КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Інформаційних технологій

(назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

на тему:
«Динамічна та персоналізована процедурна генерація ігрового контенту на основі машинного навчання»

Глінка Максим Васильович
(прізвище, ім’я та по батькові студента повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)
Інформаційних технологій
(назва кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри
д. т. н., доцент Гончаренко Т. А.
„___” _____2025 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

Динамічна та персоналізована процедурна генерація ігрового контенту на основі машинного навчання
(назва)

Виконав:
Глінка Максим Васильович
(прізвище, ім’я та по батькові повністю)
122. Комп’ютерні науки
(спеціальність)
Інформаційні управляючі системи та технології
(освітня програма)
Групи КН-21-2
Керівники Гончаренко Т. А. (1), Долгополов С. Ю. (2)
(прізвище та ініціали)
д.т.н., доцент (1), ас. (2)
(вчене звання, науковий ступінь)
Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій

Випускова кафедра: Інформаційних технологій

Освітній рівень: «бакалавр»

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри

д. т. н., доцент Гончаренко Т. А.

„___" _____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

Глінка Максим Васильович

(прізвище, ім'я та по батькові студента)

1. Тема роботи
Динамічна та персоналізована процедурна генерація ігрового контенту на основі машинного навчання
затверджена наказом ректора КНУБА № *** від **.**.202*
2. Керівник роботи Гончаренко Т. А., д. т. н., доцент; Долгополов С. Ю, ас.
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Строк подання студентом роботи до захисту _____
4. Зміст пояснювальної записки за розділами:
Р. 1. Теоретичні основи процедурної генерації ігрового контенту.
Р. 2. Методологічні засади проєктування системи персоналізованої процедурної генерації.
Р. 3. Програмно-технічне рішення динамічної та персоналізованої процедурної генерації ігрового контенту на основі машинного навчання.
Р. 4. Ергономічні аспекти розробленої системи.
5. Графічний матеріал за розділами:
Р. 1. Методи процедурної генерації, приклади генерації з використанням різних алгоритмів, схема циклу динамічної адаптації ігрового контенту
Р. 2. Схема архітектури персоналізованої системи, діаграма інформаційних потоків в адаптивному циклі, блок-схема алгоритму моніторингу та аналізу патернів поведінки гравця, блок-схема алгоритму динамічної адаптації

контенту, блок-схема алгоритму оцінки відповідності контенту профілю гравця, схема взаємодії системи та ігрового рушія.
Р. 3. Архітектура програмного продукту ігрового рушія, розроблені компоненти в ігровому рушії, діаграма класів основних структур даних, приклад згенерованого рівня та інформація про нього.
Р. 4. Інтерфейс розробника в ігровому рушії, принципи забезпечення позитивного досвіду гравця.

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	
Розділ 2	
Розділ 3	
Розділ 4	
Остаточне оформлення роботи	
Направлення роботи для перевірки на плагіат	
Попередній захист роботи на випусковій кафедрі	
Направлення роботи на рецензування	

7. Консультанти розділів кваліфікаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Гончаренко Т. А.		
Розділ 2	Гончаренко Т. А.		
Розділ 3	Долгополов С.Ю.		
Розділ 4	Долгополов С.Ю.		

8. Дата видачі завдання _____

Зав. кафедри _____ _____
(підпис) (прізвище та ініціали)
Керівник (1) _____ _____
(підпис) (прізвище та ініціали)
Керівник (2) _____ _____
(підпис) (прізвище та ініціали)
Здобувач _____ _____
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY)

РЕЗЮМЕ (SUMMARY) до кваліфікаційної випускної роботи Здобувача:	Глінка Максим Васильович Hlynka Maksym Vasylovych		
ЗВО	Київський національний університет будівництва і архітектури		
Тема (українською та англійською)	Динамічна та персоналізована процедурна генерація ігрового контенту на основі машинного навчання Dynamic and personalized procedural generation of game content based on machine learning		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускаюча кафедра	Інформаційних технологій		
Спеціальність	122 «Комп'ютерні науки»		
Освітня програма	Інформаційні управляючі системи та технології		
Керівники	Гончаренко Т. А. та Долгополов С. Ю.		
Обсяг роботи:	пояснювальна записка, стор.	розділів	слайди презентації
	105	4	15
Ключові слова: Keywords:	Процедурна генерація контенту, машинне навчання, персоналізація, динамічна адаптація, моделювання гравця, ігровий контент, досвід гравця, розробка ігор Procedural Content Generation, Machine Learning, Personalization, Dynamic Adaptation, Player Modeling, Game Content, Player Experience, Game Development		

Робота присвячена розробці та дослідженню системи динамічної та персоналізованої процедурної генерації ігрового контенту. Система використовує машинне навчання для аналізу поведінки гравця та адаптації ігрових рівнів, покращуючи досвід гравця, підвищуючи залученість та реіграбельність гри.

The work is dedicated to the development and research of a system for dynamic and personalized procedural generation of game content. The system uses machine learning to analyze player behavior and adapt game levels, improving player experience, increasing engagement and game replayability.

Здобувач: _____ / Максим ГЛІНКА /
Керівник (1): _____/ Тетяна ГОНЧАРЕНКО /
Керівник (2): _____/ Сергій ДОЛГОПОЛОВ /
"___" _____ 2025 р.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВОГО КОНТЕНТУ 11
1.1. Сучасний стан та еволюція методів процедурної генерації в ігровій індустрії 11
1.2. Аналіз традиційних підходів до процедурної генерації контенту 15
1.3. Огляд методів машинного навчання в контексті генерації контенту 22
1.4. Проблеми персоналізації та динамічної адаптації ігрового контенту 27
Висновки до розділу 1 32
РОЗДІЛ 2. МЕТОДОЛОГІЧНІ ЗАСАДИ ПРОЄКТУВАННЯ СИСТЕМИ ПЕРСОНАЛІЗОВАНОЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ 33
2.1. Обґрунтування методів вирішення поставленої задачі 33
2.2. Формалізація задачі динамічної та персоналізованої генерації 40
2.3. Проектування архітектури системи 44
2.4. Проектування основних алгоритмів системи 49
Висновки до розділу 2 60
РОЗДІЛ 3. ПРОГРАМНО-ТЕХНІЧНЕ РІШЕННЯ ДИНАМІЧНОЇ ТА ПЕРСОНАЛІЗОВАНОЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВОГО КОНТЕНТУ НА ОСНОВІ МАШИННОГО НАВЧАННЯ 62
3.1. Обґрунтування вибору інструментарію розробки 62
3.2. Архітектура та модульна структура програмного продукту 67
3.3. Реалізація компонентів системи 76
Висновки до розділу 3 84
РОЗДІЛ 4. ЕРГОНОМІЧНІ АСПЕКТИ РОЗРОБЛЕНОЇ СИСТЕМИ 85
4.1. Ергономічні вимоги до інтерфейсів системи процедурної генерації 85
4.2. Проектування інтерфейсу користувача 89
4.3. Принципи забезпечення позитивного досвіду взаємодії гравця з процедурно згенерованим контентом 93
Висновки до розділу 4 98
ЗАГАЛЬНІ ВИСНОВКИ 99
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 102

ВСТУП

Індустрія комп’ютерних ігор переживає період стрімкого зростання, що супроводжується постійно зростаючими очікуваннями гравців щодо масштабності, різноманітності та глибини ігрових світів. Забезпечення високого рівня реіграбельності та довготривалої залученості стає ключовим фактором конкурентоспроможності ігрових проектів. Традиційні методи ручного створення ігрового контенту вимагають значних часових та фінансових витрат, що часто обмежує можливості розробників у створенні дійсно великих та варіативних віртуальних середовищ.

Процедурна генерація контенту (PCG) виникла як потужний інструментарій для автоматизації створення ігрових ресурсів, дозволяючи генерувати великі обсяги контенту на основі алгоритмів. Однак класичні методи PCG, хоча й ефективні для створення різноманітності, часто страждають від передбачуваності, браку семантичного контролю та, що найважливіше, нездатності адаптуватися до індивідуальних особливостей гравця. Це призводить до того, що згенерований контент може не відповідати рівню навичок, стилю гри чи вподобанням користувача, знижуючи якість ігрового досвіду. Виникає нагальна потреба у розробці інтелектуальних систем PCG, здатних динамічно та персоналізовано адаптувати контент у реальному часі.

Актуальність даного дослідження зумовлена необхідністю подолання обмежень традиційної PCG та зростаючим попитом на глибоко персоналізовані ігрові досвіди. Використання методів машинного навчання (ML) відкриває принципово нові можливості для створення систем PCG, які можуть аналізувати поведінку гравця, будувати його модель та генерувати контент, що динамічно підлаштовується під його потреби, тим самим підвищуючи залученість, задоволення та підтримуючи оптимальний рівень виклику. Розробка таких систем є важливим кроком до створення по-справжньому інтерактивних та емерджентних ігрових світів.

Аналіз останніх досліджень та наукових праць. Сфера процедурної генерації та застосування штучного інтелекту в іграх активно розвивається. Фундаментальний внесок у систематизацію знань про PCG та ШІ в іграх зробили G. N. Yannakakis та J. Togelius [1]. N. Shaker та співавтори детально розглянули класичні та сучасні методи PCG [2]. Роботи A. Summerville та ін. [3] і J. Liu та ін. [4] заклали основи напрямку PCGML (PCG на основі машинного навчання), проаналізувавши потенціал глибокого навчання, зокрема GAN [5] та VAE [6], для генерації контенту. Дослідження A. Khalifa та співавторів [7] продемонстрували ефективність навчання з підкріпленням (RL) для адаптивної генерації рівнів. Питання моделювання гравців досліджувалися такими авторами, як A. Drachen [8], C. Holmgård [9] та M. T. Pedersen [10]. В Україні аспекти алгоритмів PCG, зокрема з використанням графів та генераторів шуму, розглядалися у працях В. Койбічук та ін. [11], Г.В. Марчука та ін. [12], І.Ф. Лайтарука та Т.О. Гришанович [13]. Однак питання створення інтегрованих систем, що поєднують динамічний аналіз поведінки гравця з персоналізованою генерацією контенту за допомогою сучасних ML-моделей в режимі реального часу, залишаються відкритими та потребують подальших досліджень.

Метою кваліфікаційної роботи є розробка та дослідження методів та архітектури системи для динамічної та персоналізованої процедурної генерації ігрового контенту на основі машинного навчання, здатної адаптувати характеристики контенту (зокрема, ігрових рівнів) відповідно до профілю та поведінки гравця для підвищення якості ігрового досвіду.

Виходячи з мети визначимо наступні завдання:

1) Провести аналітичний огляд сучасного стану процедурної генерації контенту, методів машинного навчання для PCG та проблем персоналізації і динамічної адаптації в іграх для виявлення існуючих обмежень та формулювання задачі дослідження.
2) Спроектувати систему динамічної та персоналізованої процедурної генерації контенту, включаючи обґрунтування вибору методів

машинного навчання, формалізацію задачі, проектування архітектури системи, її основних алгоритмів та інтерфейсів інтеграції.

3) Здійснити програмну реалізацію прототипу спроектованої системи з використанням обраного інструментарію, провести її технічне тестування та верифікацію, а також розробити дизайн та виконати експериментальні дослідження для оцінки ефективності запропонованого підходу щодо впливу на досвід гравця.
4) Дослідити ергономічні аспекти розробленої системи, визначити вимоги та принципи проектування інтерфейсів і адаптивної взаємодії, провести ергономічну оцінку та сформулювати рекомендації щодо покращення користувацького досвіду для розробників та гравців.

Об’єктом дослідження є процес динамічної та персоналізованої процедурної генерації ігрового контенту.

Предметом дослідження є методи, моделі машинного навчання та алгоритми, що забезпечують аналіз поведінки гравця та адаптивну генерацію ігрового контенту (зокрема, ігрових рівнів) відповідно до профілю гравця.

Методи дослідження. Для досягнення мети роботи використано методи системного аналізу (для проектування архітектури системи), порівняльного аналізу (для вибору методів PCG та ML), математичного моделювання (для формалізації задачі), методи машинного навчання (класифікація, кластеризація, нейронні мережі для моделювання гравця та прогнозування параметрів), методи процедурної генерації (алгоритми генерації рівнів), методи програмної інженерії (для реалізації програмного продукту в Unity/C#), методи експериментального дизайну та статистичного аналізу (для оцінки ефективності системи), методи експертних оцінок та аналізу користувацького досвіду (для оцінки ергономіки).

Наукова новизна роботи полягає у:

- Розробці інтегрованої архітектури системи, що поєднує модулі динамічного аналізу поведінки гравця, ML-класифікації та

кластеризації для формування профілю, та адаптивної процедурної генерації контенту на основі цього профілю.

- Запропонованому підході до адаптації параметрів генерації контенту, що комбінує визначення кластера гравця з потенційним використанням нейромережевого прогнозування та механізмами плавної інтерполяції для забезпечення непомітної та доцільної зміни ігрового досвіду.
- Обґрунтуванні та застосуванні комплексної методики оцінки ефективності систем персоналізованої PCG, що включає як технічну верифікацію та об’єктивні метрики, так і експериментальні дослідження з аналізом суб’єктивного досвіду гравців.

Практичне значення роботи роботи полягає у створенні програмного прототипу системи динамічної та персоналізованої PCG, реалізованого в середовищі Unity. Цей прототип демонструє можливість підвищення залученості гравців та покращення їхнього досвіду шляхом адаптації контенту. Розроблені підходи та компоненти можуть бути використані розробниками ігор для створення більш інтерактивних, реіграбельних та доступних для широкої аудиторії продуктів. Зібрані під час експериментів дані та результати аналізу можуть слугувати основою для подальших досліджень у сфері інтелектуальних ігрових систем. Реалізовані модулі аналізу поведінки гравців можуть використовуватися для збору цінної ігрової аналітики.

