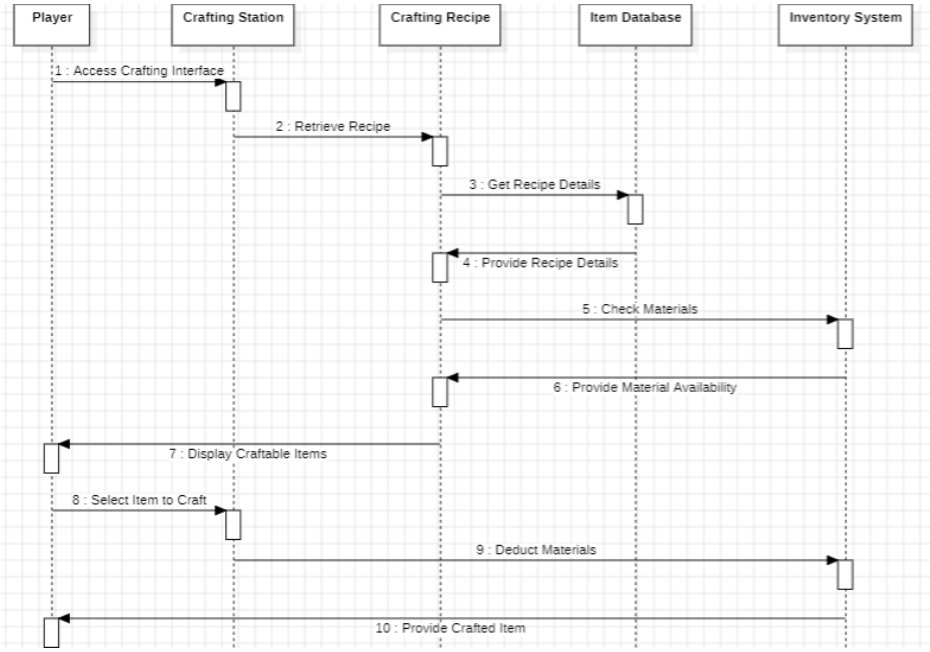


Рисунок 3.6 – Діаграма послідовності «Crafting an Item»

Діаграма кооперації (Collaboration Diagram) використовується для опису поведінки системи на рівні окремих об’єктів, які взаємодіють між собою шляхом обміну повідомленнями з метою реалізації певного варіанта використання або досягнення визначеної цілі. Така діаграма дозволяє представити модель системи як сукупність взаємопов’язаних об’єктів, що спільно беруть участь у виконанні функціональних сценаріїв [6].

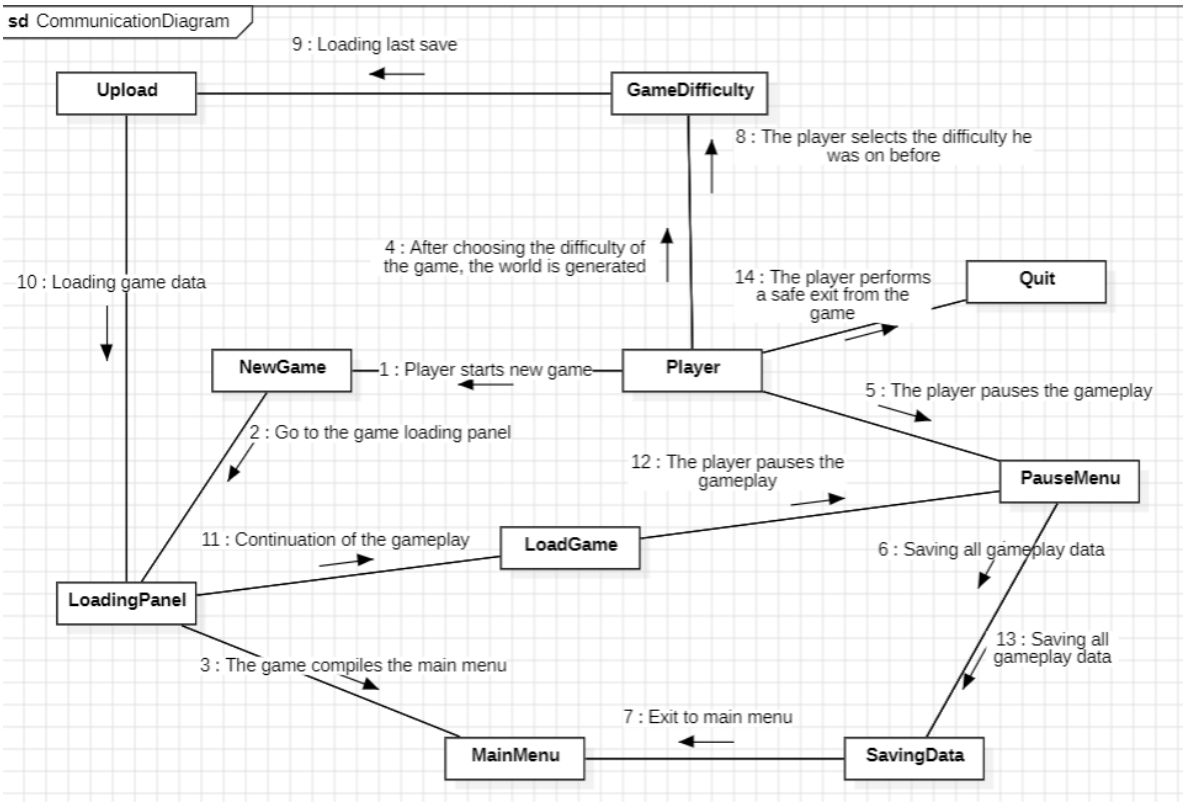


Рисунок 3.7 – Діаграма кооперації

На діаграмі кооперації відображаються об’єкти, які є екземплярами відповідних класів, а також зв’язки між ними, що виступають екземплярами асоціацій. Взаємодія між об’єктами деталізується за допомогою повідомлень, які позначаються стрілками. При цьому на діаграмі показуються лише ті об’єкти, що безпосередньо задіяні в реалізації конкретної кооперації.

3.2.4 Діаграма станів

Діаграма станів (Statechart Diagram) призначена для моделювання переходів об’єкта між різними станами протягом його життєвого циклу. Вона

використовується для аналізу динамічних аспектів системи та є особливо корисною для опису поведінки реактивних об’єктів, тобто таких, що змінюють свій стан у відповідь на зовнішні події [28]. На відміну від інших UML-діаграм, діаграма станів зосереджується на поведінці одного екземпляра класу.

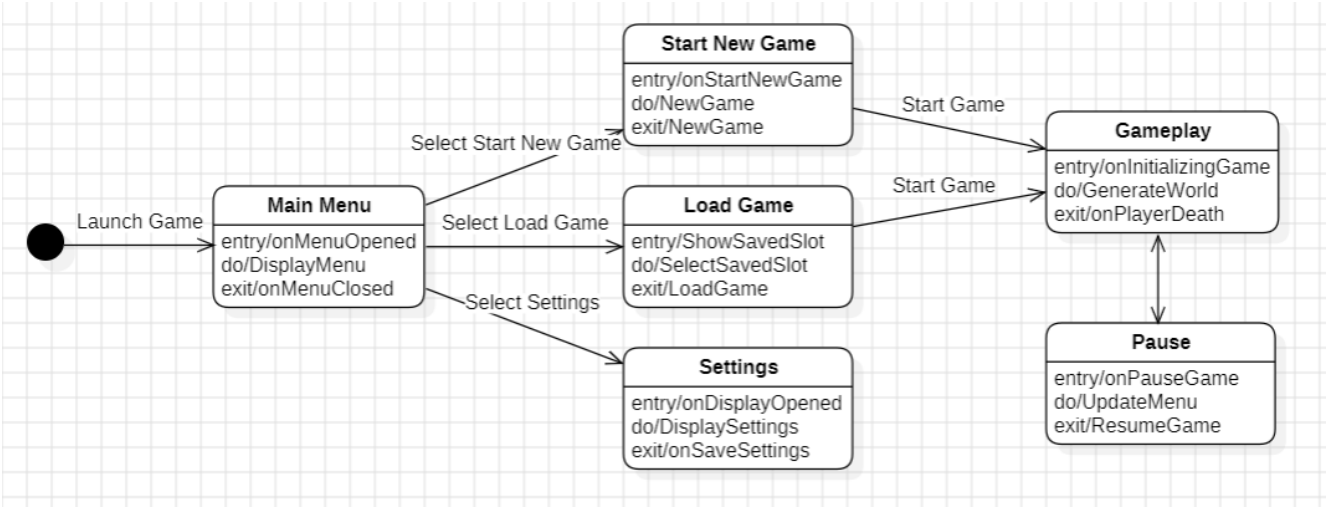


Рисунок 3.8 – Загальна діаграма станів

На загальній діаграмі станів для головного меню гри відображено можливі стани та переходи між ними, зокрема вибір режиму гри (початок нової гри або продовження збереженої), доступ до налаштувань звуку й відео, а також вихід із гри. Можливість завершення гри також передбачена з меню паузи або у разі поразки гравця.

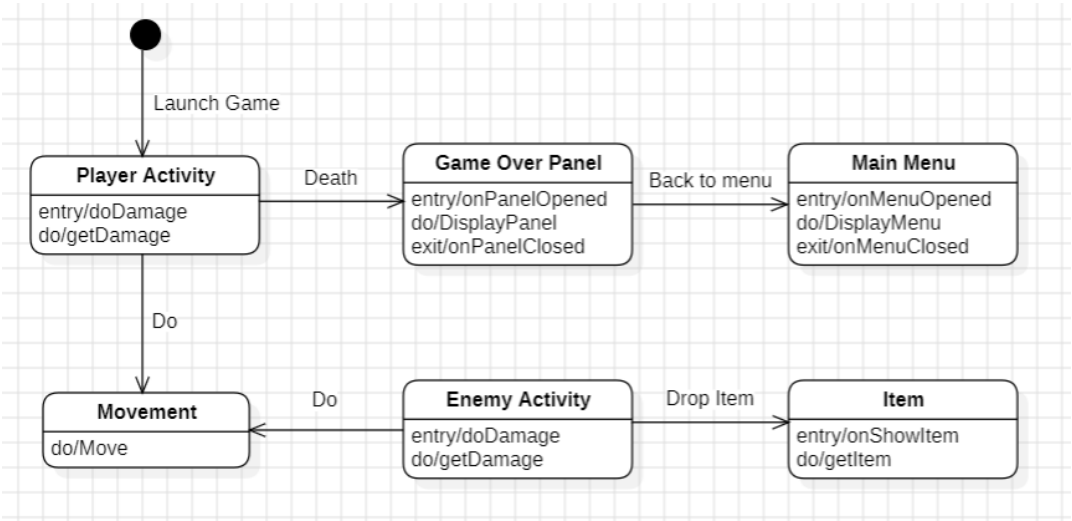


Рисунок 3.9 – Діаграма станів гравця

Діаграма станів гравця ілюструє основні стани персонажа, зокрема пересування, нанесення шкоди та отримання пошкоджень. Аналогічні стани можуть застосовуватися й до ворогів. У разі повного зменшення рівня здоров’я система переходить до стану поразки, що супроводжується відображенням відповідного повідомлення.