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, чотирьох розділів з висновками до кожного з них, загальних висновків, списку використаних джерел, що налічує 35 найменувань. Загальний обсяг роботи становить 105 сторінок. Робота містить 7 таблиць та 29 рисунків.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВОГО КОНТЕНТУ

1.1. Сучасний стан та еволюція методів процедурної генерації в ігровій індустрії

Процедурна генерація контенту (Procedural Content Generation, PCG) утвердилася як одна з ключових технологій у сучасній розробці відеоігор, що дозволяє автоматично створювати ігровий контент за допомогою алгоритмів замість ручного проектування. Цей підхід надає значні переваги, включаючи можливість створення масштабних та різноманітних ігрових світів, підвищення реіграбельності за рахунок унікальності кожної ігрової сесії, а також потенційне скорочення витрат часу та ресурсів на розробку. G. N. Yannakakis та J. Togelius у своїй фундаментальній праці зазначають, що PCG дозволяє створювати значно більші обсяги контенту, ніж це можливо при ручному створенні, відкриваючи шлях до безмежних ігрових світів та досвідів [1]. Як підкреслюють В. Койбічук та ін., розробка унікальних ігрових середовищ за допомогою алгоритмів PCG може суттєво скоротити витрати та підвищити продуктивність команди розробників, усуваючи ризик стагнації процесу [11]. Саме ці переваги зумовлюють невпинний інтерес до розвитку та вдосконалення методів PCG як з боку академічної спільноти, так і з боку ігрової індустрії.

Історично, перші застосування методів процедурної генерації були тісно пов’язані з обмеженнями обчислювальних ресурсів ранніх комп’ютерних систем. В епоху, коли обсяг пам’яті та швидкість процесорів були вкрай лімітовані, генерація контенту «на льоту» стала ефективним способом створення великих ігрових просторів, які фізично неможливо було зберегти. Класичними прикладами є ігри Rogue (1980), де підземелля генерувалися для кожної нової гри, та Elite (1984), що пропонувала гравцям дослідження величезної галактики з тисячами унікальних зоряних систем, згенерованих за допомогою простих математичних формул. M. Hendrikx та ін. у своєму огляді

відзначають, що ці ранні методи PCG переважно базувалися на генераторах псевдовипадкових чисел та відносно простих алгоритмічних правилах, спрямованих на створення лабіринтів, ландшафтів чи космічних систем [14]. Аналізуючи ці перші кроки, можна констатувати, що хоча тогочасні підходи й були відносно примітивними за сучасними мірками, вони заклали фундаментальні принципи PCG та продемонстрували величезний потенціал автоматизованої генерації для збагачення ігрового досвіду.

З плином часу та експоненційним зростанням потужності комп'ютерного обладнання методи процедурної генерації зазнали значної еволюції. Сфера їх застосування розширилася далеко за межі простої генерації рівнів чи карт. Розробники почали використовувати PCG для створення текстур (за допомогою фрактальних алгоритмів та шумів, як-от шум Перліна), моделей рослинності та архітектури (використовуючи L-системи), квестів, музичного супроводу та навіть елементів наративу [2, 15]. Г.В. Марчук та ін. у своєму огляді підкреслюють, що потреба у швидкому створенні нових ігрових світів стимулює активне вивчення та застосування різноманітних алгоритмів PCG [12]. N. Shaker, N. Togelius та M. J. Nelson у своїй книзі детально розглядають різноманіття застосувань PCG, підкреслюючи перехід від генерації статичного контенту до більш складних та динамічних елементів ігрового світу [2]. Цей етап еволюції демонструє зростаючу зрілість технології та її здатність вирішувати все ширше коло завдань у геймдеві. Водночас, саме на цьому етапі стали очевидними обмеження суто алгоритмічних підходів, особливо у питаннях створення контенту, що відчувається як «рукотворний», естетично привабливий та, що найважливіше, здатний адаптуватися до індивідуальних потреб гравця.

Сучасний етап розвитку процедурної генерації контенту нерозривно пов'язаний із впровадженням методів машинного навчання (Machine Learning, ML). Обмеження класичних підходів, такі як передбачуваність результатів, складність налаштування параметрів для досягнення бажаного стилю чи поведінки, та нездатність до глибокої адаптації, спонукали дослідників та

розробників до пошуку нових рішень. Парадигма процедурної генерації контенту на основі машинного навчання (Procedural Content Generation via Machine Learning, PCGML), як її описують A. Summerville та ін., полягає у використанні моделей ML, навчених на великих масивах існуючого контенту або даних про взаємодію гравців, для генерації нового контенту [3]. Цей підхід дозволяє створювати контент, що не лише відповідає певним структурним чи функціональним вимогам, але й наслідує стилістичні особливості навчальних даних, що відкриває шлях до генерації більш «людиноподібного» та естетично вивіреного контенту. Саме PCGML стає рушійною силою для досягнення цілей даної роботи, оскільки вона закладає основу для створення не просто різноманітного, а й персоналізованого та динамічно адаптивного ігрового досвіду.

На сьогоднішній день методи машинного навчання активно застосовуються для вирішення широкого спектру завдань у сфері PCG. Зокрема, генеративні змагальні мережі (Generative Adversarial Networks, GANs) та варіаційні автоенкодери (Variational Autoencoders, VAEs) успішно використовуються для генерації візуальних ресурсів, таких як текстури, спрайти та навіть 2D/3D моделі [4, 16]. Рекурентні нейронні мережі (Recurrent Neural Networks, RNNs) та їх більш просунуті варіанти, як-от трансформери, знаходять застосування у генерації послідовного контенту, наприклад, ігрових рівнів, музичних тем чи наративних структур [17, 18]. Методи навчання з підкріпленням (Reinforcement Learning, RL) застосовуються для створення агентів, здатних генерувати контент (наприклад, рівні або правила гри), що оптимізований під певні цілі, такі як підтримання залученості гравця або адаптація складності [7]. J. Liu та ін. у своєму огляді наголошують на значному потенціалі саме глибокого навчання для PCG, завдяки його здатності виявляти складні закономірності та ієрархічні структури в даних [4]. Дослідження G. N. Yannakakis та J. Togelius також підкреслюють зростаючу роль ML у створенні адаптивних ігрових систем, які можуть динамічно змінювати контент чи механіки у відповідь на дії та вподобання гравця [1]. Таким чином, сучасний

стан PCG характеризується переходом від жорстко закодованих алгоритмів до гнучких, керованих даними моделей, що дозволяє реалізовувати значно складніші та інтелектуальніші системи генерації, наближаючись до мети створення по-справжньому динамічного та персоналізованого ігрового досвіду.

Схематична часова шкала еволюції методів PCG зображена на рис. 1.1.

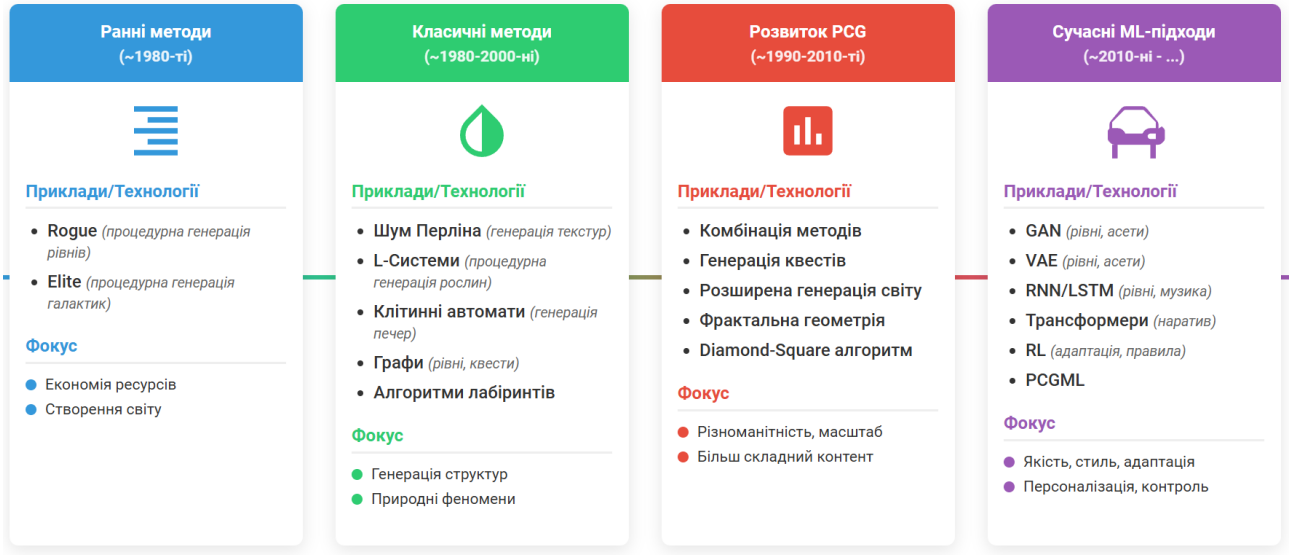

Рисунок 1.1. Еволюція методів процедурної генерації контенту (PCG)

Незважаючи на вражаючий прогрес, сучасна процедурна генерація контенту, особливо з використанням машинного навчання, все ще стикається з низкою відкритих питань та викликів. До них належать проблеми контрольованості та передбачуваності згенерованого контенту, інтерпретованість рішень, що приймаються ML-моделями, складність формалізації та оцінки якості згенерованого контенту (особливо таких суб'єктивних аспектів, як естетика чи «фановість»), а також інтеграція складних ML-моделей в реальні ігрові рушії та виробничі процеси. Тим не менш, очевидна еволюція PCG від простих генераторів випадкових чисел до складних нейромережевих архітектур свідчить про безперервний розвиток цієї галузі та її зростаючу важливість для майбутнього інтерактивних розваг. Подальше дослідження та розробка методів, зокрема тих, що поєднують переваги класичної PCG та потужність ML, є актуальним напрямком, що обіцяє

нові прориви у створенні глибоко персоналізованих та динамічно адаптивних ігрових світів, що й визначає основний фокус даної роботи.

1.2. Аналіз традиційних підходів до процедурної генерації контенту

До широкого розповсюдження методів машинного навчання, арсенал процедурної генерації контенту базувався переважно на наборі алгоритмічних технік, які стали класичними. Ці методи, що активно розвивалися з кінця XX століття, були спрямовані на вирішення конкретних завдань генерації різних типів ігрового контенту, таких як ландшафти, рівні, текстури та інші елементи ігрового світу. Їхня основна перевага полягала у можливості створення великих обсягів даних з відносно невеликого набору правил чи параметрів, що було особливо актуально в умовах обмежених ресурсів ранньої ігрової розробки [3].

Одним із найпоширеніших інструментів у класичній PCG є використання генераторів шуму. К. Перлін розробив свій знаменитий алгоритм шуму (Perlin noise) для генерації реалістичних текстур, що імітують природні нерегулярності [19]. Шум Перліна генерує псевдовипадкові, але гладкі та когерентні значення у багатовимірному просторі, що робить його ідеальним для створення природних ландшафтів, процедурних текстур та моделювання різних явищ [12]. Згодом К. Перлін удосконалив алгоритм до Покращеного шуму (Improved Noise) [20]. Інший варіант, Simplex Noise, також розроблений Перліном, є ефективнішим та менш «гучним», вирішує проблему спрямованості шуму Перліна та краще працює з великою кількістю вимірів [12]. Ці алгоритми часто використовуються у комбінації з фрактальним підходом (Fractal Noise), де кілька «октав» шуму з різними частотами та амплітудами накладаються одна на одну для досягнення високого рівня деталізації [12, 13]. І. Ф. Лайтарук та Т. О. Гришанович зазначають, що час обчислення шуму Перліна має складність $O(n_p * 2^n)$, де n_p – кількість точок, n – розмірність простору [13]. Аналізуючи цей підхід, варто відзначити його відносну простоту реалізації та контрольованість результату через параметри, однак генерація складних, структурованих об’єктів за допомогою лише шуму є проблематичною. Г. В.

Марчук та ін. у своєму порівняльному аналізі роблять висновок, що Simplex Noise є оптимальним за співвідношенням швидкості та якості, тоді як Fractal Noise забезпечує найвищу якість ціною значних обчислювальних ресурсів [12].

Приклад 2D карти висот (від чорного – низького, до білого – високого), згенерованої за допомогою алгоритму шуму Перліна з чотирма октавами представлений на рис. 1.2.

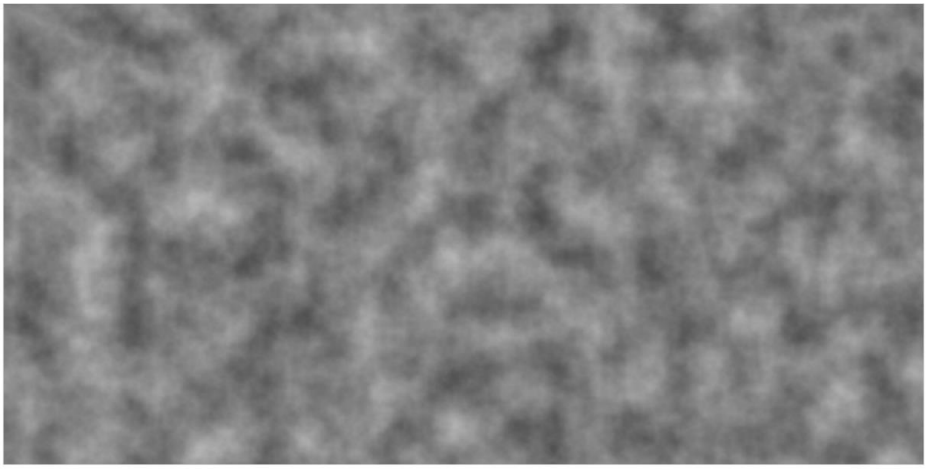

Рисунок 1.2. Приклад генерації 2D ландшафту шумом Перліна

Іншим важливим класом алгоритмів є клітинні автомати (Cellular Automata, CA). Це дискретні моделі, що складаються з регулярної сітки клітин, кожна з яких може перебувати в одному з кількох станів [13]. Стан клітини на наступному кроці часу визначається правилами, що залежать від стану самої клітини та її сусідів (визначених околом, наприклад, фон Неймана або Мура) на поточному кроці [13]. Найвідомішим прикладом є «Гра Життя» Дж. Конвея. В ігровій індустрії CA часто застосовуються для генерації двовимірних карт, особливо підземель та печер [15]. Ітеративне застосування правил, що симулюють процеси «народження» та «смерті» стін, може перетворити випадково заповнену сітку на органічні структури. І. Ф. Лайтарук та Т. О. Гришанович відзначають, що CA також широко використовуються для симуляції зміни положення рідин з часом, наводячи приклад гри Noita [13]. Вони також вказують, що часова складність однієї ітерації CA складає $O(n * r^2)$, а просторова – $O(n)$, де n – кількість клітин, r – радіус околу. Цей метод є

ефективним для створення певних типів топологій та симуляцій, однак контроль над кінцевою структурою може бути непрямим, а результати іноді потребують додаткової обробки для забезпечення ігрової функціональності [1].