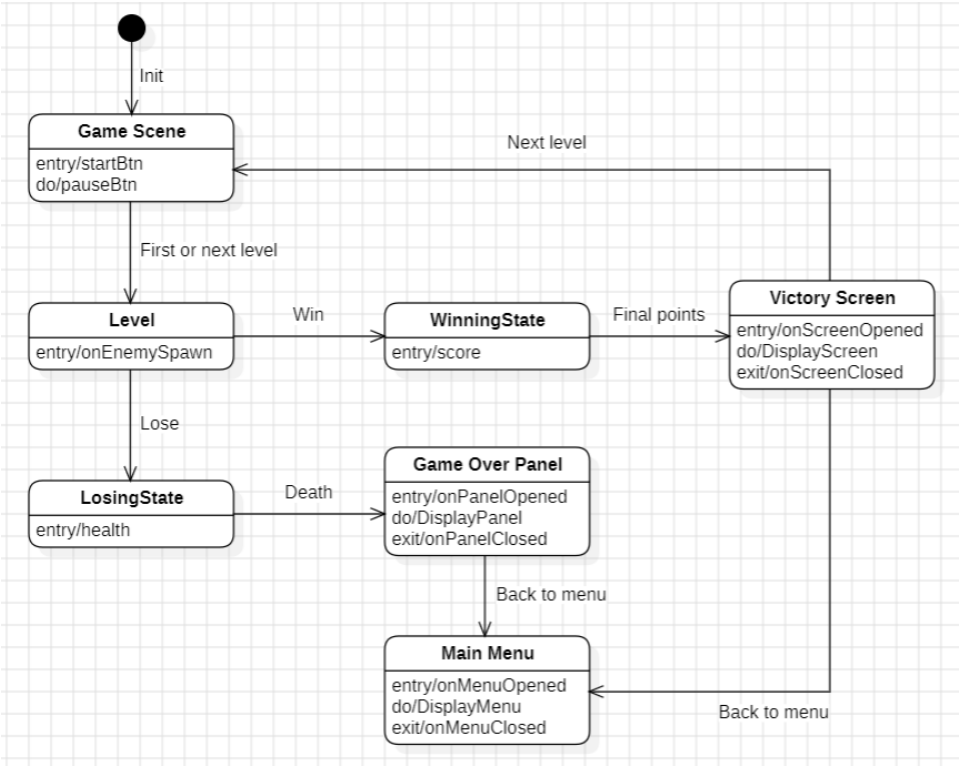


Рисунок 3.10 – Діаграма станів геймплею

Згідно з діаграмою станів геймплею, після запуску гри ініціалізується відповідний ігровий рівень. У разі його успішного проходження гравцеві відображається екран перемоги з підрахунком набраних балів, тоді як у випадку поразки система переходить до стану програшу з можливістю повернення до головного меню.

3.2.5 Діаграма компонентів та розгортання

Діаграма компонентів (Component Diagram) призначена для декомпозиції складної системи на окремі програмні компоненти та відображення взаємозв’язків

між ними. Вона дозволяє проаналізувати архітектуру системи та встановити залежності між її основними складовими [7].

Компонент являє собою логічно завершену частину системи, яка виконує певну функцію або має визначену роль. Компонентами можуть бути програмні модулі, бібліотеки, сервіси або інші автономні елементи, що зазвичай зображаються у вигляді прямокутників з відповідними назвами.

Інтерфейси визначають точки взаємодії між компонентами та відображають функціональні можливості, які компонент надає або очікує від інших частин системи. Надані інтерфейси описують доступні сервіси компонента, тоді як необхідні інтерфейси вказують на зовнішні залежності.

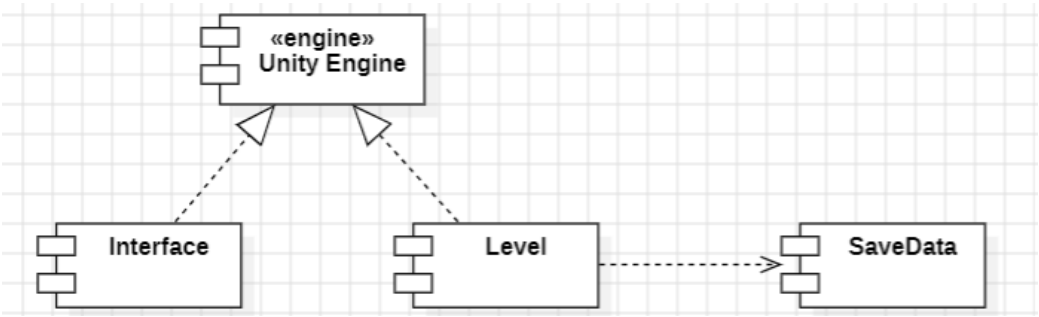


Рисунок 3.11 – Діаграма компонентів

На діаграмі компонентів відображено основні програмні компоненти системи та інтерфейси, за допомогою яких забезпечується їх взаємодія.

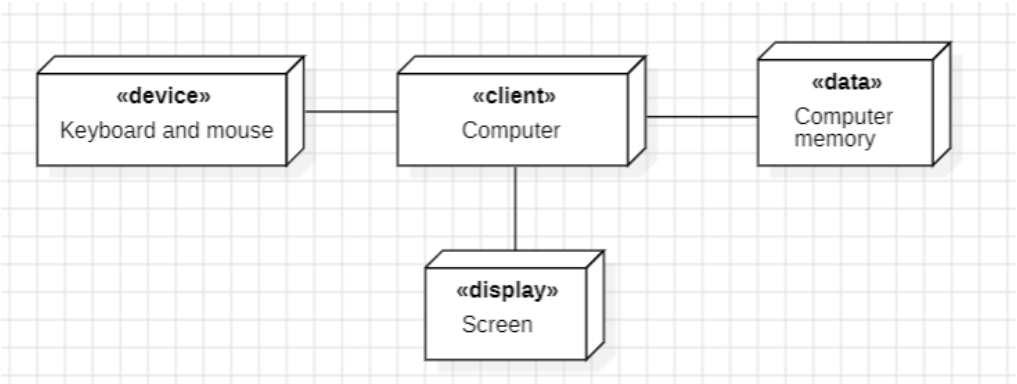


Рисунок 3.12 – Діаграма впровадження

Подання засобів впровадження (deployment view) використовується для відображення розміщення програмних компонентів на вузлах обчислювальної

системи. Воно демонструє конфігурацію апаратних і програмних елементів, а також процеси, що виконуються на відповідних вузлах. Таке подання дозволяє врахувати вимоги до продуктивності, надійності, доступності та масштабованості системи.

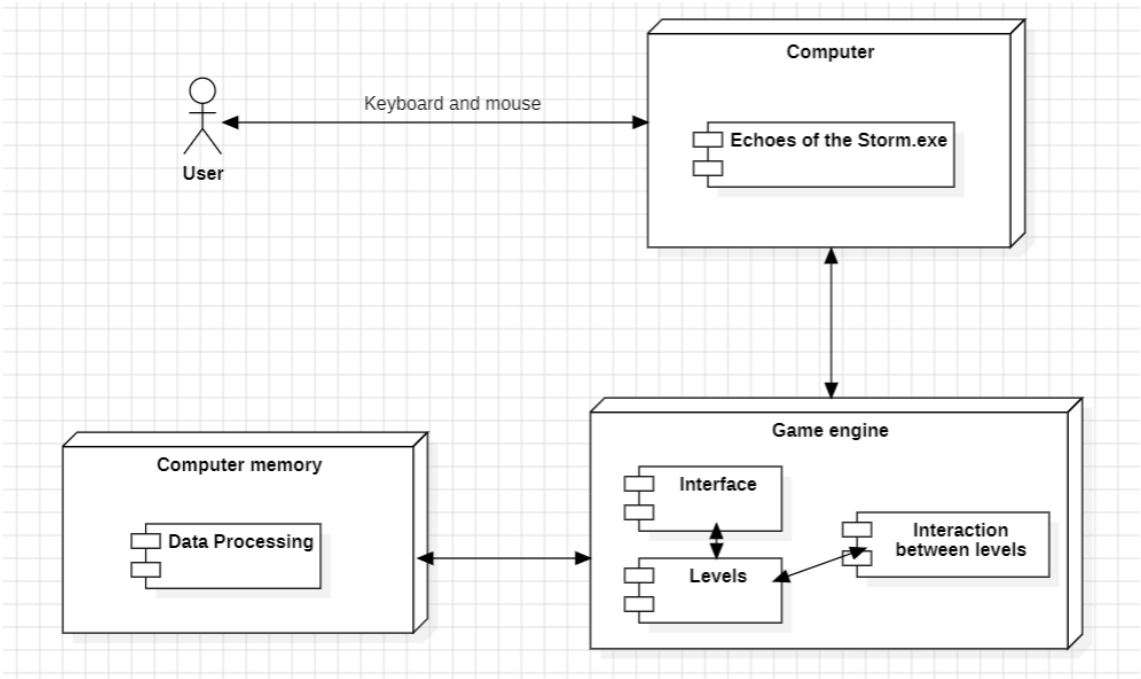


Рисунок 3.13 – Діаграма розгортання

Діаграма розгортання відображає фізичну архітектуру системи та показує, яким чином програмні артефакти розміщуються на апаратних ресурсах. До артефактів належать результати процесу розробки, зокрема виконувані файли, бібліотеки та конфігураційні компоненти. Вузли на діаграмі символізують апаратні або програмні платформи, зв’язки між ними – канали взаємодії, а вкладені елементи – програмні артефакти, що розгортаються на відповідних вузлах.

3.3 Інтерфейс користувача: мокапи та дизайн

Для розроблення мокапів ігрового застосунку (рис. 3.14–3.18) було використано онлайн-інструмент Moqups [20]. Moqups є зручним і функціонально насиченим середовищем для створення каркасів інтерфейсів вебзастосунків, програмного забезпечення та інформаційних панелей. Робочий простір редактора

складається з двох основних панелей: лівої, що містить набір інструментів, і правої, призначеної для форматування та налаштування елементів.

На панелі інструментів розміщені базові компоненти інтерфейсу, засоби керування сторінками проєкту, а також можливості завантаження зображень та іконок. Права панель забезпечує редагування властивостей доданих елементів, зокрема зміну їх розміру, кольору, форми, шрифтів, стилів тексту та візуальних ефектів. До доступних компонентів належать кнопки, форми, текстові поля, посилання, слайдери, таблиці, геометричні фігури та інші елементи, необхідні для проєктування користувацьких інтерфейсів.

Головне меню гри (рис. 3.14) містить чотири основні кнопки: New Game для початку нової гри, Load Game для завантаження збереженого прогресу, Settings для доступу до налаштувань та Exit для виходу із застосунку. Кожна з кнопок відповідає окремій дії та забезпечує навігацію між ключовими функціями гри.

Рисунок 3.14 – Мокап головного меню ігрового застосунку

На рис. 3.15 наведено мокап діалогового вікна підтвердження виходу з гри, яке використовується під час ігрового процесу. Інтерфейс цього вікна надає

гравцеві можливість підтвердити або скасувати завершення гри за допомогою кнопок «Так» та «Ні».

Рисунок 3.15 – Панель виходу з гри

Панель налаштувань гри (рис. 3.16) містить кілька розділів, зокрема конфігурацію, параметри яскравості, налаштування клавіатури, геймпада та графіки. У кожному розділі передбачено відповідні параметри, які користувач може змінювати відповідно до власних уподобань.