Етапи генерації печероподібної структури за допомогою клітинного автомата: початковий випадковий шум, проміжний стан після кількох ітерацій згладжування, більш чітка структура після додаткових ітерацій представлені на рис. 1.3.

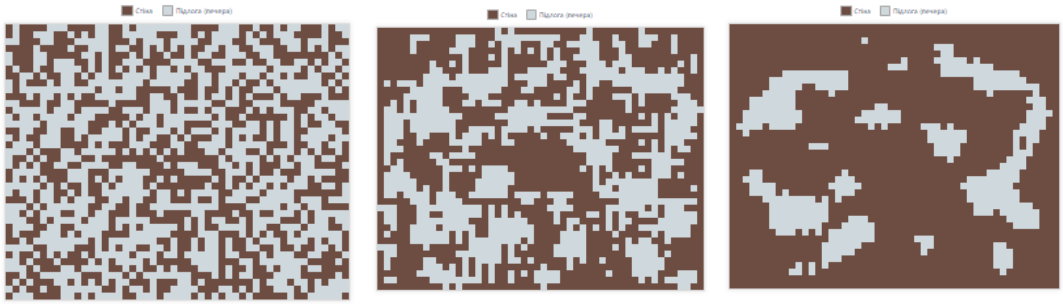

Рисунок 1.3. Етапи генерації печерної структури за допомогою клітинного автомата

L-системи (Системи Ліденмаєра), розроблені А. Ліденмаєром для моделювання росту рослин, є ще одним потужним інструментом традиційної PCG [21]. L-системи – це тип формальної граматики, що використовує правила для паралельної заміни символів у рядку (аксіомі). Кожен символ у рядку може бути інтерпретований як команда для побудови геометрії [15]. Завдяки своїй здатності моделювати процеси росту та самоподібні структури, L-системи широко використовуються для генерації реалістичної рослинності, фрактальних візерунків, а іноді й для проектування мереж доріг чи річок [15]. Перевагою L-систем є здатність генерувати дуже складні та візуально правдоподібні структури з компактного набору правил. Проте їх застосування здебільшого обмежене генерацією об’єктів з розгалуженою або фрактальною структурою.

Схематичне зображення фрактальної структури дерева, згенерованої за допомогою L-системи шляхом рекурсивного застосування правил перепису символів представлене на рис. 1.4.

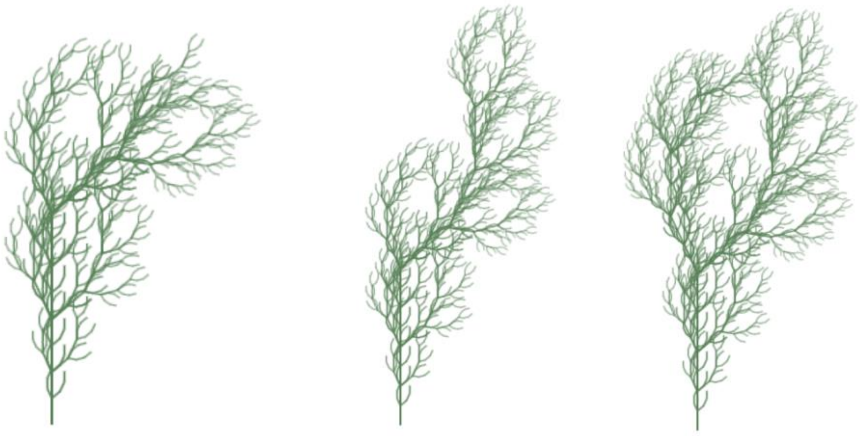

Рисунок 1.4. Приклад генерації дерева за допомогою L-системи

Методи на основі графів також відіграють важливу роль у класичній PCG. В. Койбічук та ін. підкреслюють наукову новизну використання графів для процедурної генерації ігрового контенту, що дозволяє моделювати складні структури та зв’язки, наприклад, при створенні підземель, квестів чи міських планів [11]. Графи можуть використовуватися як основа для різних алгоритмів генерації:

- **Алгоритми випадкових блукань (Random Walks).** Наприклад, алгоритм «Drunkard’s Walk», де агент випадковим чином переміщується по сітці, «прорізаючи» шляхи або кімнати. В. Койбічук та ін. аналізують та реалізують цей алгоритм, зазначаючи його придатність для недетермінованих ігрових середовищ [11]. A. Koesnaedi та W. Istiono також досліджували застосування цього алгоритму для генерації рівнів у roguelike-іграх [22]. Приклад візалізації алгоритму «Drunkard’s Walk» представлено на рис. 1.5.

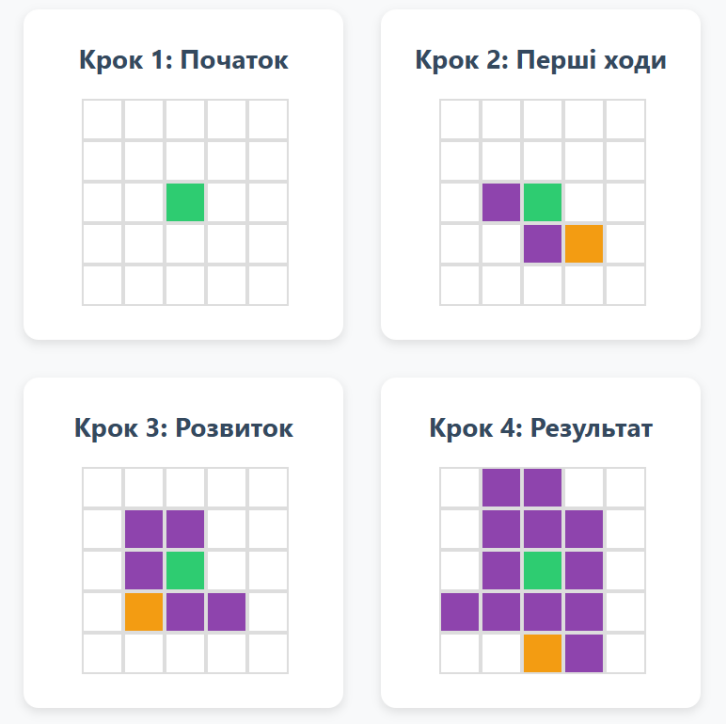

Рисунок 1.5. Принцип роботи алгоритму випадкових блукань

- **Бінарний поділ простору (Binary Space Partitioning, BSP).** Метод, що рекурсивно ділить простір на менші області за допомогою прямих ліній, створюючи деревоподібну структуру [11]. Цей підхід часто використовується для генерації підземель та будівель, забезпечуючи хорошу структурованість та контроль над розміщенням кімнат. В. Койбічук та ін. реалізують та візуалізують роботу BSP [11]. C. D. Cóth розглядає нові розробки в області BSP [23]. Приклад поділу простору за допомогою BSP представлений на рис. 1.6.

Рисунок 1.6. Принцип поділу простору за допомогою BSP

- **Граматики графів та переписування графів (Graph Grammars / Graph Rewriting).** Аналогічно до L-систем, але застосовуються до графів. Правила визначають, як замінювати підграфи на інші, дозволяючи генерувати складні структури, такі як міста чи замки [13]. І. Ф. Лайтарук та Т. О. Гришанович обговорюють алгебраїчний підхід до переписування графів (single/double pushout) та метод П. Меррелла, заснований на прикладах [13].
- **Методи розподілу простору (Space Distribution).** Окрім BSP, існують інші методи розподілу ігрового простору. Діаграми Вороного (Voronoi Diagrams) ділять простір на регіони на основі близькості до набору заданих точок («сайтів») [13]. І. Ф. Лайтарук та Т. О. Гришанович описують алгоритм Форчуна для побудови діаграм Вороного зі складністю $O(n \log n)$ та розглядають використання різних метрик відстані (Манхеттенської, Евклідової, Мінковського) для отримання країв різної форми [13]. Діаграми Вороного можуть використовуватися для генерації біомів, розподілу ресурсів або визначення зон впливу [13].

Інші класичні алгоритми включають генерацію лабіринтів (наприклад, алгоритм випадкового обходу в глибину), фрактальну генерацію (наприклад, Diamond-Square – алгоритм для генерації карт висот, що вимагає розміру сітки 2^{n+1}, та різноманітні граматичні підходи [12].

Незважаючи на свою ефективність та широке застосування, традиційні алгоритмічні методи процедурної генерації мають низку суттєвих обмежень, які стають особливо помітними при спробі створити глибоко персоналізований та динамічно адаптивний контент.

Одним з ключових недоліків є відсутність семантичного розуміння та обмеженість контролю високого рівня. Алгоритми генерують контент, суворо дотримуючись закладених правил, але не «розуміють» його змісту чи впливу на гравця [15]. Як зазначають N. Shaker та ін., розробник зазвичай контролює процес генерації через низькорівневі параметри (наприклад, частоту шуму,

правила CA, кути повороту в L-системах), тоді як досягнення конкретних високорівневих цілей (наприклад, певного темпу гри, емоційного настрою, стилістичної відповідності) вимагає значних зусиль з налаштування та експериментів [2]. Це ускладнює створення контенту, який би точно відповідав дизайнерському задуму або динамічно адаптувався до мінливих обставин.

Іншою проблемою є потенційна передбачуваність та повторюваність результатів. Хоча метою PCG є створення різноманітності, базові алгоритми можуть генерувати контент, що має впізнавані структурні патерни. Гравці, особливо досвідчені, можуть розпізнати основний алгоритм генерації, що призводить до зниження відчуття новизни та дослідження [1]. Наприклад, ландшафти, згенеровані лише шумом Перліна або алгоритмом Diamond-Square, можуть виглядати одноманітно або мати характерну «прямолінійність» [12]. Клітинні автомати також можуть призводити до схожих структур, а їх контроль є непрямим [1, 13]. Для подолання цього розробникам доводиться комбінувати різні методи та додавати значну кількість «ручних» доробок.

Також класичним методам властива складність у відтворенні специфічних естетичних стилів або «рукотворного» відчуття. Алгоритми добре справляються зі створенням природних або фрактальних структур, але їм важко імітувати нюанси та непередбачувані деталі, які вносить людський дизайнер. Контент, згенерований суто алгоритмічно, іноді може сприйматися як занадто «математичний», стерильний або позбавлений індивідуальності [3]. Досягнення певного художнього стилю за допомогою лише традиційних PCG інструментів є нетривіальною задачею.

Обчислювальна складність та вимоги до ресурсів також можуть бути обмеженням. Як зазначають Г. В. Марчук та ін., Fractal Noise, хоч і забезпечує високу якість, вимагає значних ресурсів [12]. І. Ф. Лайтарук та Т. О. Гришанович вказують на потенційно експоненційну складність переписування графів у найгіршому випадку через NP-повноту задачі пошуку гомоморфізму підграфів [13]. Алгоритм Diamond-Square має обмеження на розмір сітки [12].

Нарешті, найважливішим обмеженням у контексті даної роботи є низька здатність до динамічної адаптації та персоналізації. Класичні методи PCG зазвичай застосовуються або на етапі розробки, або один раз на початку ігрової сесії для створення статичного контенту. Реалізація механізмів, які б дозволили контенту динамічно змінюватися у відповідь на дії, навички чи вподобання конкретного гравця, є складною задачею для традиційних підходів. Вони не мають вбудованих механізмів для аналізу поведінки гравця або навчання на основі цього аналізу для коригування процесу генерації. Як підкреслюють G. N. Yannakakis та J. Togelius, саме потреба в адаптивності та персоналізації стала одним з головних рушіїв для інтеграції машинного навчання в PCG [1, 3].

Таким чином, аналіз традиційних підходів до процедурної генерації контенту показує їхню фундаментальну важливість та ефективність для вирішення багатьох завдань геймдеву. Однак їхні внутрішні обмеження, пов’язані з контролем, передбачуваністю, стилізацією та, особливо, адаптивністю, вказують на необхідність застосування більш потужних та гнучких методів, таких як ті, що базуються на машинному навчанні.

1.3. Огляд методів машинного навчання в контексті генерації контенту

Обмеження традиційних алгоритмічних методів процедурної генерації, детально розглянуті у попередньому підрозділі, зокрема їхня недостатня гнучкість, складність досягнення бажаної естетики та відсутність механізмів адаптації, стимулювали активний пошук нових підходів. У цьому контексті методи машинного навчання (ML) відкрили нову еру в процедурній генерації, започаткувавши напрям, відомий як процедурна генерація контенту на основі машинного навчання (Procedural Content Generation via Machine Learning, PCGML) [3]. На відміну від класичних підходів, де правила генерації задаються розробником вручну, PCGML використовує алгоритми, здатні навчатися на основі даних – це може бути існуючий ігровий контент (рівні, моделі, текстури, наративи), дані про взаємодію гравців або комбінація того й іншого [3]. Як

зазначають J. Liu та ін., здатність ML-моделей виявляти складні закономірності та приховані структури в даних дозволяє генерувати контент, що є не лише різноманітним, але й стилістично узгодженим, потенційно більш правдоподібним та, що є ключовим для даної роботи, адаптивним [4].

Одними з найпотужніших інструментів у арсеналі PCGML є глибокі генеративні моделі, зокрема генеративні змагальні мережі (Generative Adversarial Networks, GANs) та варіаційні автоенкодери (Variational Autoencoders, VAEs). Ці моделі навчаються апроксимувати розподіл даних у навчальній вибірці та генерувати нові зразки, що схожі на вихідні дані.

Ілюстративний приклад фрагменту ігрового рівня у стилі Super Mario Bros, згенерованого за допомогою генеративної змагальної мережі (GAN) представлений на рис. 1.7.

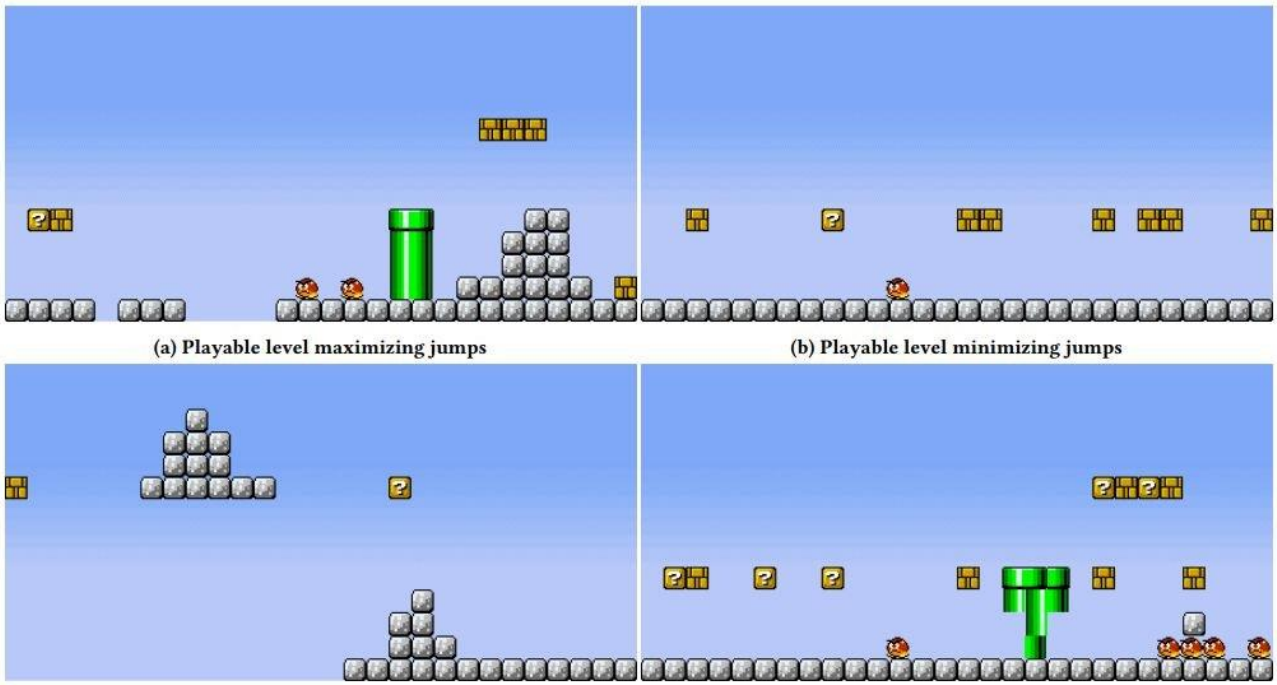

Рисунок 1.7. Приклад ігрового рівня, згенерованого за допомогою GAN

Генеративні змагальні мережі (GANs), запропоновані I. Goodfellow та ін., складаються з двох нейронних мереж – генератора та дискримінатора – що навчаються у змагальний спосіб [21]. Генератор намагається створити реалістичні зразки даних (наприклад, зображення рівня), а дискримінатор – відрізнити згенеровані зразки від реальних. У процесі навчання обидві мережі

вдосконалюються, що в ідеалі призводить до генератора, здатного створювати високоякісний контент. У сфері PCG GANs успішно застосовуються для генерації 2D-графіки, такої як спрайти персонажів чи об’єктів, текстури, а також для створення карт та рівнів [5, 16]. Наприклад, V. Volz та ін. продемонстрували використання GAN для генерації рівнів гри Super Mario Bros, які нагадували рівні, створені людиною [5]. Аналізуючи застосування GAN, варто відзначити їхню здатність генерувати візуально переконливий та часто несподіваний контент, що може імітувати певний стиль. Однак тренування GAN може бути нестабільним, а контроль над процесом генерації (наприклад, забезпечення виконання певних ігрових обмежень, як-от зв’язність рівня) залишається складним завданням [4], що вимагає розробки специфічних архітектур або методів керування генерацією. Варіаційні автоенкодери (VAEs), представлені D. P. Kingma та M. Welling, є іншим типом генеративних моделей, що базуються на ідеї кодування вхідних даних у латентний простір меншої розмірності та подальшого декодування з цього простору для реконструкції вхідних даних або генерації нових [6]. VAE навчаються не лише відтворювати дані, але й моделювати розподіл ймовірностей у латентному просторі, що дозволяє генерувати нові зразки шляхом семплування з цього простору. В PCG VAEs використовуються для схожих завдань, що й GANs, наприклад, для генерації рівнів або візуальних асетів. Перевагою VAEs порівняно з GANs часто є більш стабільне навчання та краще структурований латентний простір, що полегшує контроль над генерацією шляхом маніпуляцій у цьому просторі [4]. Проте, згенеровані VAE зразки іноді можуть бути менш чіткими або «розмитими» порівняно з результатами GAN. Дослідження, як-от робота A. Sarkar та ін. з генерації рівнів Doom, показують потенціал VAE для створення функціонального контенту, але також вказують на потребу вдосконалення для досягнення рівня якості та різноманітності, необхідного для сучасних ігор [16]. Дослідження J. S. Ryan та ін. демонструють використання просторових GAN для генерації даних про висоту на основі регіонального навчання [24].

Інший важливий клас ML-моделей для PCG пов’язаний з генерацією послідовних даних, таких як ігрові рівні (як послідовність сегментів або подій), музика, діалоги чи сюжетні лінії. Для таких завдань часто використовуються рекурентні нейронні мережі (RNNs) та їх більш просунуті варіанти, як-от мережі з довгою короткочасною пам’яттю (Long Short-Term Memory, LSTM), а також архітектури трансформерів.

Рекурентні нейронні мережі (RNNs) та LSTM розроблені спеціально для обробки послідовностей, оскільки вони мають внутрішню пам’ять (стан), яка дозволяє враховувати попередні елементи послідовності при генерації наступного [25]. Це робить їх природним вибором для завдань PCG, де важливий контекст та залежності між елементами. Наприклад, A. Summerville та M. Mateas використовували LSTM для генерації рівнів Super Mario Bros шляхом навчання на існуючих рівнях, представлених як послідовності символів [17]. Інші дослідження застосовували RNN/LSTM для генерації ігрової музики, текстових квестів та діалогів. Сильною стороною цих моделей є їхня здатність вивчати локальні закономірності та структуру в послідовних даних. Однак, класичні RNN страждають від проблеми згасання градієнтів, що ускладнює вивчення довгострокових залежностей, хоча LSTM значною мірою цю проблему вирішують [25]. Тим не менш, генерація дуже довгих та глобально когерентних послідовностей (наприклад, складних сюжетних ліній) може залишатися викликом. Трансформери, вперше представлені A. Vaswani та ін. для задач машинного перекладу, здійснили революцію в обробці природної мови та все активніше застосовуються в PCG [18]. Трансформери використовують механізм уваги (attention mechanism), який дозволяє моделі зважувати важливість різних частин вхідної послідовності при генерації кожного елемента вихідної, що ефективно вирішує проблему довгострокових залежностей. Завдяки своїй потужності, трансформери демонструють вражаючі результати в генерації тексту, включаючи ігровий наратив, діалоги та описи об’єктів. M. Roemmele досліджував використання трансформерних моделей для генерації інтерактивних наративів [26]. Потенційно, трансформери можуть

застосовуватися і для генерації інших типів послідовного контенту, таких як рівні чи музика, хоча це менш досліджена область порівняно з текстом. Недоліками трансформерів є їхня висока обчислювальна складність та потреба у великих обсягах даних для навчання. Проте, їхня здатність моделювати складні залежності робить їх надзвичайно перспективними для створення глибоких та змістовних ігрових елементів.

Генетичні алгоритми (Genetic Algorithms, GA), хоча й часто розглядаються окремо, також належать до сфери обчислювального інтелекту та знаходять застосування в PCG. GA – це еволюційні алгоритми, що використовують принципи природного відбору (схрещування, мутація, селекція) для пошуку оптимальних рішень [13]. У контексті PCG, «особиною» (хромосомою) може бути представлення ігрового контенту (наприклад, рівень, набір правил), а «пристосованість» (fitness) визначається функцією, що оцінює якість цього контенту за певними критеріями (наприклад, іграбельність, складність, естетика) [13]. І. Ф. Лайтарук та Т. О. Гришанович описують застосування GA для генерації будівель, підземель та світів, пов'язаних з ігровою логікою, відзначаючи їх універсальність та можливість чіткого визначення бажаних характеристик через фітнес-функцію [13]. Складність GA оцінюється як $O(GMN)$, де G – кількість поколінь, M – кількість генів особини, N – розмір популяції [13]. E. Soares de Lima та ін. використовували GA у поєднанні з автоматизованим плануванням для генерації ігрових квестів [27]. Хоча GA не є «генеративними моделями» у тому ж сенсі, що GAN/VAE, вони надають потужний механізм для пошуку та оптимізації в просторі можливого контенту, що може бути особливо корисним для задач адаптації, де фітнес-функція залежить від гравця.

Навчання з підкріпленням (Reinforcement Learning, RL) є ще одним важливим напрямком ML, що знаходить застосування в PCG, особливо для завдань, пов'язаних з адаптацією та оптимізацією. В RL агент навчається приймати рішення шляхом взаємодії з середовищем та отримання сигналів винагороди або покарання за свої дії [28]. У контексті PCG, RL-агент може

виступати в ролі генератора контенту (наприклад, дизайнера рівнів), а середовищем може бути симуляція гри або модель гравця. Агент навчається генерувати такий контент, який максимізує певну функцію винагороди, наприклад, залученість гравця, складність, що відповідає навичкам гравця, або новизну. A. Khalifa та ін. продемонстрували використання RL для генерації рівнів у різних іграх, де агент навчався створювати рівні певної складності або іграбельності. RL є особливо привабливим для динамічної адаптації контенту, оскільки агент може навчатися коригувати генерацію в реальному часі на основі спостережуваної поведінки гравця [1, 7]. Це тісно пов'язано з цілями даної роботи щодо персоналізації та динамічності. Однак, розробка ефективних функцій винагороди для складних цілей (наприклад, «фановість» рівня) є нетривіальною задачею, а навчання RL-агентів може вимагати значних обчислювальних ресурсів та часу.

Підсумовуючи огляд методів машинного навчання для PCG, можна констатувати їхній величезний потенціал для подолання обмежень традиційних підходів. Генеративні моделі дозволяють вивчати та відтворювати стилі, послідовні моделі – генерувати структурований контент, а навчання з підкріпленням відкриває шлях до адаптивної генерації, орієнтованої на гравця. Саме комбінація цих підходів, як вважає автор, може стати основою для створення систем динамічної та персоналізованої процедурної генерації ігрового контенту, що є предметом дослідження даної роботи.

1.4. Проблеми персоналізації та динамічної адаптації ігрового контенту

Поява потужних методів машинного навчання в арсеналі процедурної генерації контенту відкрила шлях до реалізації амбітної мети – створення ігрових світів, які не лише різноманітні, але й здатні динамічно адаптуватися до унікальних характеристик кожного гравця. Персоналізація та динамічна адаптація розглядаються як ключові фактори для підвищення залученості, підтримки інтересу протягом тривалого часу (реіграбельність), забезпечення

доступності для гравців з різним рівнем навичок та, зрештою, для створення більш глибокого та значущого ігрового досвіду [1]. Однак, реалізація ефективних систем персоналізації та адаптації на основі PCGML є складним завданням, що пов’язане з низкою фундаментальних проблем, які потребують вирішення. Ці проблеми стосуються як розуміння гравця, так і механізмів впливу на ігровий контент.

Основою будь-якої системи персоналізації чи адаптації є здатність гри «розуміти» гравця. Це вимагає розробки моделей гравця (player models) – формальних представлень, що фіксують релевантні характеристики користувача, такі як його навички, знання, вподобання, стиль гри, емоційний стан чи цілі [1]. Побудова таких моделей нерозривно пов’язана зі збором та аналізом даних про поведінку гравця під час ігрового процесу. Ці дані можуть включати широкий спектр інформації: від низькорівневих дій (натискання клавіш, рух миші) до високорівневих показників (час проходження рівня, кількість спроб, зібрані ресурси, вибір стратегій, частота використання певних механік). A. Drachen, A. Canossa та G. N. Yannakakis ще на прикладі Tomb Raider: Underworld продемонстрували, як аналіз телеметрії гравця може бути використаний для виявлення різних стилів гри за допомогою методів кластеризації [8]. Сучасні підходи все частіше використовують методи машинного навчання для побудови моделей гравців. Наприклад, класифікатори можуть визначати рівень майстерності гравця, регресійні моделі – прогнозувати ймовірність його «відтоку» з гри, а методи навчання без учителя – виявляти неочевидні патерни поведінки чи формувати «ігрові персони» [9]. Схема циклу динамічної адаптації ігрового контенту зображена на рис 1.8.