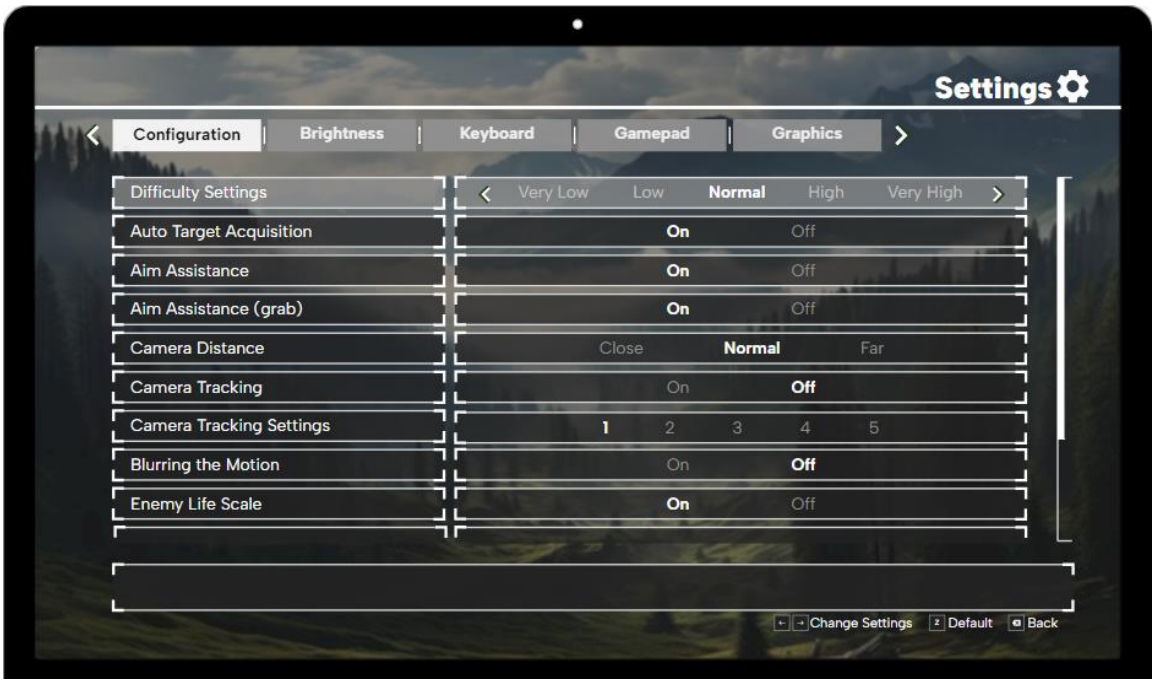


Рисунок 3.16 – Панель налаштувань гри

Ігровий застосунок підтримує збереження прогресу в кількох окремих слотах, що дозволяє гравцям використовувати різні комбінації персонажів, їх навичок і ресурсів. Такий підхід сприяє гнучкому керуванню ігровими сесіями та збереженню індивідуальних налаштувань. Мокап інтерфейсу збереження містить елементи для вибору та керування слотами, зручно розташовані на екрані, що забезпечує зрозумілу та інтуїтивну взаємодію користувача з системою збережень (рис. 3.17).

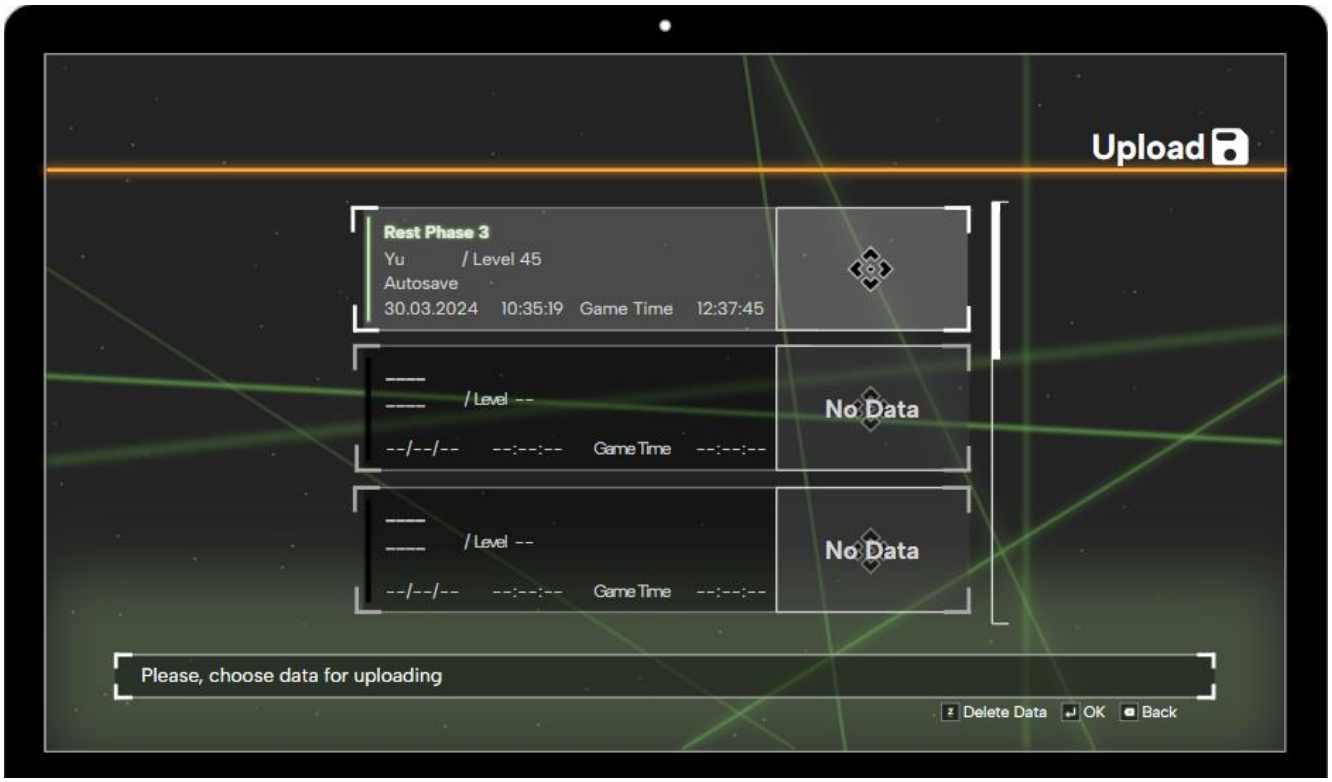


Рисунок 3.17 – Мокап панелі зі збереженими ігровими процесами

Панель спорядження персонажів (рис. 3.18) призначена для вибору та налаштування навичок, озброєння й броні кожного персонажа. Окрім цього, у даному інтерфейсі відображаються ключові ігрові параметри, зокрема максимальний рівень здоров’я, поточний рівень персонажа, показники сили, атаки та захисту, запас мани, кількість золота, а також ігровий час.

Рисунок 3.18 – Мокап панелі спорядження персонажів

Отримані мокапи інтерфейсів ігрового застосунку дозволяють наочно представити структуру користувацького інтерфейсу та логіку взаємодії гравця з основними функціональними модулями системи. Запропоновані рішення забезпечують зрозумілу навігацію, логічне розташування елементів керування та зручний доступ до ключових можливостей гри. Це створює передумови для комфортного користувацького досвіду, зменшує когнітивне навантаження на гравця та сприяє ефективній взаємодії з ігровим середовищем на всіх етапах проходження.

Висновки до розділу 3

Цей розділ було присвячено моделюванню та проєктуванню ігрового застосунку. На першому етапі шляхом побудови сценаріїв використання визначено основні функціональні вимоги системи та взаємодію користувача з її ключовими елементами. Це дозволило сформувати чітке уявлення про очікувану поведінку гри у типових ситуаціях.

На основі сформованих сценаріїв було побудовано UML-діаграми – варіантів використання, класів, послідовності, діяльності, станів, компонентів, розгортання та взаємодії. Кожен тип діаграм надав можливість розглянути систему з різних точок зору: від структурної організації об’єктів до динаміки їх поведінки та способів взаємодії між компонентами. Такий підхід забезпечив формування цілісної та узгодженої архітектурної моделі ігрового застосунку.

Завершальним етапом стало проєктування інтерфейсу користувача, у межах якого було створено серію мокапів, що охоплюють ключові екрани гри – головне меню, панель виходу, налаштування, систему збережень та панель споряджень персонажів. Розробка цих макетів дозволила структурувати навігацію, визначити логіку переходів між інтерфейсними елементами, сформувати узгоджену систему меню та продумати зручність взаємодії гравця з основними функціями гри. Мокапи слугували не лише візуальними шаблонами, а й інструментом перевірки відповідності між логічною моделлю системи та реалізацією користувацького досвіду.

Таким чином, результати цього розділу сформували детальне архітектурне підґрунтя для подальшої реалізації ігрового застосунку та визначили ключові принципи його функціонування, взаємодії елементів і користувацького інтерфейсу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Технічна реалізація інтерфейсу та графіки

Сучасні відеоігри постійно еволюціонують, поєднуючи розгалужені сюжетні лінії, масштабні віртуальні світи та велику кількість ігрових механік. У зв’язку з цим для розробників особливо важливим є правильне визначення пріоритетів у дизайні користувацького інтерфейсу, адже саме він забезпечує зрозумілу та зручну взаємодію гравця з ігровою системою. До складу користувацького інтерфейсу входять меню, кнопки, піктограми, елементи HUD, карти, індикатори та інші інтерактивні компоненти, з якими гравець безпосередньо взаємодіє під час гри. Залежно від способу подання інформації та ступеня інтеграції в ігровий світ, інтерфейси відеоігор класифікують на чотири основні типи.

Дієгетичний інтерфейс інтегрований безпосередньо в ігровий світ і сприймається як його невід’ємна частина. Його елементи є доступними не лише для гравця, а й, умовно, для персонажів гри: вони можуть бути «побачені», «почуті» або «використані» у межах ігрової реальності. Прикладом такого підходу є відображення рівня здоров’я персонажа на елементі костюма в грі Dead Space або інвентар у вигляді предметів, розкладених усередині одягу персонажа в Alone in the Dark.

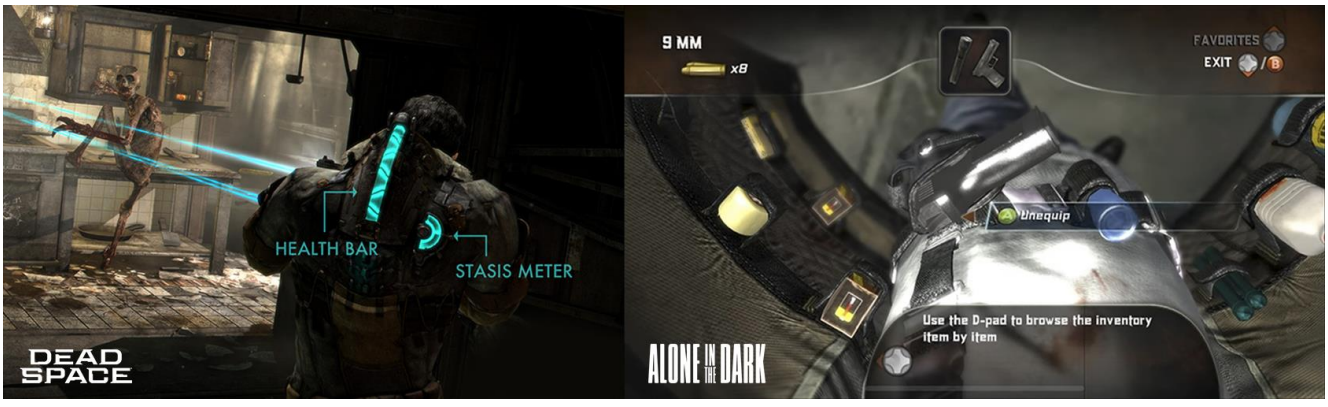


Рисунок 4.1 – Інтерфейс Dead Space та Alone in the Dark

Недієгетичний інтерфейс, навпаки, не має прямого зв’язку з ігровим світом і накладається поверх зображення, існуючи виключно для потреб гравця. До цього

типу належать традиційні панелі HUD, лічильник здоров'я, боєприпасів, очок чи обраної зброї. Подібний інтерфейс широко використовується, зокрема, в іграх Doom та Call of Duty: Black Ops II, де інформаційні елементи постійно відображаються на екрані.

Рисунок 4.2 – Інтерфейс Doom та Call of Duty Black Ops 2

Просторовий інтерфейс поєднує елементи, які розміщуються безпосередньо в ігровому просторі, але при цьому не є частиною логіки ігрового світу. Такі елементи створюються виключно для зручності гравця та допомагають орієнтуватися у середовищі. До них належать маркери завдань, підсвічування об'єктів або вказівники над персонажами. Наприклад, у The Legend of Zelda: Breath of the Wild стрілки над ворогами сигналізують про ціль атаки, а в World of Warcraft: Wrath of the Lich King знаки оклику над NPC вказують на доступні завдання.

Рисунок 4.3 – Інтерфейс Legend of Zelda: Breath of the Wild та World of Warcraft: Wrath of the Lich King

Метаінтерфейс охоплює елементи, які умовно сприймаються як частина ігрового досвіду, але не існують у світі гри у фізичному сенсі. До цього типу

належать динамічна камера, що слідкує за персонажем і змінює ракурс залежно від подій, а також аудіовізуальні ефекти, такі як звук серцебиття чи кров’яні бризки на екрані, що сигналізують про небезпеку або критичний стан героя. Поширеним прикладом метаінтерфейсу є ігрові мобільні телефони, які з’являються на екрані для взаємодії, але передбачають використання їх персонажем. Такий підхід реалізовано в іграх Persona 5, Watch Dogs та Grand Theft Auto V.