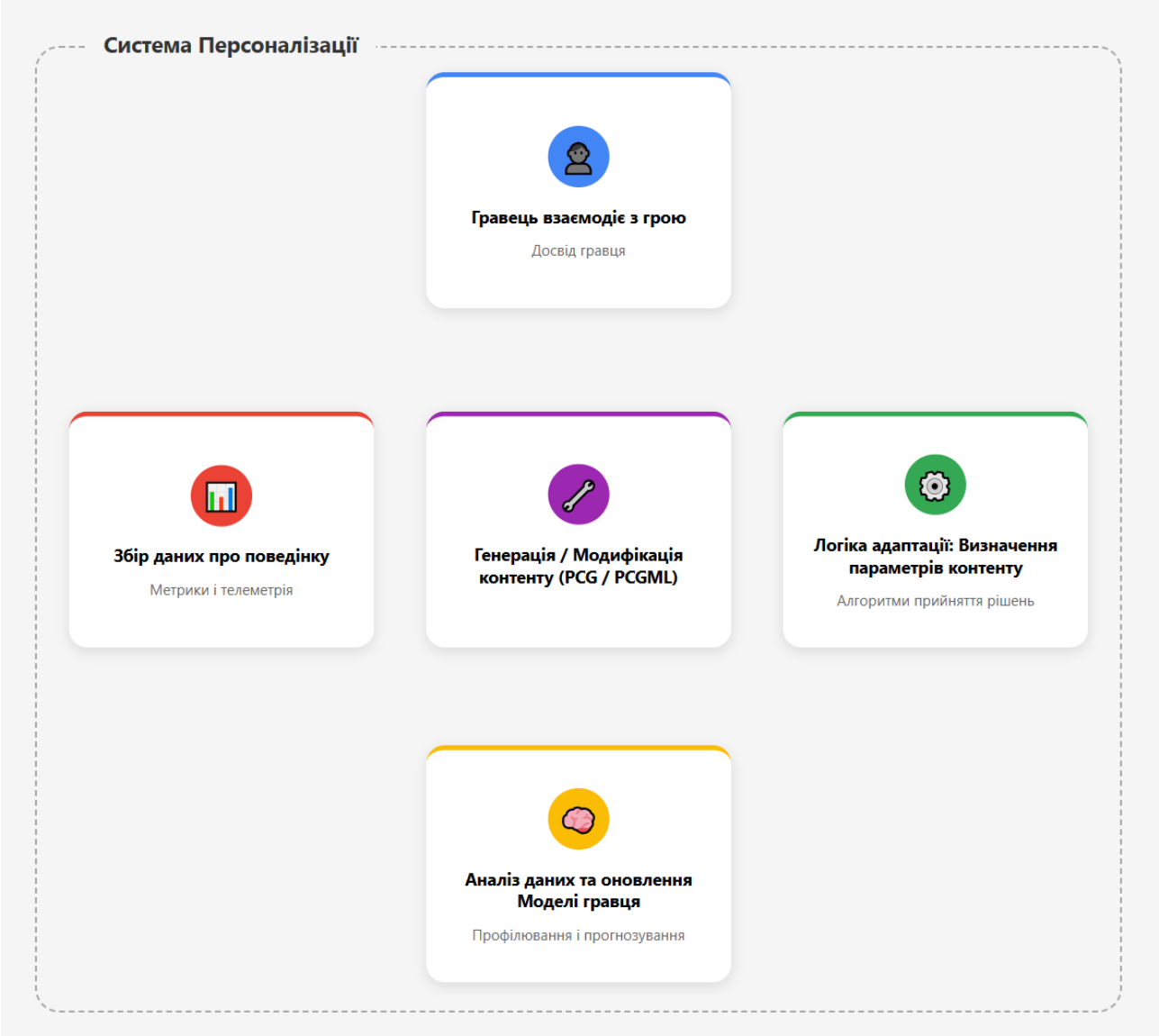

Рисунок 1.8. Схема циклу динамічної адаптації ігрового контенту

Незважаючи на прогрес, моделювання гравця залишається значним викликом. По-перше, необхідно визначити, які саме аспекти поведінки гравця є найбільш значущими для конкретного завдання адаптації та як їх надійно виміряти. По-друге, моделі повинні бути динамічними, тобто здатними оновлюватися в реальному часі або з невеликою затримкою, щоб відображати зміни у стані та навичках гравця. По-третє, постають питання точності, надійності та інтерпретованості моделей, особливо тих, що побудовані за допомогою складних ML-алгоритмів [1]. Недостатньо точна або неправильно інтерпретована модель може призвести до неадекватної адаптації, що негативно вплине на досвід гравця. По-четверте, збір та використання детальних даних

про поведінку гравців порушує важливі етичні питання та питання конфіденційності, які необхідно враховувати при проектуванні таких систем. Розробка точних, динамічних та інтерпретованих моделей гравця є критично важливою передумовою для успішної реалізації персоналізованої PCG, і саме ML-підходи надають найбільш перспективні інструменти для вирішення цієї задачі, хоча й потребують ретельного дослідження та валідації.

Маючи модель гравця, наступним кроком є використання цієї інформації для модифікації ігрового контенту чи механік з метою покращення досвіду гравця. Цей процес називається динамічною адаптацією. Одним з найвідоміших та найраніше досліджених прикладів є динамічне налаштування складності (Dynamic Difficulty Adjustment, DDA). Мета DDA – підтримувати оптимальний рівень виклику для гравця, уникаючи ситуацій, коли гра стає занадто легкою (що призводить до нудьги) або занадто складною (що викликає фрустрацію) [29]. DDA-системи зазвичай маніпулюють параметрами гри, такими як характеристики ворогів (здоров’я, швидкість, шкода), кількість ресурсів, доступних гравцю, або навіть складність головоломок, базуючись на поточній продуктивності гравця (наприклад, відсоток перемог, швидкість проходження). R. Hunicke обґрунтовувала важливість DDA для підтримки стану «потоку» (flow), але також вказувала на потенційні ризики: якщо адаптація занадто очевидна, гравець може відчути, що його зусилля не мають значення, або що гра «піддається» йому, що руйнує відчуття досягнення [29].

Сучасні підходи до адаптації виходять за рамки простого регулювання складності. Персоналізація може стосуватися значно ширшого спектру аспектів ігрового досвіду. Наприклад, система може адаптувати тип контенту відповідно до вподобань гравця: генерувати більше дослідницьких секцій для гравця, який любить досліджувати, або більше бойових зіткнень для гравця, орієнтованого на екшн. G. N. Yannakakis та J. Togelius описують концепцію Experience-Driven PCG, де генерація контенту безпосередньо керується бажаним емоційним станом або досвідом гравця [1]. Вона може включати адаптацію візуального стилю, музичного супроводу, наративних елементів (наприклад, генерування

квестів або діалогів, що відповідають стилю гри чи попереднім рішенням гравця) або навіть самих ігрових механік. Методи навчання з підкріпленням [7], розглянуті раніше, є особливо перспективними для таких завдань, оскільки RL-агент може навчатися генерувати контент, що максимізує показники, пов'язані з бажаним досвідом гравця (наприклад, час гри, позитивні відгуки, досягнення певних ігрових станів).

Однак, реалізація таких складних адаптивних систем пов'язана з новими викликами. Проблема контролю та узгодженості стає ще гострішою: як забезпечити, щоб адаптивно згенерований контент залишався функціональним, логічним та відповідав загальному стилю та наративу гри? Проблема оцінки ефективності адаптації також ускладнюється: як виміряти, чи дійсно адаптація покращує досвід гравця, особливо коли йдеться про суб'єктивні аспекти, такі як задоволення, занурення чи «фан»? A. Summerville та ін. підкреслюють, що оцінка PCGML-систем, особливо адаптивних, є однією з ключових відкритих проблем [3]. Крім того, інтеграція модулів моделювання гравця, логіки адаптації та генерації контенту в єдину систему, здатну працювати в реальному часі, є складним інженерним завданням.

Таким чином, проблеми персоналізації та динамічної адаптації ігрового контенту охоплюють два взаємопов'язані аспекти: глибоке розуміння гравця через моделювання його поведінки та розробка механізмів для ефективного та значущого впливу на ігровий контент за допомогою процедурної генерації. Класичні методи PCG не надають достатніх інструментів для вирішення цих проблем, тоді як методи машинного навчання пропонують потужні засоби як для моделювання гравців, так і для генерації адаптивного контенту. Проте, залишається низка відкритих питань щодо надійності, контрольованості, оцінки та інтеграції таких систем. Саме на подолання цих викликів шляхом розробки інтегрованої системи персоналізованої та динамічної PCG на основі ML і спрямоване дане дипломне дослідження. Розроблена концептуальна схема системи представлена на рис. 1.9.

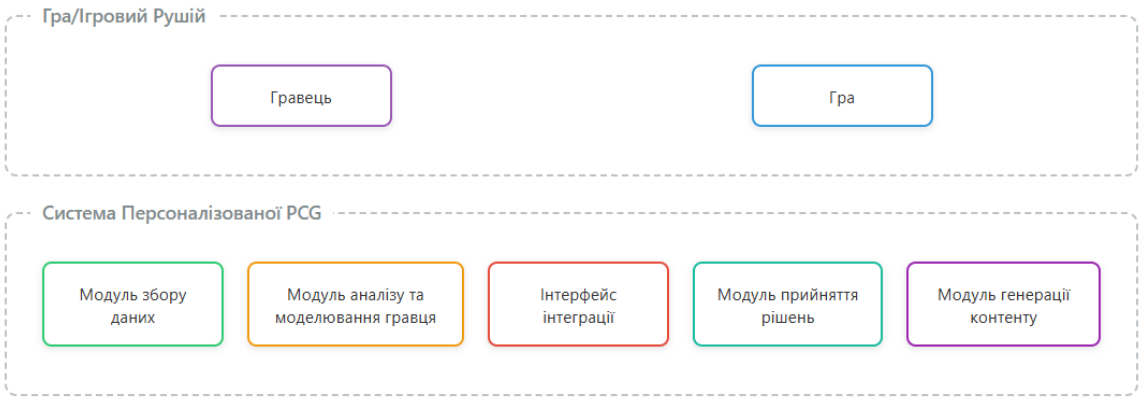

Рисунок 1.9. Концептуальна схема проектованої системи, що показує основні модулі

Висновки до розділу 1

Отже, у першому розділі представлено аналіз, який показує, що процедурна генерація контенту (PCG) пройшла значний шлях еволюції від ранніх алгоритмів до сучасних методів, де ключову роль відіграє машинне навчання (ML), спрямоване на подолання обмежень класичних підходів. Встановлено, що традиційні алгоритми, такі як генератори шумів, клітинні автомати, L-системи та методи на основі графів, ефективні для певних завдань генерації, проте характеризуються недостатньою контрольованістю, потенційною передбачуваністю та низькою здатністю до динамічної адаптації та персоналізації контенту під гравця. На противагу цьому, методи ML, зокрема генеративні моделі (GAN, VAE), послідовні моделі (RNN/LSTM, трансформери), генетичні алгоритми та навчання з підкріпленням (RL), демонструють значний потенціал для створення більш якісного, стилізованого та адаптивного контенту завдяки їхній здатності навчатися на даних та оптимізувати генерацію під задані цілі. Разом з тим, аналіз висвітлює ключові проблеми у реалізації ефективної персоналізації, що пов’язані зі складністю точного та динамічного моделювання гравця та розробкою надійних, контрольованих механізмів адаптації. Проведений аналіз обґрунтовує актуальність дослідження та визначає основну задачу роботи – розробку інтегрованої системи динамічної та персоналізованої PCG на основі ML.

РОЗДІЛ 2.
МЕТОДОЛОГІЧНІ ЗАСАДИ ПРОЄКТУВАННЯ СИСТЕМИ ПЕРСОНАЛІЗОВАНОЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

2.1. Обґрунтування методів вирішення поставленої задачі

Як було встановлено в аналітичному огляді, досягнення мети розробки системи динамічної та персоналізованої процедурної генерації ігрового контенту вимагає інтеграції підходів до моделювання гравця, генерації контенту та механізмів адаптації. Поставлені задачі дослідження передбачають не лише огляд існуючих методів, але й обґрунтований вибір конкретних технологій та алгоритмів, які будуть покладені в основу проектованої системи. Цей вибір має враховувати специфіку завдання, зокрема потребу аналізувати поведінку гравця в динаміці та генерувати складний, структурований контент (наприклад, ігрові рівні), що адаптується до виявлених характеристик гравця.

Ефективна персоналізація та адаптація неможливі без розуміння гравця. Як зазначалося раніше, для цього необхідно розробити модель гравця, що відображає його релевантні характеристики. Враховуючи складність та багатогранність людської поведінки в ігровому середовищі, а також великі обсяги даних, які можна зібрати під час гри (телеметрія), методи машинного навчання видаються найбільш придатними для побудови таких моделей. Вони здатні виявляти неочевидні патерни та залежності в даних, які важко формалізувати за допомогою експертних правил [1, 9].

Для вирішення задачі моделювання гравця у контексті даної роботи пропонується використовувати комбінацію методів керованого та некерованого навчання.

Методи керованого навчання (Supervised Learning), такі як класифікація та регресія, можуть бути використані для прогнозування конкретних характеристик гравця, якщо є можливість зібрати відповідним чином розмічені дані. Наприклад, можна навчати класифікатор (наприклад, на основі дерев рішень (Decision Trees), методу опорних векторів (Support Vector Machines,

SVM) або нейронних мереж (Neural Networks)) для визначення стилю гри гравця (наприклад, «дослідник», «агресор», «обережний») або його поточного рівня навичок (наприклад, «новачок», «досвідчений», «експерт»). Дерева рішень, як показують дослідження M. T. Pedersen та ін. у контексті передбачення поведінки гравців, мають перевагу в інтерпретованості, що важливо для розуміння того, на основі яких факторів система приймає рішення щодо адаптації [10]. SVM та нейронні мережі можуть забезпечити вищу точність класифікації при роботі зі складними, багатовимірними даними телеметрії [1]. Регресійні моделі можуть використовуватися для прогнозування кількісних показників, таких як ймовірність успішного проходження наступного сегменту рівня або очікуваний час на його завершення. Таким чином, застосування методів керованого навчання дозволяє отримати конкретні оцінки стану чи типу гравця, які можуть безпосередньо використовуватися для керування адаптацією.

Методи некерованого навчання (Unsupervised Learning), зокрема кластеризація (наприклад, за допомогою алгоритму k-середніх (k-means) або DBSCAN), можуть бути корисними для виявлення природних груп гравців зі схожими патернами поведінки без попередньої розмітки даних. Кластеризація дозволяє виявити непередбачені стилі гри або сегменти аудиторії [8]. Результати кластеризації можуть бути використані для створення «персон гравців», як це було запропоновано K. Holmgård та ін., що в подальшому може слугувати основою для більш грубої, але все ж значущої персоналізації контенту [9]. Отже, некероване навчання може доповнити кероване, допомагаючи краще зрозуміти різноманіття гравців.

Порівняння основних методів машинного навчання для завдань моделювання гравця за ключовими характеристиками представлене у табл. 2.1.