Рисунок 4.4 – Інтерфейс Grand Theft Auto 5 та Watchdogs

Загалом якісний дизайн інтерфейсу користувача спрямований на чітке представлення основної ігрової інформації та забезпечення постійного візуального зворотного зв’язку, що полегшує орієнтацію в ігровому процесі й підтримує прийняття рішень гравцем.

Рисунок 4.5 – Інтерфейс ігрового застосунку

У межах розробленого ігрового застосунку з використанням алгоритмів машинного навчання основну увагу зосереджено на застосуванні недієгетичного та просторового типів інтерфейсу, що забезпечує ефективне інформування гравця та зручну взаємодію з адаптивними ігровими механіками.

4.1.1 Сцени

Сцени (Scenes) в ігровому рушії Unity є базовими контейнерами, у межах яких розміщують ігрові об’єкти (Game Objects). Вони використовуються для формування ігрового середовища, створення персонажів, перешкод, декоративних елементів, а також компонентів користувацького інтерфейсу [10].

Під час створення нового проєкту в Unity автоматично доступна стандартна сцена, яка містить основні елементи – камеру (Main Camera) та спрямоване джерело освітлення (Directional Light), що забезпечують базові умови для відображення ігрового простору (рис. 4.6).

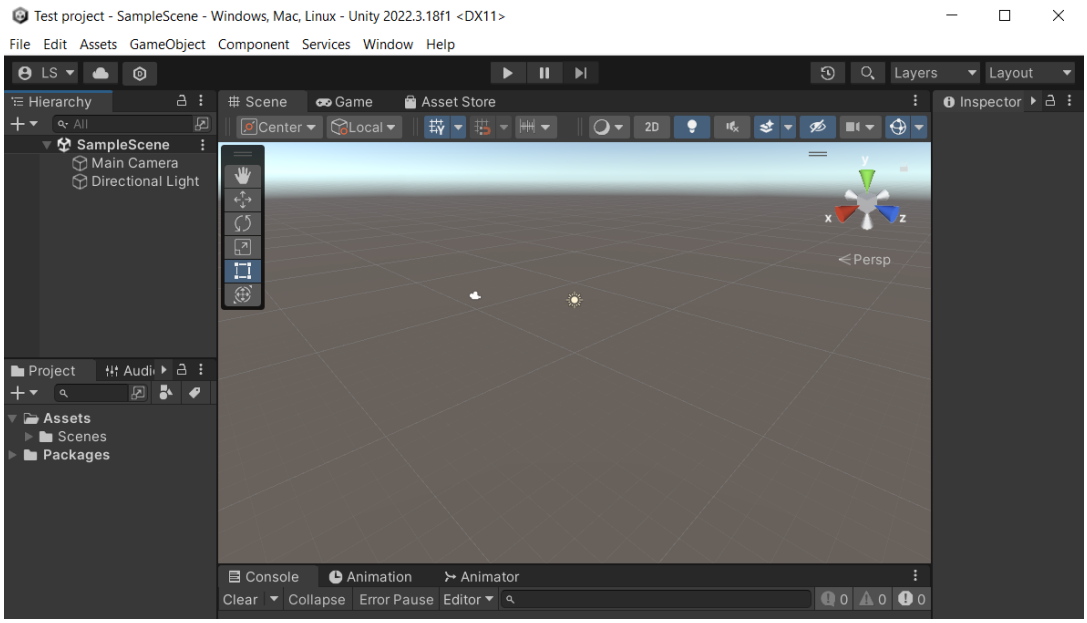


Рисунок 4.6 – Приклад сцени в Unity

Unity надає кілька способів створення нових сцен [4]. Зокрема, сцени можуть бути створені через діалогове вікно створення нової сцени (File → New Scene або за допомогою комбінації клавіш Ctrl / Cmd + N), через меню ресурсів (Assets → Create → Scene), безпосередньо у вікні проєкту шляхом створення сцени

в папці Scenes, а також програмним шляхом – за допомогою скриптів мовою C# із використанням відповідних методів.

Для впорядкування структури проєкту та запобігання плутанини між ігровими об’єктами і скриптами рекомендується надавати сценам та компонентам зрозумілі назви відповідно до їх функціонального призначення.

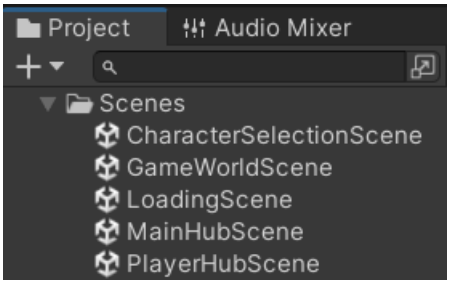


Рисунок 4.7 – Створені сцени в Unity

У межах кожної сцени відбувається взаємодія та керування ігровими об’єктами. До ігрових об’єктів можуть належати як фізичні елементи ігрового світу, що безпосередньо спостерігаються гравцем (персонажі, рослинність, освітлення, зброя, снаряди, візуальні ефекти тощо), так і нефізичні, або логічні, компоненти, зокрема менеджери ресурсів, контролери ігрових режимів чи системи керування станом гри[29].

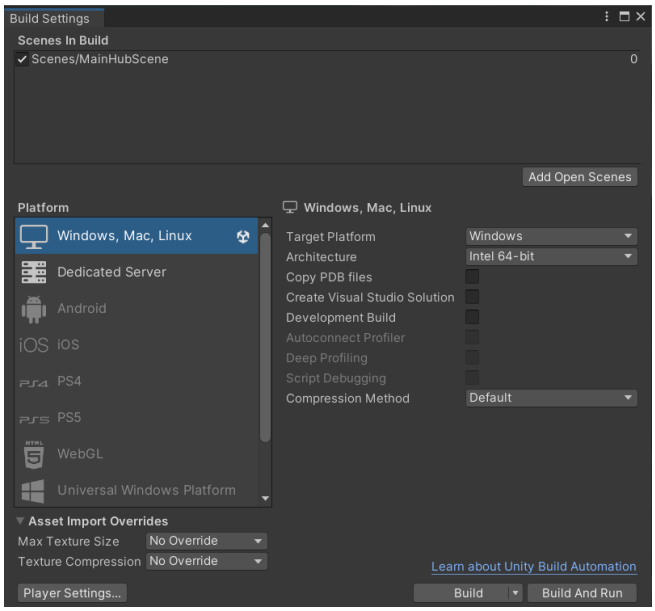


Рисунок 4.8 – Вікно налаштування збірки

Після додавання всіх необхідних елементів до сцени головного меню її необхідно завантажити для перевірки коректності роботи та виявлення можливих помилок. Для цього сцену слід обов’язково додати до налаштувань збірки, інакше під час спроби завантаження буде згенеровано помилку. Додавання здійснюється через меню File → Build Settings, після чого відкривається вікно налаштування збірки (рис. 4.8), у якому обирається цільова платформа та перелік сцен, що підлягають завантаженню.

4.1.2 Іконки та кнопки для меню та ігрових елементів

Для реалізації користувацького інтерфейсу, зокрема ігрових меню, в Unity використовуються спеціальні UI-об’єкти, до яких належать кнопки, зображення, списки, слайдери, перемикачі та інші компоненти [8]. Серед них базовими елементами, що використовуються для створення більшості інтерфейсів, є зображення, кнопки, текстові елементи та панелі, тоді як UI-компоненти, зокрема повзунки й поля введення, зазвичай реалізуються як їх поєднання з додатковими програмними складовими.

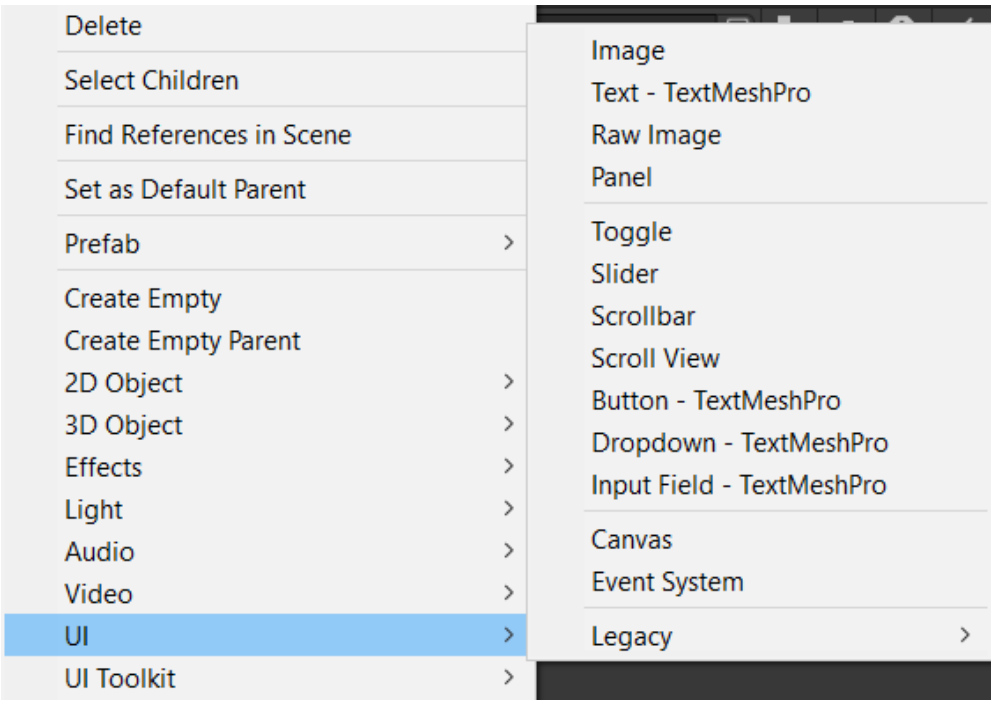


Рисунок 4.9 – UI-об’єкти в Unity

Для роботи з елементами користувацького інтерфейсу в Unity створюється об’єкт Canvas, який слугує основним полотном для розміщення UI-компонентів і відображається у вигляді прямокутної області у вікні сцени, що спрощує процес їх компонування. Хоча використання одного Canvas для всього інтерфейсу є можливим, такий підхід може негативно впливати на продуктивність, оскільки Unity оновлює весь Canvas у разі зміни будь-якого його елемента, наприклад під час анімації кнопок або взаємодії з повзунками. З метою оптимізації доцільно використовувати кілька Canvas для різних груп елементів, розділяючи динамічні та статичні компоненти, а також застосовувати панелі (Panels) для збереження ієрархічної структури.

Панель являє собою двовимірний прямокутний елемент, призначений для групування UI-компонентів, і може містити фоновий колір або зображення, що налаштовуються через компонент Image у вікні інспектора. Зображення є неінтерактивним графічним елементом інтерфейсу, який використовується для візуального представлення інформації або як складова інших UI-компонентів; окрім стандартного елемента Image, Unity підтримує компонент Raw Image, що дозволяє відображати довільні текстури, на відміну від Image, який працює лише з ресурсами типу Sprite.

До панелі головного меню додаються текстові елементи та кнопки. Для створення текстових компонентів використовується TextMeshPro, який забезпечує високу якість відображення тексту та широкі можливості стилізації. Кнопки створюються аналогічним способом із використанням TextMeshPro, що дозволяє поєднати інтерактивність із якісним текстовим оформленням, а налаштування параметрів усіх UI-елементів здійснюється через вікно інспектора після вибору відповідного об’єкта в ієрархії.

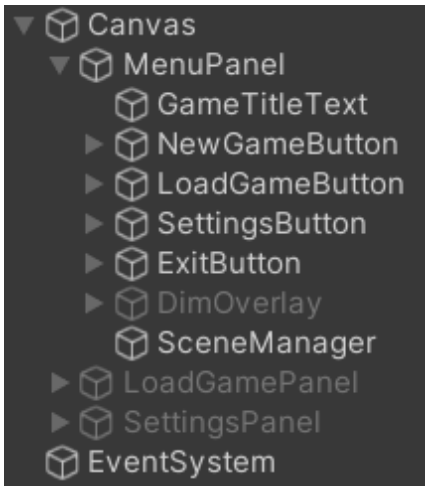


Рисунок 4.10 – Canvas з UI елементами

На Canvas, зображеному на рис. 4.10, розміщено три панелі: MenuPanel, LoadGamePanel та SettingsPanel. Панель MenuPanel виконує функцію головного меню ігрового застосунку, надаючи гравцеві можливість розпочати нову гру, завантажити збережений прогрес або змінити налаштування. Панелі LoadGamePanel і SettingsPanel активуються відповідно після натискання кнопок LoadGameButton та SettingsButton.

4.1.3 3D-моделі об’єктів

3D-модель являє собою цифрове представлення об’єкта у тривимірному просторі. У програмному середовищі Blender геометрія 3D-моделей формується з вершин, ребер і граней, які визначають форму та структуру об’єкта. Такі моделі можуть варіюватися від простих геометричних примітивів, зокрема кубів і сфер, до складних та високодеталізованих об’єктів, персонажів і елементів ігрового середовища [22].

Для оптимізації процесу моделювання в Blender використовуються модифікатори, що дозволяють автоматично змінювати та трансформувати геометрію об’єктів, значно скорочуючи час створення моделей. На початковому етапі було підібрано якісні референсні зображення, які додано до робочої сцени через меню Add → Image → Reference. Для додавання нових об’єктів застосовується комбінація клавіш Shift + A, а базові інструменти трансформації:

переміщення (G), масштабування (S) та обертання (R) використовуються для маніпулювання об’єктами в тривимірному просторі.

У режимі редагування (Edit Mode), який активується клавішею Tab, здійснюється безпосереднє керування вершинами, ребрами та гранями моделі. За допомогою вершин площини можна точно відтворювати контури окремих частин об’єкта відповідно до референсів (рис. 4.11). Нові елементи геометрії додаються з використанням інструмента Extrude (клавіша E). Для забезпечення симетрії моделі застосовується модифікатор Mirror або виконується ручне дублювання геометрії з обох боків, залежно від особливостей об’єкта.

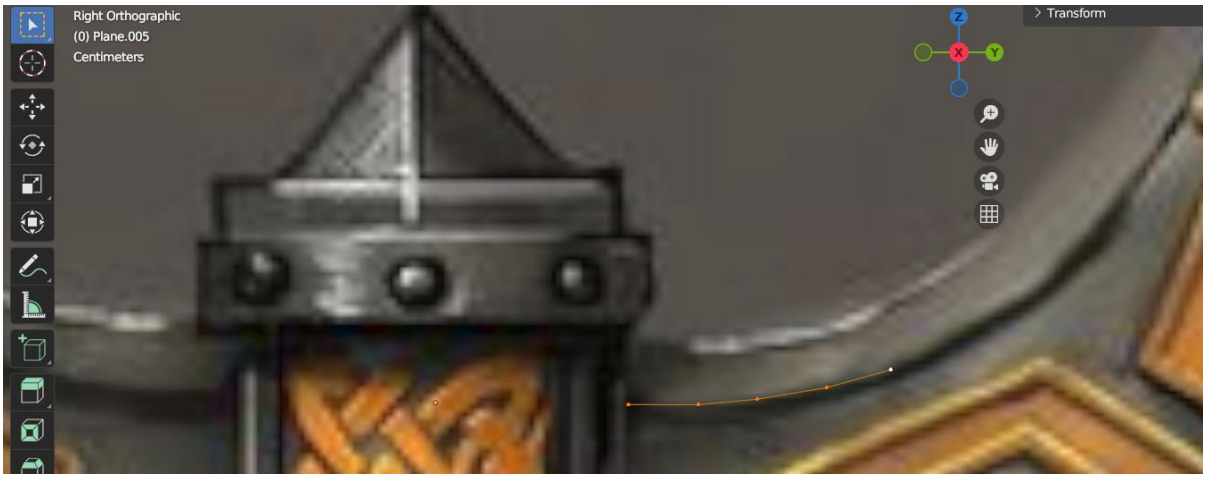


Рисунок 4.11 – Процес розробки 3D-моделі сокири

Додані вершини та ребра поєднуються між собою за допомогою клавіші F. Для уточнення структури сітки використовується інструмент Loop Cut (Ctrl + R), після чого шляхом масштабування (S) формується необхідний об’єм моделі (рис. 4.12). Після завершення моделювання леза сокири додаються інші складові частини, зокрема топорище. Для його створення застосовується циліндр, на поверхню якого накладаються площини з використанням модифікаторів Subdivision, Shrinkwrap, Displace та Solidify, що дозволяє сформувати обмотку та надати об’єкту реалістичного вигляду.

Рисунок 4.12 – Збільшення об’єму створеного об’єкта 3D-моделі

У результаті виконання зазначених етапів отримується повноцінна 3D-модель сокири, яка використовується як елемент озброєння ігрового персонажа та/або противника (рис. 4.13).

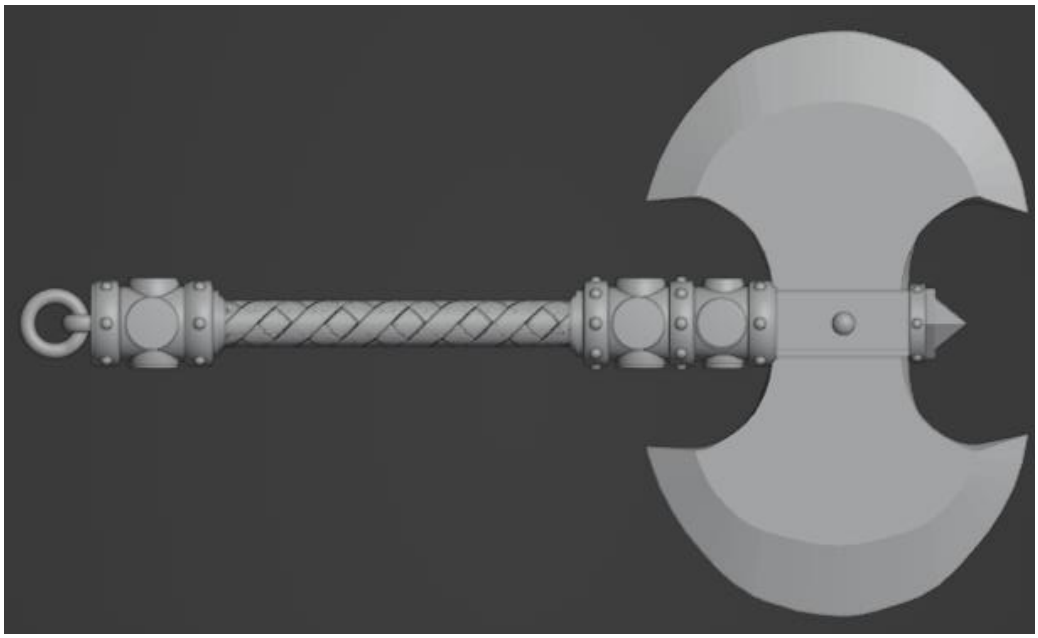


Рисунок 4.13 – 3D-модель сокири

Під час створення 3D-моделей зброї лучника (рис. 4.14) базова форма лука була змодельована з використанням циліндра, який поступово трансформувався за

допомогою інструментів Extrude, Rotate, Move та Scale для досягнення характерної вигнутої форми. Стріли створювалися аналогічним чином, починаючи з циліндра для древка. Вістря стріли формувалося з площини з використанням модифікатора Mirror, що забезпечило симетричність і точність геометрії. Сагайдак також був змодельований на основі циліндра.

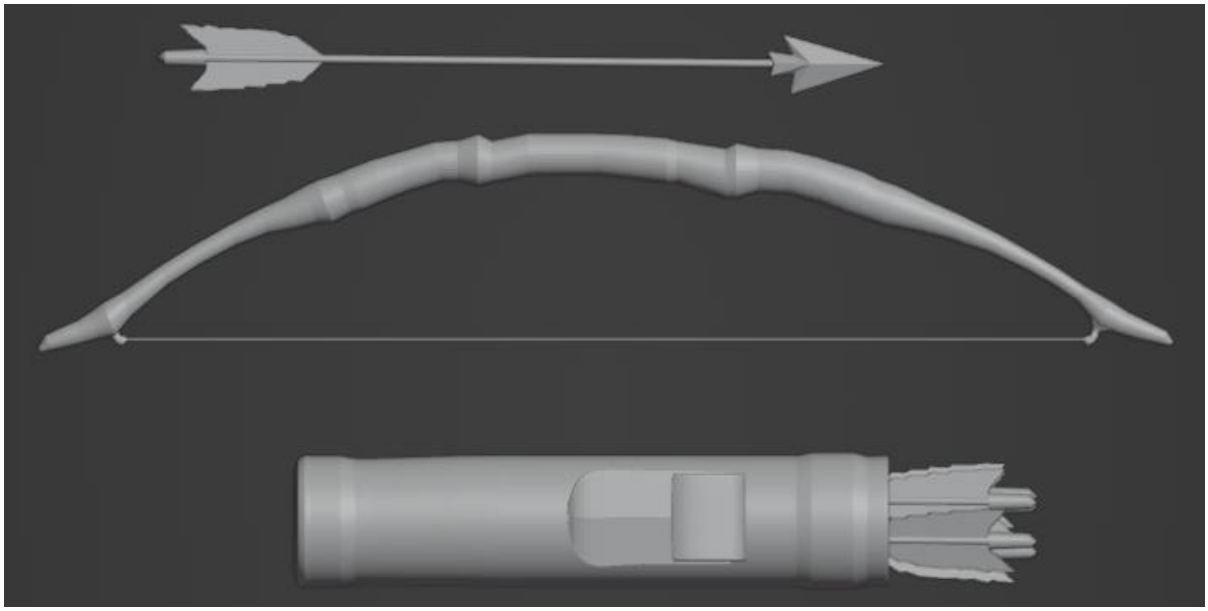


Рисунок 4.14 – 3D-модель зброї лучника

Незважаючи на важливість навколишнього середовища для формування атмосфери та сприйняття ігрового світу, центральну роль у взаємодії з грою відіграють персонажі. Їхній зовнішній вигляд, характерні риси, поведінка, мотивація та моральні якості сприяють емоційному залученню гравця, який часто ототожнює себе з керованим персонажем протягом ігрового процесу.

Першим етапом моделювання персонажа є підбір референсних зображень з різних ракурсів. Після цього у Blender створюється базова форма у вигляді куба, який розміщується відповідно до фронтального референсу. За аналогією з попередніми моделями, використовується режим редагування та інструмент Extrude для формування загальних обрисів тіла. На цьому етапі моделювання рук, стоп і голови не здійснюється, оскільки вони створюються окремо. Після завершення формування основної геометрії у фронтальній проєкції модель коригується відповідно до бічних референсів.

Моделювання здійснювалося за принципом поступового ускладнення: від простих форм до деталізованої геометрії з використанням інструментів Extrude та Sculpt, а також допоміжних засобів, таких як Bevel для згладження країв, Loop Cut and Slide для додавання нових ліній та Knife для створення точних розрізів. Для досягнення плавності поверхонь застосовувалося згладжування шляхом повторного переміщення вершин, що дозволило надати моделі більш природного вигляду (рис. 4.15).