Таблиця 2.1. Порівняльний аналіз методів ML для моделювання гравця

Метод	**Тип задачі**	**Типова Точність**	**Інтерпретованість**	**Швидкість (Навч./Викон.)**	**Вимоги до даних (Обсяг/Розмітка)**
Класифікація/Регресія					
Дерева рішень (DT)	Класиф. / Регрес.	Середня	Висока	Швидко / Дуже швидко	Низький / Обов'язкова
Метод опорних векторів (SVM)	Класиф. / Регрес.	Сер. / Висока	Низька / Середня	Повільно / Швидко	Середній / Обов'язкова
Нейронні мережі (NN)	Класиф. / Регрес.	Висока / Дуже вис.	Дуже низька	Дуже повільно / Дуже швидко	Високий / Обов'язкова
Кластеризація					
k-середніх (k-means)	Кластеризація	Залежить від даних	Середня	Швидко / Швидко	Середній / Не потрібна (потрібне k)
DBSCAN	Кластеризація	Залежить від даних	Середня	Сер. / Повільно	Середній / Не потрібна (потрібні eps, min_{pts})

Враховуючи необхідність динамічного оновлення моделі гравця, обрані методи повинні підтримувати можливість інкрементного навчання або швидкого перенавчання на нових даних, що надходять під час ігрової сесії. Простота деяких моделей (наприклад, дерев рішень або наївного баєсівського класифікатора) може бути перевагою з точки зору обчислювальної ефективності в реальному часі. Тому на етапі проектування передбачається вибір конкретних алгоритмів з урахуванням балансу між точністю моделювання, інтерпретованістю результатів та обчислювальною складністю.

Вибір методу для генерації основного ігрового контенту (у рамках даної роботи фокус буде зроблено на генерації структури ігрових рівнів для 2D-платформера) є ключовим. Як показав аналіз, методи PCGML мають значні переваги над традиційними алгоритмами у здатності навчатися на прикладах, відтворювати стилі та генерувати складний, правдоподібний контент. Тому для модуля генерації контенту пропонується використовувати саме ML-підходи.

Генеративні змагальні мережі (GANs) [21] продемонстрували високу якість у генерації візуально схожих на людські рівнів для таких ігор, як Super

Mario Bros [5] чи Doom [16]. Їхня здатність генерувати різноманітні та часто несподівані результати є привабливою. Однак, як зазначалося, GANs страждають від проблем стабільності навчання та складності контролю над генерацією (наприклад, забезпечення іграбельності рівня).

Варіаційні автоенкодери (VAEs) [6] пропонують більш стабільне навчання та краще структурований латентний простір, що потенційно полегшує керування генерацією шляхом маніпуляцій у цьому просторі [4, 16]. Це може бути корисним для адаптації, оскільки різні області латентного простору можуть відповідати рівням з різними характеристиками (наприклад, складністю). Недоліком може бути дещо нижча якість (розмитість) згенерованих зразків порівняно з GAN.

Рекурентні нейронні мережі (RNN/LSTM) [25] добре підходять для генерації контенту, що має послідовну структуру, як-от рівні, представлені у вигляді послідовності сегментів або плиток [17]. Вони здатні вивчати локальні залежності та правила компонування елементів. Їхня генерація зазвичай є більш контрольованою, ніж у GAN, оскільки відбувається покроково. Це також може спростити адаптацію шляхом впливу на вибір наступного елемента послідовності.

Навчання з підкріпленням (RL) [28], як у підході PCGRL [7], дозволяє безпосередньо оптимізувати процес генерації контенту під певні цілі, пов’язані з гравцем (наприклад, підтримання певного рівня складності або залученості). Це виглядає найбільш прямим шляхом до адаптивної генерації. Однак, розробка відповідних функцій винагороди та стабільне навчання RL-агента для генерації складних рівнів є нетривіальними завданнями.

Порівняння ключових архітектур глибокого навчання для генерації ігрового контенту за основними властивостями представлене у табл. 2.2.

Таблиця 2.2. Порівняльний аналіз архітектур нейронних мереж для генерації контенту

Архітектура	**Якість генерації (Візуальна)**	**Стабільність навчання**	**Контрольованість генерації**	**Обчислювальна складність**	**Придатність для адаптації**
GAN	Висока (чіткі зображення)	Низька	Низька / Середня (латент./умова)	Висока	Середня (умова / латент.)
VAE	Сер. / Вис. (можлива розмитість)	Висока	Середня / Висока (латент. простір)	Сер. / Висока	Висока (латент. простір)
RNN/LSTM	Середня (локальна когерентність)	Середня	Висока (послідовна генерація)	Середня	Висока (керування кроком)
Трансформер	Дуже висока (глобальна когерентність)	Середня	Середня / Висока (умова / увага)	Дуже висока	Висока (умова / контекст)

Враховуючи мету створення системи, що генерує структуровані рівні та дозволяє здійснювати керовану адаптацію, комбінований підхід видається найбільш перспективним. Наприклад, можна використовувати генеративну модель (VAE або GAN) для вивчення загального стилю та базових будівельних блоків рівнів з існуючих даних, а потім використовувати механізми на основі RL або послідовної генерації (LSTM) для компонування цих блоків або налаштування параметрів генерації відповідно до моделі гравця. Такий гібридний підхід дозволить поєднати переваги різних методів: здатність генеративних моделей до вивчення стилю та здатність RL/LSTM до контрольованої та цілеспрямованої генерації. Дослідження, що комбінують різні ML-техніки для PCG, починають з’являтися і демонструють багатообіцяючі результати [4]. Саме цей напрямок буде досліджено при проектуванні алгоритмів системи.

Ключовим елементом системи є механізм, що пов’язує модель гравця з процесом генерації контенту. Вибір цього механізму визначає, як саме інформація про гравця впливатиме на згенерований контент.

Параметричне керування. Модель гравця (наприклад, оцінка навичок) використовується для прямого налаштування параметрів генератора (наприклад, щільності ворогів, довжини стрибків, кількості ресурсів). Відносно простий підхід, але він може бути недостатньо гнучким для тонкої адаптації складних структур.

Керування у латентному просторі. Якщо використовується генеративна модель типу VAE/GAN, модель гравця може використовуватися для вибору або модифікації вектора у латентному просторі, з якого генерується контент. Різні області латентного простору можуть відповідати контенту з різними властивостями [5, 16]. Потенційно більш потужний підхід, але вимагає дослідження структури латентного простору.

Умовна генерація (Conditional Generation). Модель генерації (наприклад, умовний GAN або VAE, або LSTM/трансформер з додатковими входами) безпосередньо навчається генерувати контент, що відповідає певним умовам, якими можуть бути характеристики гравця. Наприклад, мережа може отримувати на вхід бажаний рівень складності або стиль гри.

Адаптація на основі RL. Як згадувалося, RL-агент [7] може навчатися генерувати контент, що максимізує винагороду, яка залежить від моделі гравця (наприклад, винагорода за підтримання стану «потоку» або відповідність вподобанням). Найбільш прямий спосіб реалізації цілеспрямованої адаптації.

Враховуючи мету динамічної адаптації та потенціал RL для оптимізації під цілі, пов’язані з гравцем, підхід на основі навчання з підкріпленням (RL) або умовної генерації виглядає найбільш перспективним для реалізації механізму адаптації в даній роботі. RL дозволяє системі самостійно навчатися стратегіям генерації, що найкращим чином відповідають стану гравця, тоді як умовна генерація забезпечує прямий контроль над властивостями контенту на основі профілю гравця. Вибір між ними або їх комбінація буде залежати від

конкретної архітектури генератора контенту та доступності даних для навчання.

Класифікація та приклади метрик для оцінки процедурно згенерованого контенту представлена у табл. 2.3.

Таблиця 2.3. Метрики оцінки якості та різноманітності згенерованого контенту

Категорія метрик	**Призначення**	**Приклади метрик**
Функціональні	Перевірка базової працездатності контенту	Прохідність рівня (A*), відповідність правилам гри, наявність ключових елементів
Схожості зі зразком	Оцінка відповідності стилю навчальних даних	Статистичні розподіли елементів (KL-дивергенція), точність класифікатора (GAN)
Різноманітності	Вимірювання варіативності згенерованих зразків	Відстань між зразками (в латентному / вихідному просторі), кількість унікальних патернів
Орієнтовані на гравця	Оцінка контенту з точки зору гравця	Оцінка складності, прогнозована залученість / фан [10], відповідність профілю гравця
Суб'єктивні (людські)	Загальна оцінка людиною (гравцем / експертом)	Рейтинги (шкала Лайкерта), опитування (GEQ [1]), аналіз відгуків, A / B тестування

Отже, для вирішення поставленої задачі пропонується використовувати наступний комплекс методів машинного навчання:

- **Для моделювання гравця.** Комбінація методів керованого (класифікація/регресія з використанням дерев рішень, SVM або нейронних мереж) та некерованого навчання (кластеризація) для аналізу даних телеметрії та побудови динамічної моделі гравця (стиль, навички).
- **Для генерації контенту (рівнів).** Гібридний підхід, що потенційно поєднує глибокі генеративні моделі (VAE/GAN) для вивчення стилю та базових елементів з методами послідовної генерації (LSTM) або RL для контрольованого та структурованого компонування рівня.
- **Для механізму адаптації.** Підходи на основі навчання з підкріпленням (RL) або умовної генерації (Conditional Generation) для

зв’язку моделі гравця з процесом генерації контенту, забезпечуючи цілеспрямовану адаптацію.

Структурна схема розробленої системи, що показує основні функціональні модулі та потоки даних між ними та ігровим рушієм (Unity) зображена на рис. 2.1.

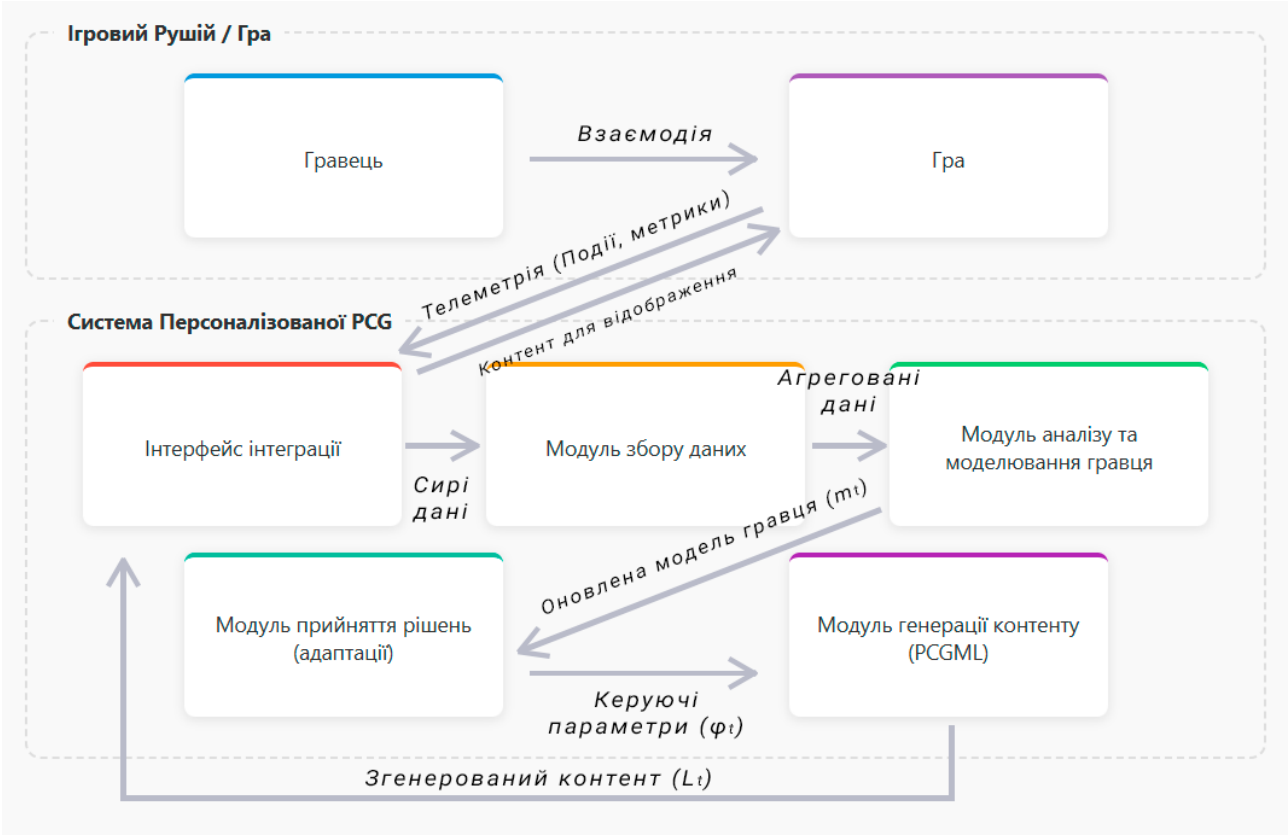

Рисунок 2.1. Структурна схема архітектури системи персоналізованої PCG

Такий вибір методів дозволяє повною мірою використати переваги сучасного машинного навчання для вирішення завдань аналізу даних, генерації складного контенту та оптимізації під потреби користувача.

2.2. Формалізація задачі динамічної та персоналізованої генерації

У рамках даної роботи під ігровим контентом L розуміється елемент ігрового світу, що процедурно генерується. Для конкретності, розглянемо випадок генерації двовимірного ігрового рівня (наприклад, для гри жанру платформер або dungeon crawler). Такий рівень може бути формально

представлений у вигляді матриці (або тензора) $L \in \mathcal{L}$, де $\mathcal{L}$ – простір усіх можливих рівнів. Наприклад, L може бути матрицею розміром $W \times X$, де кожен елемент l_{ij} належить до скінченної множини типів ігрових плиток або об'єктів $T = \{0,1, \dots, T_{max}\}$ (наприклад, 0 – порожній простір, 1 – земля, 2 – перешкода, 3 – бонус тощо):

$$L = [l_{ij}]_{W \times X},\ l_{ij} \in T, \tag{2.1}$$

Процес процедурної генерації контенту можна розглядати як стохастичний процес або детерміновану функцію, що відображає набір вхідних параметрів у вихідний контент L. У контексті PCGML, цей процес реалізується за допомогою моделі машинного навчання G, навченої на певному наборі даних.

Якщо використовується генеративна модель, така як VAE або GAN, то генерація зазвичай відбувається шляхом відображення вектора z з латентного простору $Z \subseteq R^d$ у простір контенту $\mathcal{L}$. Генератор G є функцією (зазвичай нейронною мережею):

$$G: Z \to \mathcal{L}, \tag{2.2}$$

Згенерований рівень L отримується як $L = G(z)$, де вектор z зазвичай семплується з апріорного розподілу $P(Z)$ (наприклад, стандартного нормального розподілу) [6, 21].

Якщо використовується послідовна модель, така як RNN/LSTM або агент RL, процес генерації розглядається як послідовність кроків $t = 1, \dots, N$. На кожному кроці t на основі поточного стану s_t (що представляє частково згенерований рівень або іншу релевантну інформацію) обирається дія a_t (наприклад, вибір наступної плитки або сегмента рівня). Дія обирається відповідно до політики π, параметризованої вектором θ:

$$a_t \sim \pi(a_t | s_t; \theta), \tag{2.3}$$

Послідовність дій $\{a_1, \dots, a_N\}$ визначає кінцевий контент L. Наприклад, s_t може бути прихованим станом LSTM [17], а a_t – вибором наступного символу

в рядковому представленні рівня. В RL s_t є станом середовища генерації, а π – політикою агента-генератора [7].

Ключовим аспектом для персоналізації та адаптації є можливість керування процесом генерації. Тому модель генератора G (або політика π) повинна залежати не лише від внутрішніх параметрів θ та випадкового шуму/початкового стану, але й від зовнішнього керуючого сигналу або параметра $\phi \in \Phi$, що відображає бажані характеристики контенту. Формально, процес генерації можна представити як $L = G(z; \phi, \theta)$ для генеративних моделей, або $a_t \sim \pi(a_t|s_t; \phi, \theta)$ для послідовних моделей.

Параметр ϕ може представляти, наприклад, бажаний рівень складності, стилістичні особливості, розмір рівня тощо. Саме через цей параметр ϕ буде здійснюватися вплив моделі гравця на генерацію контенту.

Адаптація передбачає налаштування процесу генерації контенту на основі інформації про гравця. Для цього необхідні два ключові компоненти: модель гравця та механізм адаптації, що пов'язує модель гравця з параметрами генерації.

Модель гравця M_p – це формальне представлення релевантних характеристик гравця у певний момент часу t. Модель може бути представлена вектором $m_t \in M$, де M – простір станів моделі гравця. Цей вектор може містити різні компоненти, що відображають:

- **Навички (Skill).** Оцінка рівня майстерності гравця, наприклад, $skill_t \in [0,1]$.
- **Стиль гри (Playstyle).** Категоріальна змінна або розподіл ймовірностей за стилями, наприклад, $style_t \in \{explorer, fighter, collector, \dots\}$.
- **Вподобання (Preferences).** Оцінка того, яким типам контенту чи викликів гравець надає перевагу.
- **Поточний стан (State).** Наприклад, рівень втоми, фрустрації чи залученості, якщо їх вдається оцінити.

Модель гравця є динамічною, тобто вона оновлюється з часом на основі спостережуваної поведінки гравця. Нехай D_t – це набір даних (телеметрії), зібраних про взаємодію гравця з грою за певний проміжок часу до моменту *t*, $D_t = \{e_1, e_2, \dots, e_k\}$, де e_i – події (натискання клавіш, переміщення, отримання шкоди, збір предметів тощо). Процес оновлення моделі гравця можна представити функцією *U*:

$$m_{t+1} = U(m_t, D_{t+1}), \tag{2.4}$$

Функція U може бути реалізована за допомогою різних методів ML: класифікатори, регресори, методи кластеризації, що обробляють нові дані D_{t+1} та оновлюють вектор характеристик m_t [8, 9, 10].

Механізм адаптації визначає, як поточна модель гравця m_t впливає на процес генерації контенту. Формально, це функція *A*, яка відображає стан моделі гравця m_t у керуючий параметр ϕ_t для генератора:

$$A: M \to \Phi, \tag{2.5}$$
$$\phi_t = A(m_t)$$

Цей параметр ϕ_t потім використовується генератором *G* (або політикою π) для створення наступного фрагменту контенту L_t: $L_t = G(z_i; \phi_t, \theta)$ або $a_k \sim \pi(a_k | s_k; \phi_t, \theta)$, для $\mathrm{k} = 1 \dots N$.

Функція адаптації *A* реалізує логіку персоналізації. Наприклад:

- Якщо m_t вказує на низький рівень навичок, *A* може встановити ϕ_t, що відповідає низькій складності (менше ворогів, простіші перешкоди).
- Якщо m_t вказує на стиль «дослідник», *A* може встановити ϕ_t, що сприяє генерації більших рівнів з секретними областями.
- Якщо використовується VAE, *A* може відображати m_t у певну область латентного простору *Z*, з якої потім семплується z_t.

Метою адаптації є оптимізація певного критерію, пов’язаного з досвідом гравця. Цей критерій можна формалізувати як функцію $E(L, m)$, що оцінює якість контенту L для гравця з моделлю m. Наприклад, E може представляти залученість, задоволення, час гри або ймовірність досягнення стану «потоку».

Тоді задача адаптивної системи полягає у виборі такої послідовності керуючих параметрів $\phi_t = A(m_t)$, щоб максимізувати очікуване значення E протягом ігрової сесії:

$$max_{\{L_t\}} E[\sum_t \gamma^t E(L_t, m_t)], \quad (2.6)$$

де L_t генерується за допомогою $G(\cdot; A(m_t), \theta)$, а m_t оновлюється через U. Це формулювання природно вкладається в рамки навчання з підкріпленням, де агент (система адаптації) обирає дії (параметри ϕ_t) для максимізації довгострокової винагороди E [1, 28].

Таким чином, формалізація задачі включає визначення математичних представлень для контенту L, моделі генерації $G(\cdot; \phi, \theta)$, моделі гравця m_t, функції оновлення моделі гравця U та функції адаптації A.

2.3. Проектування архітектури системи

Для реалізації функціональності динамічної та персоналізованої процедурної генерації ігрового контенту на основі машинного навчання необхідна добре продумана архітектура системи. З огляду на комплексність завдання, що включає збір даних, аналіз поведінки, прийняття рішень та генерацію контенту, доцільною є побудова модульної архітектури. Такий підхід забезпечує гнучкість, масштабованість, полегшує розробку, тестування та потенційну модифікацію окремих частин системи без суттєвого впливу на інші. Архітектура повинна забезпечувати ефективну взаємодію між компонентами для реалізації циклу адаптації: спостереження за гравцем → аналіз → прийняття рішення → генерація/адаптація контенту.

Запропонована система складається з декількох ключових функціональних компонентів (модулів), кожен з яких відповідає за виконання специфічних завдань у рамках загального процесу персоналізованої генерації. Основними компонентами системи є:

1. **Модуль збору даних (Data Collector).** Відповідає за збір необроблених даних (телеметрії) про взаємодію гравця з ігровим середовищем. Ці дані надходять з ігрового рушія через

спеціалізований інтерфейс. Модуль фіксує різноманітні події, такі як дії гравця (рух, стрибки, атаки, використання предметів), події ігрового світу (отримання шкоди, досягнення цілей, зіткнення), а також показники продуктивності (час проходження сегментів, кількість спроб, зібрані ресурси). Зібрані дані можуть тимчасово буферизуватися та передаватися для подальшої обробки у Модуль аналізу та моделювання гравця. Важливим аспектом є визначення гранулярності та частоти збору даних, щоб забезпечити достатню інформативність без надмірного навантаження на систему.

2. **Модуль аналізу та моделювання гравця (Player Modeler).** Є ядром підсистеми розуміння гравця. Цей модуль отримує дані від Модуля збору даних та застосовує методи машинного навчання для побудови та оновлення моделі гравця m_t. Він реалізує функцію оновлення $U(m_t, D_{t+1})$. Наприклад, модуль може використовувати класифікатори для визначення поточного стилю гри або рівня навичок, регресійні моделі для прогнозування продуктивності, або алгоритми кластеризації для виявлення патернів поведінки. Модель гравця m_t, що зберігається та оновлюється цим модулем, містить формалізовані характеристики гравця (наприклад, вектор ознак), які потім передаються до Модуля прийняття рішень щодо адаптації. Важливою вимогою до цього модуля є здатність до динамічного оновлення моделі в реальному часі або з мінімальною затримкою.
3. **Модуль прийняття рішень щодо адаптації (Adaptation Logic).** Відповідає за інтерпретацію поточної моделі гравця m_t та визначення того, як саме слід адаптувати генерацію контенту. Цей модуль реалізує функцію адаптації $A(m_t) = \phi_t$. Він перетворює абстрактні характеристики гравця (навички, стиль) у конкретні керуючі параметри ϕ_t для Модуля генерації контенту. Ці параметри можуть визначати бажаний рівень складності, тип викликів, щільність певних елементів, структурні характеристики рівня тощо. Логіка адаптації

може бути реалізована за допомогою набору правил, нечіткої логіки, або ж, що є більш перспективним для складних взаємозв'язків, за допомогою навченої моделі (наприклад, політики в рамках RL, яка обирає ϕ_t для максимізації очікуваної залученості гравця).

4. **Модуль генерації контенту (Content Generator).** Основний компонент, що відповідає безпосередньо за створення ігрового контенту L_t (наприклад, ігрових рівнів). Він використовує обрані моделі машинного навчання G (наприклад, VAE, LSTM, GAN, або їх комбінацію) та отримує від Модуля адаптації керуючі параметри ϕ_t. На основі цих параметрів та, можливо, випадкового вектора z_t, модуль генерує новий екземпляр контенту: $L_t = G(z_t; \phi_t, \theta)$. Згенерований контент повинен відповідати не лише параметрам адаптації, але й базовим вимогам функціональності та стилістики гри. Після генерації контент передається до ігрового рушія через Інтерфейс інтеграції.
5. **Інтерфейс інтеграції з грою (Game Interface).** Забезпечує двосторонній зв'язок між системою персоналізованої генерації та ігровим рушієм. Він передає дані телеметрії від гри до Модуля збору даних та доставляє згенерований контент L_t від Модуля генерації контенту назад до гри для його відображення та використання у геймплеї. Цей компонент має бути розроблений з урахуванням специфіки цільового ігрового рушія (наприклад, Unity, Unreal Engine) та забезпечувати ефективну та надійну передачу даних.

Такий модульний поділ дозволяє чітко розмежувати відповідальності та сприяє незалежній розробці та тестуванню кожного компонента системи.

Ефективне функціонування системи залежить від чітко визначених інформаційних потоків та протоколів взаємодії між її компонентами. Основний цикл роботи системи, що реалізує динамічну адаптацію, виглядає наступним чином:

1. **Гравець взаємодіє з грою.** Ігровий рушій генерує події та дані телеметрії на основі дій гравця та стану гри.

2. **Збір даних.** Інтерфейс інтеграції передає потокові або періодичні дані телеметрії D_t до Модуля збору даних.
3. **Агрегація та передача даних.** Модуль збору даних може виконувати первинну обробку (фільтрацію, агрегацію) та передає підготовлені дані до Модуля аналізу та моделювання гравця.
4. **Оновлення моделі гравця.** Модуль аналізу та моделювання гравця обробляє отримані дані та оновлює внутрішнє представлення моделі гравця m_t, використовуючи функцію U.
5. **Передача моделі для адаптації.** Актуальна модель гравця m_t передається до Модуля прийняття рішень щодо адаптації.
6. **Визначення параметрів генерації.** Модуль адаптації на основі m_t визначає набір керуючих параметрів $\phi_t = A(m_t)$, що специфікують бажані характеристики наступного фрагменту контенту.
7. **Запит на генерацію.** Параметри ϕ_t передаються до Модуля генерації контенту.
8. **Генерація контенту.** Модуль генерації контенту використовує параметри ϕ_t та свою внутрішню модель G для створення нового контенту L_t. Цей процес може включати перевірку на функціональність (наприклад, прохідність).
9. **Передача контенту в гру.** Згенерований контент L_t передається через Інтерфейс інтеграції до ігрового рушія.
10. **Інтеграція контенту.** Ігровий рушій інтегрує новий контент в ігровий світ (наприклад, завантажує наступний сегмент рівня). Гравець починає взаємодіяти з адаптованим контентом, і цикл замикається.

Важливо зазначити, що цей цикл може бути реалізований по-різному. Наприклад, оновлення моделі гравця та прийняття рішень щодо адаптації можуть відбуватися асинхронно до основного ігрового циклу, щоб не впливати на продуктивність гри. Генерація контенту також може відбуватися заздалегідь (наприклад, генерація наступного сегменту рівня, поки гравець проходить поточний) або «на льоту» для миттєвої реакції на дії гравця, залежно від вимог

до динамічності та обчислювальної складності генератора. Взаємодія між модулями повинна бути реалізована через чітко визначені API (Application Programming Interfaces), що забезпечить слабку зв'язність компонентів.

Діаграма послідовності, що ілюструє кроки взаємодії модулів системи під час одного циклу динамічної адаптації контенту представлена на рис. 2.2.