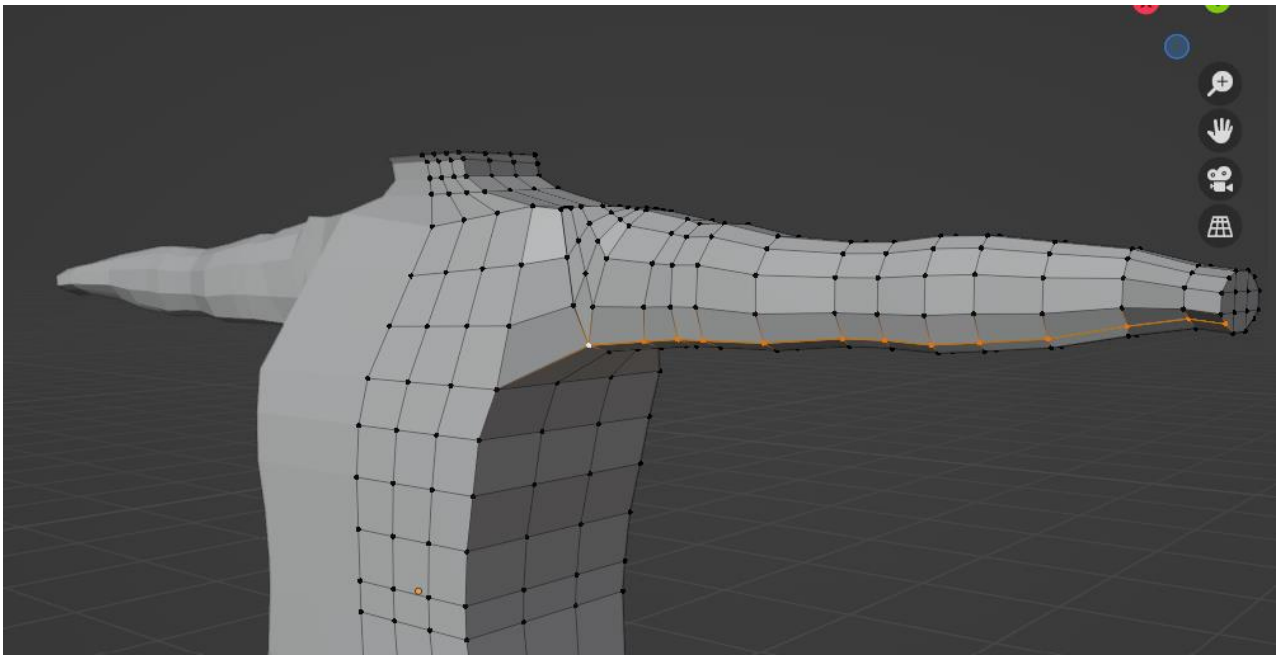


Рисунок 4.15 – Процес моделювання персонажа

Наступним етапом моделювання стало створення кистей рук і стоп персонажа. Кисті формувалися на основі базової геометрії у вигляді куба з подальшим поетапним додаванням пальців із використанням інструментів Extrude та Rotate, що дозволило точно відтворити їх форму та пропорції. Стопи створювалися аналогічним чином, шляхом поступової трансформації простої форми у більш деталізований об’єкт з урахуванням анатомічних особливостей. Такий підхід забезпечив узгодженість геометрії кінцівок з основною моделлю персонажа та підготував її до подальшої анімації.

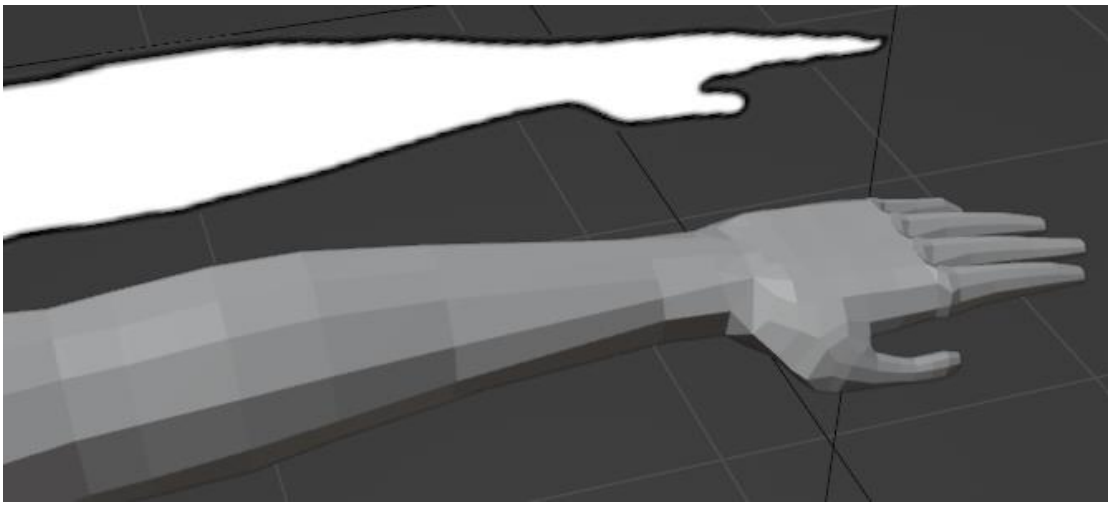


Рисунок 4.16 – Створення кінцівок персонажа

Для наповнення ігрової сцени в Unity, де ландшафт використовується як базовий елемент середовища, додаються персонажі та об’єкти взаємодії. Основу ландшафту можна створити у Blender з використанням площини, яка модифікується за допомогою інструментів Grab, Inflate, Smooth та інших, що дозволяє сформувати рельєф з височинами, долинами та пагорбами [14].

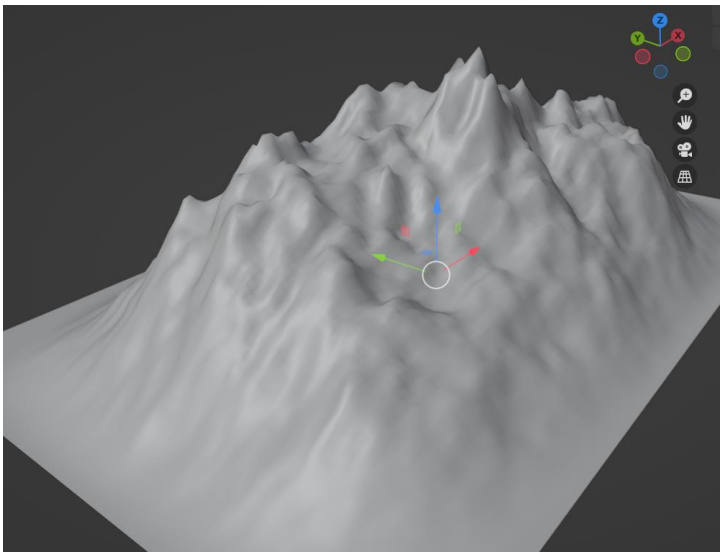


Рисунок 4.17 – Створення височин у Blender

Завершальним етапом є експорт створених 3D-моделей з Blender та їх імпорт до Unity. Для цього використовується формат FBX шляхом вибору File → Export → FBX (.fbx) після чого експортований файл додається до папки Assets у середовищі Unity [13].

4.2 Реалізація алгоритмів машинного навчання

У розробленому ігровому застосунку алгоритми машинного навчання реалізовано як окремий логічний модуль, що відповідає за адаптивну поведінку ігрових об’єктів та автоматичне налаштування параметрів ігрового процесу. Функціональність ML-підсистеми зосереджена в класі MLManager, який поєднує всі необхідні дані моделей та забезпечує їх використання іншими компонентами гри.

MLManager містить структуру даних, що відповідають за класифікацію складності (масив diffucultyCentroids). Вхідні дані формує клас PlayerMetrics, який накопичує статистику поведінки гравця (точність атак, час реакції, отриману шкоду тощо). Кожен новий набір показників передається до MLManager для визначення рекомендованого рівня складності, який застосовується до ігрових параметрів (здоров’я ворогів, їхня кількість, рівень загрози).

Для керування логікою супротивників використовується клас EnemyAgent, який отримує з MLManager доступ до Q-таблиці (qTable) та викликає методи для отримання оптимальної дії залежно від стану NPC. MLManager інтерпритує стан супротивника через спеціалізовані структури та повертає дію, яку EnemyAgent виконує через власні методи. Таким чином, ML-модуль забезпечує варіативність поведінки NPC без застосування жорстко визначених сценаріїв.

Клас CompanionAgent також взаємодіє з MLManager та отримує від нього визначений стиль поведінки гравця відповідно до масиву styleCentroids. Кожен стиль відповідає певному набору тактичних реакцій союзника, які застосовується через методи оновлення поведінки. Тобто, союзник динамічно підлаштовується під стиль гри користувача, забезпечуючи підтримку, що відповідає його поточним діям та стратегії.

Окремий тип поведінки реалізовано для босів, які відрізняються складнішими шаблонами взаємодії та багатофазовою структурою бою. Для них використано політику (тобто навчену модель вибору дій), отриману за допомогою алгоритму Proximal Policy Optimization (PPO), що входить до складу ML-Agents.

Навчена політика була дискретизована та інтегрована до MLManager у вигляді масиву bossPolicy, який використовується класом BossAgent для вибору оптимальної дії відповідно до поточного стану бою. Завдяки цьому поведінка босів є більш динамічною, стратегічною та непередбачуваною.

4.2.1 Навчання моделей

Оскільки ігровий застосунок використовує алгоритми машинного навчання у вигляді попередньо обчислених структур даних, навчання всіх моделей здійснювалося поза середовищем Unity, у окремому програмному середовищі, орієнтованому на обробку даних та аналітичні обчислення. Для організації симуляційного середовища та взаємодії між Python-скриптами і Unity-оточенням використовувався інструментарій Unity ML-Agents, що забезпечував можливість багаторазового моделювання бойових ситуацій та збору навчальних даних [11].

Для моделі адаптивного налаштування складності використовувався набір статистичних показників гравця, які відповідають структурі класу PlayerMetrics: точність атак, час реакції, отримана шкода та інші параметри, що характеризують стиль і ефективність дій користувача. На основі цих даних формувався масив векторів ознак, кожен з яких відображав результат проходження рівня або бою.

Далі проводилося маркування даних відповідно до трьох рівнів складності, після чого застосовувався алгоритм класифікації k-ближчих сусідів. У ході навчання визначалися центроїди кожного класу, які пізніше використовуються в Unity для швидкої класифікації. Фрагмент Python-коду навчання класифікатора:

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import numpy as np, json

X = df[["accuracy", "reaction_time", "damage_taken", "damage_dealt"]].values
y = df["difficulty_label"].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_scaled, y)

centroids = []
for label in np.unique(y):
    centroids.append(X_scaled[y == label].mean(axis=0))
```

```
with open("difficulty_model.json", "w") as f:
    json.dump(
        {
            "centroids": np.array(centroids).tolist(),
            "scaler_mean": scaler.mean_.tolist(),
            "scaler_scale": scaler.scale_.tolist()
        }, f)
```

Отримані центроїди експортувалися у формат, сумісний із внутрішнім поданням класу MLManager (масив difficultyCentroids).

Для моделювання поведінки супротивників використовувався алгоритм навчання з підкріпленням Q-learning, у результаті роботи якого формується таблиця Q-значень. Навчання проводилося у спеціальному симуляційному середовищі, де агент багаторазово виконував різні дії, оцінював їх наслідки та коригував свою поведінку. Фрагмент Python-коду навчання Q-таблиці:

```
import numpy as np, random

states = 4
actions = 4
Q = np.zeros((states, actions))

alpha = 0.1
gamma = 0.9


def choose_action(s):
    return np.argmax(Q[s]) if random.random() > 0.1 else random.randint(
        0, actions - 1)


def reward(s, a):
    if s == 0 and a == 0: return 10
    if s == 2 and a == 1: return 8
    return -1


for _ in range(5000):
    s = random.randint(0, states - 1)
    a = choose_action(s)
    r = reward(s, a)
    ns = random.randint(0, states - 1)

    Q[s, a] += alpha * (r + gamma * np.max(Q[ns]) - Q[s, a])

np.save("q_table.npy", Q)
```

Згенерована таблиця Q-значень експортувалася у числовому форматі та пізніше використовувалася Unity-компонентом MLManager для вибору поведінкових дій NPC.

Для адаптації поведінки союзника використовувався алгоритм K-means, який дозволив розподілити стиль гри користувачів на кілька груп: агресивний,

тактичний, оборонний. У навчання брали участь параметри бойової активності, частота використаних умінь, середня дистанція до супротивників, кількість отриманих ушкоджень тощо. Фрагмент Python-коду кластеризації:

```
from sklearn.cluster import KMeans

X = df[["accuracy", "reaction_time", "damage_taken", "damage_dealt"]].values
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_scaled)

centroids = kmeans.cluster_centers_

with open("style_model.json", "w") as f:
    json.dump({"style_centroids": centroids.tolist()}, f)
```

Отримані центроїди зберігалися у форматі багатовимірних масивів та використовувалися під час гри в методі PredictStyle() класу MLManager.

Окремим етапом було навчання моделі для керування поведінкою босів, які відрізняються складнішими шаблонами бою та багаторазовою структурою взаємодії з гравцем. Для цієї задачі використовувався алгоритм Proximal Policy Optimization (PPO), що входить ML-Agents і реалізує архітектуру «актор-критик» [21].

У межах цього підходу агент-бос навчався у симуляційному середовищі ML-Agents, отримуючи винагороди за ефективні дії (точні удари, успішні ухилення, перехід у наступну фазу бою) та штрафи за неефективні рішення. У процесі багатоциклових тренувань оптимізувалася політика вибору дій $\pi(a|s)$, яка надалі задає поведінку боса. Фрагмент коду для отримання дії з політики PPO:

```
obs = np.array([distance, boss_hp, player_hp, phase], dtype=np.float32)
action = model.predict(obs)
```

Після завершення тренування модель PPO експортувалася у формат ONNX та проходила процедуру дискредитації: для кожної комбінації станів визначалась дія з максимальною ймовірністю. Фрагмент коду дискретизації політики:

```
outputs = session.run(None, {input_name: obs})
best_action = int(np.argmax(outputs[0][0]))
```

Таким чином формувався масив bossPolicy, який надалі завантажується до MLManager та використовується класом BossAgent під час обчислення поведінкових реакцій боса.

Усі моделі навчалися у зовнішньому середовищі та експортувалися у стандартизованому форматі вкладених масивів (float[][]) і числових структур (float[]), що зберігали інформацію про центроїди класифікації, кластери стилів, Q-значення або політику PPO. Ці дані використовувалися як вихідні для подальшої інтеграції у Unity.

4.2.2 Інтеграція з ігровим рушієм

Інтеграція моделей машинного навчання в ігровий рушій Unity здійснювалася через окремий керуючий компонент MLManager, який відповідає за завантаження, декодування та використання попередньо обчислених структур даних. Усі результати навчання експортувалися у форматі вкладених масивів (float[][]) та зберігалися у JSON-файлах, після чого завантажувалися в Unity під час старту гри.

Під час ініціалізації MLManager здійснював десеріалізацію отриманих масивів і перетворення їх у двовимірні структури (float[,]), оптимізовані для швидкого доступу під час виконання гри. Такий підхід дає змогу уникнути додаткових витрат пам’яті та обчислень у реальному часі та забезпечує стабільну продуктивність на всіх етапах геймплею. У внутрішній структурі MLManager формувалися такі масиви:

– difficultyCentroidsRaw difficultyCentroids – центроїди класифікації складності, що використовуються під час визначення рівня гри на основі PlayerMetrics;

– styleCentroidRaw styleCentroids – центроїди кластерних стилів гравця, які CompanionAgent застосовує для вибору тактичної поведінки;

– qTableRaw qTable – таблиця Q-значень для EnemyAgent, що визначає оптимальну дію супротивника у конкретному стані;

– bossPolicyRaw bossPolicy – дискретизована політика PPO для боса, яка містить імовірнісні оцінки дій у кожному стані та використовується BoddAgent.

Нижче наведено фрагмент JSON-файлу, що використовується MLManager для ініціалізації внутрішніх масивів під час старту гри.

```
"difficultyCentroids":[
   [
      0.15,
      -0.32,
      0.48,
      -0.10,
      0.22
   ],
   [
      0.45,
      0.10,
      -0.12,
      0.35,
      0.05
   ],
   [
      0.78,
      0.42,
      -0.30,
      0.60,
      -0.15
   ]
],
```

Наведений формат збереження забезпечує безпосереднє зіставлення числових даних із внутрішнім поданням моделей у класі MLManager. Після завершення завантаження MLManager надає відповідні методи доступу для ігрових компонентів.

Клас EnemyAgent звертається до qTable для отримання оптимальної поведінкової реакції, виконуючи вибір дії методом argmax за значенням Q. Клас CompanionAgent використовує обчислений стиль гравця та застосування відповідного набору поведінкових шаблонів, узгоджених з поточним кластером K-means. Клас BossAgent взаємодіє із MLManager через метод GetBossAction(), який визначає дію боса на основі політики PPO. Під час оновлення стану боса BossAgent формує відповідний індекс стану, передає його MLManager та отримує дію з найбільшою імовірністю виконання. Далі ця дія реалізується за допомогою внутрішніх методів BossAgent, що забезпечує складнішу, багатофазну та адаптивну поведінку боса порівняно зі звичайними супротивниками.

Взаємодія Unity-скриптів і ML-підсистеми базується на компонентній архітектурі рушія: кожен агент – гравець, супротивник, союзник чи бос, що містить посилання на MLManager і викликає його методи у процесі оновлення. Такий

підхід забезпечує централізоване управління навчальними моделями, уніфікований доступ до даних і чітке розділення ролей між логікою гри та обчислювальними моделями машинного навчання.

Завдяки цьому процес інтеграції передбачає лише роботу з уже готовими даними без виконання навчання в реальному часі, що дозволило досягти високої продуктивності й адаптивності гри відповідно до стилю користувача та структури ігрових ситуацій.

4.3 Написання скриптів та логіки гри

Реалізацію ігрової логіки здійснено за допомогою C#-скриптів, що функціонують як компоненти Unity та взаємодіють із середовищем через цикл Update(), систему фізики та подій. Кожен клас виконує окрему функцію, що забезпечує модульність, повторне використання коду та узгодженість з підсистемою машинного навчання.

Базовий клас PlayableCharacter реалізує спільні характеристики ігрових сутностей, зокрема кількість здоров’я, швидкість руху, силу атаки та методи взаємодії, які успадковуються іншими класами, включно з Player, EnemyAgent, CompanionAgent та BossAgent. Реалізовано методи обробки ушкоджень та смертіі персонажа. Фрагмент коду, що демонструє базову реаліізацію методу зменшення здоров’я:

```
public virtual void TakeDamage(float amount)
{
    currentHealth -= amount;
    if (currentHealth <= 0) OnDeath();
}
```

Клас PlayerMovement відповідає за переміщення персонажа та взаємодію з фізичним середовищем Unity через компонент CharacterController. Обробляється введення користувача, розрахунок напряму руху, гравітаційна взаємодія та стрибок. Фрагмент коду, що ілюструє базову обробку введення користувача:

```
if (controller.isGrounded)
{
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");
    moveDirection = transform.forward * v * speed;
    if (Input.GetButton("Jump")) moveDirection.y = jumpForce;
}
```

Клас CameraController забезпечує прив’язку камери до позиції гравця та реалізує плавне слідування за ним. Основна ідея полягає у додаванні фіксованого зміщення та постійному орієнтуванні камери на ціль. Фрагмент коду основного оновлення камери:

```
transform.position = target.position + offset;
transform.LookAt(target);
```

Трекер метрик відповідає за накопичення статистичних даних, необхідних для роботи ML-модулів. Зокрема, фіксуються точність атак, середній час реакції, завдана та отримана шкода. Зібрані дані надалі передаються у MLManager для класифікації складності та визначення стилю гравця. Фрагмент коду оновлення статистики точності:

```
public void RegisterShot(bool hit)
{
    shots++;
    if (hit) hits++;
    metrics.accuracy = (float) hits / shots;
}
```

Клас MLManager виконує завантаження та подальшу інтерпретацію попередньо обчислених моделей машинного навчання, отриманих під час офлайн-навчання. У структурі зберігаються параметри нормалізації ознак, центроїди класифікації складності, центроїди кластеризації стилів, таблиця Q-значень, а також дискретизована політика PPO для керування поведінкою боса. Фрагмент коду вибору дії супротивника на основі Q-таблиці:

```
public int GetEnemyAction(int state)
{
    int best = 0;
    float bestValue = float.MinValue;
    for (int i = 0; i < qTable.GetLength(1); i++)
    {
        if (qTable[state, i] > bestValue)
        {
            bestValue = qTable[state, i];
            best = i;
        }
    }
    return best;
}
```

Клас EnemyAgent використовує дані ML-модуля для формування варіативної поведінки супротивників. Після оцінки поточного стану NPC агент звертається до MLManager за дією, яку слід виконати. Фрагмент взаємодії зі схемою вибору дій:

```
int action = ml.GetEnemyAction(GetState());
Execute(action);
```

При роботі класу CompanionAgent реалізовано підлаштування поведінки союзника під стиль гри користувача на основі кластеризації. Після отримання індексу стилю від MLManager обирається відповідний набір тактичних реакцій. Фрагмент перемикання поведінки:

```
int style = ml.PredictStyle(features);
if (style != currentStyle)
{
    currentStyle = style;
    ApplyStyle(style);
}
```

Для реалізації складнішої поведінки босів застосовано окремий агент BossAgent, який використовує дискретизовану політику PPO, отриману внаслідок офлайн-навчання в ML-Agents. Перед виконанням дії агент визначає свій поточний стан, після чого запитує MLManager щодо відповідної поведінкової реакції. Фрагмент коду вибору дії:

```
void Update()
{
    int state = EvaluateState();
    int action = ml.GetBossAction(state);
    ExecuteBossAction(action);
}
```

Фрагмент методу MLManager, що повертає дію згідно з політикою PPO:

```
public int GetBossAction(int state)
{
    int best = 0;
    float max = float.NegativeInfinity;
    for (int i = 0; i < bossPolicy.GetLength(1); i++)
    {
        if (bossPolicy[state, i] > max)
        {
            max = bossPolicy[state, i];
            best = i;
        }
    }
    return best;
}
```

Завдяки цьому боса характеризує складна, фазова, нелінійна поведінка, що не може бути реалізована традиційними сценаріями.

Офлайн-моделі зберігаються у JSON-форматі та завантажуються в Unity засобами JsonUtility. Це забезпечує сумісність між Python-скриптами та C#-частиною проєкту. Фрагмент коду завантаження моделі:

```
TextAsset asset = Resources.Load < TextAsset > ("ml_model");
model = JsonUtility.FromJson < MLModelData > (asset.text);
```

Підсистеми інвентаря, квестів та збереження програми реалізовано окремими класами відповідно до UML-діаграми класів. Дані гри серіалізуються у JSON. Фрагмент коду серіалізації:

```
string json = JsonUtility.ToJson(data);
File.WriteAllText(savePath, json);
```

Висновки до розділу 4

У межах четвертого розділу здійснено повну програмну реалізацію ігрового застосунку, що охоплює технічне проєктування інтерфейсу, побудову графічних елементів, створення тривимірних моделей, інтеграцію алгоритмів машинного навчання та розроблення скриптів для керування логікою ігрових об’єктів. Інтерфейс користувача створено з дотриманням принципів послідовності, мінімалістичного дизайну, візуальної ієрархії та контекстного зворотного зв’язку, що забезпечило зручність навігації й інтуїтивність взаємодії. Графічні ресурси (сцени, моделі персонажів, об’єкти довкілля, зброя та інші елементи) спроєктовано у Blender, оптимізовано під вимоги рушія та імпортовано до Unity з належним налаштуванням матеріалів і текстур.

Особлива увага була приділена реалізації підсистеми машинного навчання. У роботі застосовано класифікаційні моделі, алгоритми навчання з підкріпленням (Q-learning), методи кластеризації (K-means), а також політику PPO для формування складної поведінки боса. Усі моделі навчалися офлайн у Python-середовищі, після чого результати навчання були приведені до єдиного числового формату та інтегровані до ML-модуля Unity без необхідності виконання обчислень у реальному часі.

Розроблені C#-скрипти забезпечили функціонування ключових механік: руху персонажа, керування камерою, обробки взаємодій, логіки противників та союзників, системи метрик гравця, інвентаря, квестів і збереження прогресу. Реалізована архітектура передбачає модульність, повторне використання компонентів та чітке розмежування відповідальності між класами, що підвищує масштабованість та підтримуваність системи.

ВИСНОВКИ

У ході написання кваліфікаційної магістерської роботи розроблено ігровий застосунок з адаптивним ігровим процесом, реалізованим із використанням алгоритмів машинного навчання. Проведено аналіз обраної предметної області та сучасних підходів до розроблення ігрових застосунків, зокрема рішень, що використовують алгоритми машинного навчання для реалізації адаптивної поведінки. На основі виконаного аналізу сформовано задачі дослідження та специфікацію вимог до програмного забезпечення, що визначили функціональні та нефункціональні характеристики системи.

Розроблено концепцію, архітектуру та ігрову механіку застосунку з урахуванням необхідності динамічної адаптації ігрового процесу. Проєктування системи включало моделювання об’єкта та предмета дослідження, побудову сценаріїв використання, UML-діаграм, а також створення функціональних моделей програмного забезпечення. Окрему увагу приділено проєктуванню інтерфейсу користувача та графічного дизайну, що відповідають змінним параметрам ігрового середовища.

У межах роботи використано ігровий рушій Unity, який забезпечує інтеграцію графіки, фізики, анімації та засобів штучного інтелекту, а також мову програмування C# для реалізації ігрової логіки. Для навчання агентів машинного навчання та аналізу даних застосовано мову програмування Python. Створення 3D-моделей і візуальних елементів виконано з використанням Blender та Adobe Photoshop.

Поставленої мети було досягнуто шляхом інтеграції алгоритмів машинного навчання в логіку ігрового застосунку, що дозволило реалізувати механізми динамічної адаптації ігрового процесу. Адаптивність забезпечувалася аналізом дій та поведінки гравця, на основі яких система коригувала параметри складності, поведінку неігрових персонажів і окремі елементи ігрового середовища. Використання моделей машинного навчання дало змогу перейти від статичних сценаріїв до персоналізованого досвіду, у якому складність та реакції гри

змінюються відповідно до індивідуальних характеристик користувача, що підтверджує ефективність обраного підходу.

Виконано такі **завдання**:

– досліджено принципи роботи алгоритмів машинного навчання та можливості їх використання в ігрових застосунках;

– проаналізовано сучасні ігри, що використовують машинне навчання, та визначено їхні ключові особливості;

– розроблено концепцію, архітектуру та механіку ігрового застосунку;

– створено інтерфейс та графічний дизайн;

– реалізовано ігровий застосунок і забезпечено його адаптивність за рахунок використання алгоритмів машинного навчання;

– перевірено якість роботи застосунку.

При тестуванні роботи ігрового застосунку помилок не виявлено. Розроблений застосунок може бути використаний як основа для створення подібних ігрових проєктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сеніва К. Р. Способи використання нейронних мереж та машинного навчання в комп'ютерних іграх. *Вісник Хмельницького національного університету*. 2021. № 2 (295). с. 97-100. DOI: 10.31891/2307-5732-2021-295-2-97-100.

2. Ameisen E. Building Machine Learning Powered Applications. Sebastopol : O'Reilly Media, 2020. 257 p

3. Bond J. G. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#. 3rd ed. Boston : Addison-Wesley, 2022. 1296 p.

4. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game. 1st ed. Sebastopol : O'Reilly Media, 2019. 405 p.

5. Class Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/class-diagram (access date: 06.10.2025).

6. Communication Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/communication-diagram (access date: 08.10.2025).

7. Component Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/component-diagram (access date: 10.10.2025).

8. Davis A., Baptiste T., Craig R. Unity 3D Game Development: Designed for passionate game developers Engineered to build professional games. Birmingham : Packt Publishing, 2022. 370 p.

9. Engelbrecht D. Introduction to Unity ML-Agents: Understand the Interplay of Neural Networks and Simulation Space Using the Unity ML-Agents Package. 1st ed. Berkeley : Apress, 2023. 224 p.

10. Ferrone H. Learning C# by Developing Games with Unity: Get to grips with coding in C# and build simple 3D games in Unity 2023 from the ground up. 7th ed. Birmingham : Packt Publishing, 2022. 458 p.

11. Gallatin K., Albon C. Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning. 2nd ed. Sebastopol : O'Reilly Media, 2023. 413 p.

12. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 3rd ed. Sebastopol : O'Reilly Media, 2022. 861 p.

13. Grey S. Mind-Melding Unity and Blender for 3D Game Development: Unleash the power of Unity and Blender to create amazing games. 1st ed. Birmingham : Packt Publishing, 2021. 460 p.

14. Hu C. Research on the integrated application of machine learning in Unity. *Applied and Computational Engineering*. 2024. Vol. 82, No. 1. pp. 161-166. DOI: 10.54254/2755-2721/82/20241033.

15. Lanham M. Learn Unity ML-Agents – Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games. Birmingham : Packt Publishing, 2018. 204 p.

16. Lapan M. Deep Reinforcement Learning Hands-On. 3rd ed. Birmingham : Packt Publishing, 2024. 716 p.

17. Lengyel E. Foundations of Game Engine Development. Vol. 2: Rendering. Lincoln : Terathon Software LLC, 2019. 412 p.

18. Liu Y. (Hayden). Python Machine Learning by Example: Unlock Machine Learning Best Practices with Real-World Use Cases. 4th ed. Birmingham : Packt Publishing, 2024. 518 p.

19. Millington I. AI for Games. 1st ed. Boca Raton : CRC Press, 2019. 1030 p.

20. Online Moqup. Wireframe & UI Prototyping Tool. Moqups. URL: https://moqups.com/ (access date: 13.10.2025).

21. Patent for the invention 11709864 USA, IPC (2023.01) G06N 3/08. Method for creating enhanced non-player characters in video games based on player behavior analysis / Activision Publishing, Inc. № US 11709864 B2; publ. 18.07.2023. URL: https://mpost.io/ru/activision-blizzard-granted-patent-for-ai-generated-npcs-pioneering-realism-in-video-games/ (access date: 08.09.2025).

22. Pearson S. Learn Blender Simulations the Right Way: Create attractive and realistic animations with Mantaflow, rigid and soft bodies, and Dynamic Paint. Birmingham : Packt Publishing, 2022. 368 p.

23. Raschka s., Mirjalili V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. 3rd ed. Birmingham : Packt Publishing, 2019. 770 p.

24. Raut U., Galchhaniya P., Nehete A., Shinde R., Bhoite A. Unity ML-Agents: Revolutionizing Gaming Through Reinforcement Learning. *Proceedings of the 2024 2nd World Conference on Communication & Computing (WCONF)*. 2024. pp. 1-7. DOI: 10.1109/WCONF61366.2024.10692314.

25. Rocha D. A., Duarte J. C. Simulating Human Behaviour in Games using Machine Learning. *Proceedings of 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 2019. pp. 163-172. DOI: 10.1109/SBGames.2019.00030.

26. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. Harlow : Pearson Education, 2009. 1152 p.

27. Sequence Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/sequence-diagram (access date: 09.10.2025).

28. Statechart Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/statechart-diagram (access date: 10.10.2025).

29. Sung K., Smith G. Basic Math for Game Development with Unity 3D: A Beginner's Guide to Mathematical Foundations. 2nd ed. New York : Apress, 2023. 575 p.

30. Unity – Manual: Unity User Manual 2022.3 (LTS). URL: https://docs.unity3d.com/Manual/index.html (access date: 15.09.2025).

31. Use Case Diagram. URL: https://docs.staruml.io/working-with-uml-diagrams/use-case-diagram (access date: 03.10.2025).

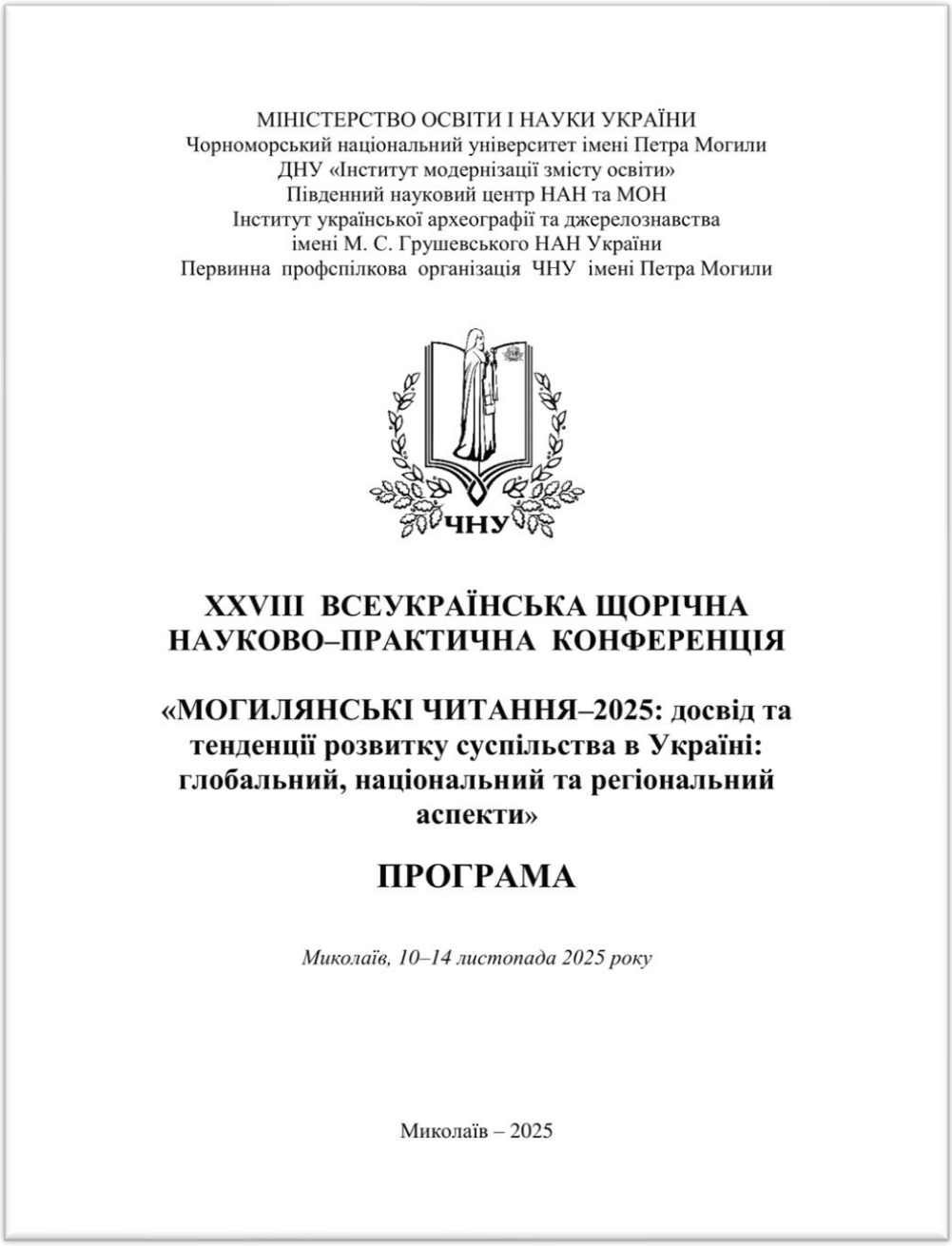
32. Video Game Genres: Everything You Need to Know. URL: https://www.hp.com/us-en/shop/tech-takes/video-game-genres (access date: 09.09.2025).

ДОДАТОК А

Апробація кваліфікаційної магістерської роботи

Результати досліджень було представлено на конференції:

Могилянські читання – 2025: досвід і тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти. Технічні науки : XXVIII Всеукр. наук.-практ. конф. (10–14 листоп. 2025 р., м. Миколаїв). Тези. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2025.



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ імені Петра Могили

ЧНУ

XXVIII ВСЕУКРАЇНСЬКА ЩОРІЧНА НАУКОВО–ПРАКТИЧНА КОНФЕРЕНЦІЯ

«МОГИЛЯНСЬКІ ЧИТАННЯ–2025: досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

ПРОГРАМА

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Рисунок А.1 – Обкладинка збірника тез доповідей конференції