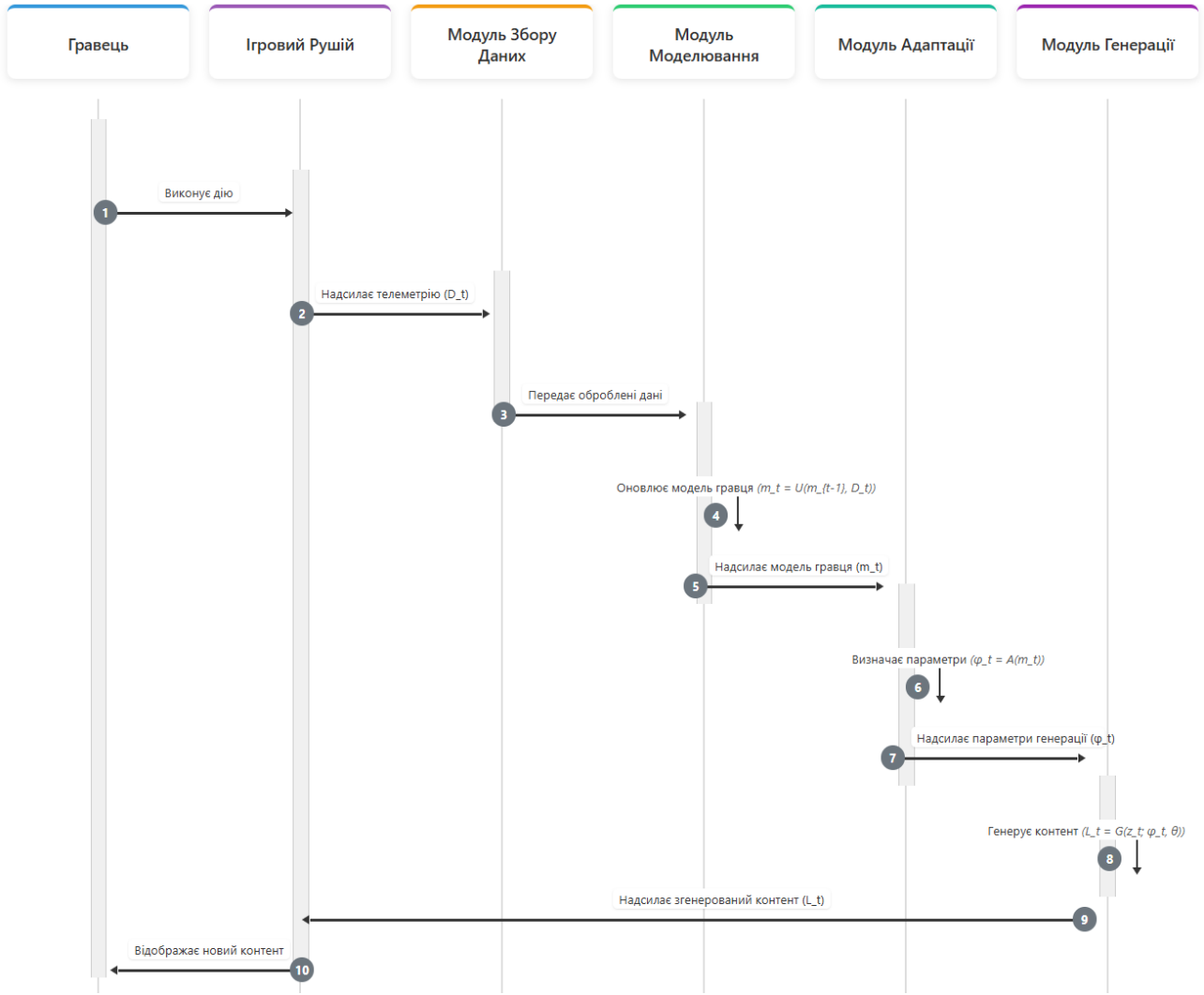

Рисунок 2.2. Діаграма інформаційних потоків в адаптивному циклі

Запропонована архітектура та описані інформаційні потоки створюють основу для розробки системи, здатної реалізувати складний процес динамічної та персоналізованої генерації ігрового контенту.

2.4. Проектування основних алгоритмів системи

Цей алгоритм реалізує функціональність Модуля аналізу та моделювання гравця і відповідає за перетворення потоку необроблених даних телеметрії D_t на значущу модель гравця m_t. Процес включає кілька послідовних кроків.

По-перше, відбувається агрегація даних. Необроблені події телеметрії, що надходять з високою частотою, агрегуються за певний період часу (наприклад, кожні 30 секунд) або за логічний сегмент гри (наприклад, після проходження кімнати або завершення квесту). Це дозволяє отримати зведені показники, що відображають поведінку гравця за значущий проміжок часу.

По-друге, виконується виділення ознак (Feature Engineering). З агрегованих даних розраховується набір числових ознак, що характеризують різні аспекти поведінки гравця. Вибір ознак залежить від жанру гри та цілей адаптації, але може включати такі показники, як відсоток успішного завершення завдань, середня швидкість проходження, частота використання певних ігрових механік (наприклад, атаки ближнього бою, магії, блокування), кількість зроблених помилок (отримання шкоди, падіння у прірву), відсоток дослідженої території рівня, кількість зібраних ресурсів чи бонусів тощо. Ретельний вибір та розрахунок інформативних ознак є критично важливим для якості подальшого моделювання [8].

По-третє, здійснюється застосування моделей машинного навчання для оновлення моделі гравця m_t. На основі розрахованих ознак працюють попередньо навчені ML-моделі. Наприклад, для визначення стилю гри ($style_t$) може використовуватися класифікатор, такий як Наївний Баєсівський класифікатор або Дерево Рішень, навчений розрізняти типові патерни поведінки, характерні для різних стилів (наприклад, висока частота атак для «агресора», велика досліджена площа для «дослідника»). Для оцінки рівня навичок ($skill_t$) може застосовуватися регресійна модель (наприклад, лінійна регресія або нейронна мережа), що прогнозує майстерність гравця на основі його ефективності (швидкості, точності, кількості помилок), як це було продемонстровано M. T. Pedersen та ін. для гри Super Mario Bros [10]. Також

можуть використовуватися методи кластеризації, наприклад k-середніх, для виявлення невідомих або непередбачених груп гравців зі схожою поведінкою [9].

Нарешті, відбувається оновлення та збереження моделі гравця. Результати роботи ML-моделей (нові оцінки навичок, визначений стиль гри тощо) інтегруються у вектор m_t, оновлюючи поточне представлення гравця. Це оновлене представлення m_t зберігається для використання Модулем прийняття рішень щодо адаптації та слугує базою для наступного циклу оновлення. Важливою характеристикою цього алгоритму є його здатність працювати динамічно, регулярно оновлюючи модель гравця впродовж ігрової сесії.

Блок-схема алгоритму роботи Модуля аналізу та моделювання гравця зображена на рис. 2.3.

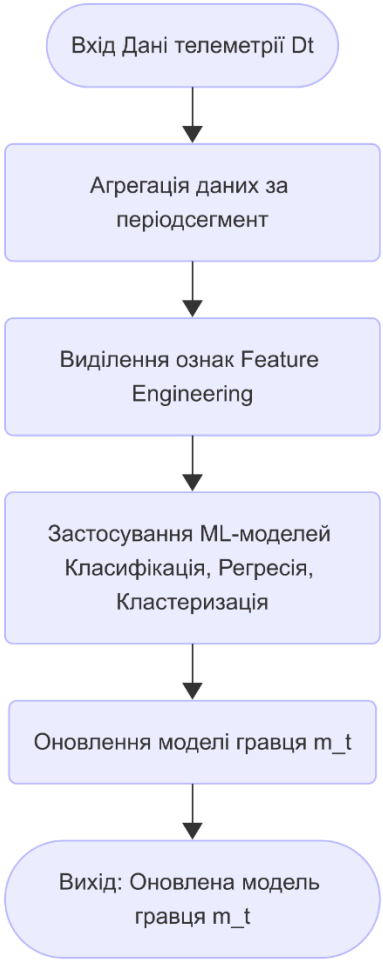

Рисунок 2.3. Блок-схема алгоритму моніторингу та аналізу патернів поведінки гравця

Цей алгоритм лежить в основі Модуля прийняття рішень щодо адаптації і реалізує функцію $A(m_t) = \phi_t$. Його завдання – перетворити поточну модель гравця m_t на конкретний набір керуючих параметрів ϕ_t, які будуть використані генератором контенту.

Алгоритм починається з отримання актуальної моделі гравця m_t від Модуля аналізу та моделювання. З цієї моделі вилучаються ключові характеристики, релевантні для адаптації, наприклад, поточна оцінка навичок $skill_t$ та визначений стиль гри $style_t$.

Наступним кроком є застосування логіки адаптації для визначення параметрів ϕ_t. Ця логіка може бути реалізована різними способами. Найпростіший варіант – використання набору правил, визначених експертом (дизайнером гри). Наприклад, правило може виглядати так: «Якщо $skill_t < 0.3$, встановити $\phi_{difficulty} = low$ (низька щільність ворогів, більше підказок)». Інше правило: «Якщо $style_t = fighter$, встановити $\phi_{challenge\_type} = combat$ (генерувати більше бойових зіткнень)». Іншим підходом є використання математичних функцій для плавного відображення характеристик гравця у параметри контенту. Наприклад, бажана щільність перешкод $\phi_{obstacle\_density}$ може бути зростаючою функцією від $skill_t$:

$$\phi_{obstacle\_density} = c_1 + c_2 \cdot skill_t, \tag{2.7}$$

Більш складним та гнучким підходом є використання навчання з підкріпленням (RL). У цьому випадку Модуль адаптації діє як RL-агент, що перебуває у стані, визначеному моделлю гравця m_t. Агент обирає дію, яка полягає у виборі параметрів ϕ_t, з метою максимізації довгострокової винагороди, що відображає залученість або задоволеність гравця [1, 7]. Навчання такої політики вибору ϕ_t може відбуватися офлайн на симульованих даних або онлайн під час взаємодії з реальними гравцями.

На завершення, розрахований вектор керуючих параметрів ϕ_t передається до Модуля генерації контенту для використання у наступному

циклі генерації. Цей алгоритм забезпечує зв'язок між розумінням гравця та процесом створення контенту, реалізуючи адаптивний цикл.

Цей алгоритм є важливим інструментом для оцінки ефективності роботи системи персоналізації. Він дозволяє кількісно визначити, наскільки згенерований контент L_t відповідає цілям адаптації, що були поставлені на основі моделі гравця m_t та виражені через параметри ϕ_t

Алгоритм отримує на вхід згенерований контент L_t, цільові параметри ϕ_t, за якими він генерувався, та модель гравця m_t, для якої він призначений.

Першим кроком є аналіз характеристик згенерованого контенту. З об'єкта L_t (наприклад, матриці рівня) вилучаються фактичні характеристики, що відповідають цільовим параметрам. Наприклад, якщо ϕ_t включав бажану щільність ворогів, то алгоритм підраховує реальну щільність ворогів у згенерованому рівні L_t. Якщо ϕ_t визначав бажаний розмір рівня, вимірюється фактичний розмір L_t. Таким чином, отримується вектор фактичних характеристик контенту $C(L_t)$.

Другим кроком є порівняння фактичних та цільових характеристик. Розраховується метрика, що відображає ступінь розбіжності між вектором цільових параметрів ϕ_t та вектором фактичних характеристик $C(L_t)$. Це може бути, наприклад, евклідова відстань, косинусна подібність або інша міра близькості у просторі характеристик. Чим менша розбіжність, тим краще генератор виконав завдання адаптації.

Третім, опціональним, кроком може бути використання моделі прогнозування досвіду гравця. Якщо існує модель $E(L, m)$, що прогнозує очікуваний досвід (наприклад, залученість, задоволення, складність сприйняття) гравця з моделлю m при взаємодії з контентом L, то її можна застосувати до згенерованого L_t та поточної моделі гравця m_t. Результат $E(L_t, m_t)$ дасть прогнозовану оцінку якості контенту саме для цього гравця, що є більш глибокою метрикою відповідності, ніж просто порівняння технічних параметрів. Побудова такої моделі E сама по собі є складною задачею, що може вимагати збору даних від реальних гравців [10].

Блок-схема алгоритму роботи Модуля прийняття рішень щодо адаптації представлена на рис. 2.4.

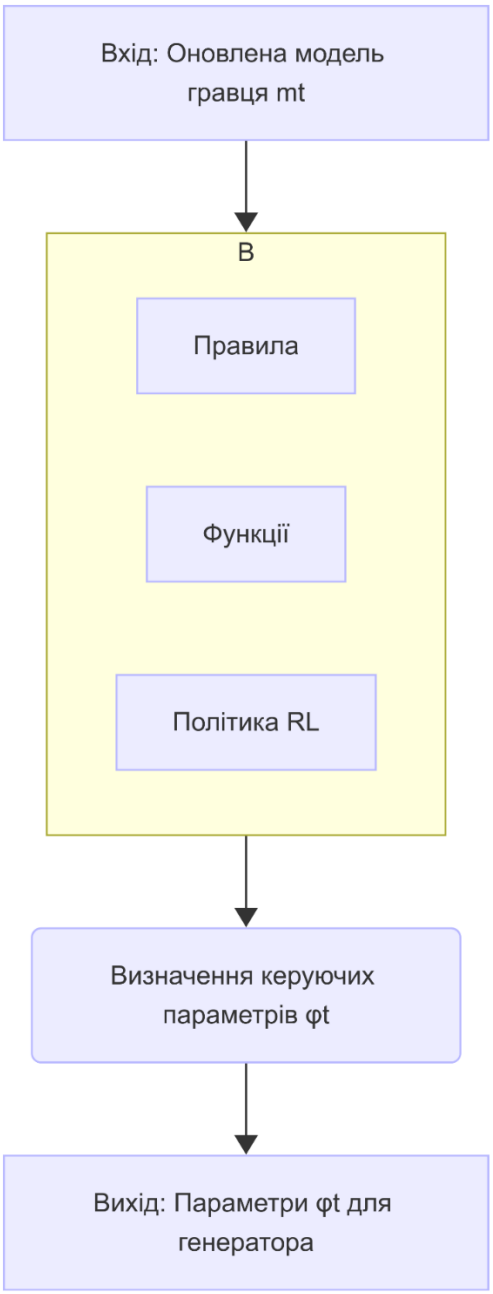

Рисунок 2.4. Блок-схема алгоритму динамічної адаптації контенту

Результатом роботи алгоритму є кількісна оцінка відповідності згенерованого контенту профілю гравця. Ця оцінка може використовуватися під час експериментальних досліджень для порівняння різних підходів до адаптації, для моніторингу роботи системи в реальному часі, а також може слугувати компонентом функції винагороди для RL-агента у Модулі адаптації.

Представлені алгоритми деталізують ключові процеси, що відбуваються всередині розроблюваної системи. Їх проектування базується на методах машинного навчання та принципах адаптивних систем, обґрунтованих раніше. Наступним кроком буде розгляд питань, пов’язаних з інтеграцією цих алгоритмів та всієї системи з ігровими рушіями.

Ілюстративний графік, що показує можливу залежність параметра генерації (наприклад, щільності перешкод) від оціненого рівня навичок гравця представлений на рис. 2.5.

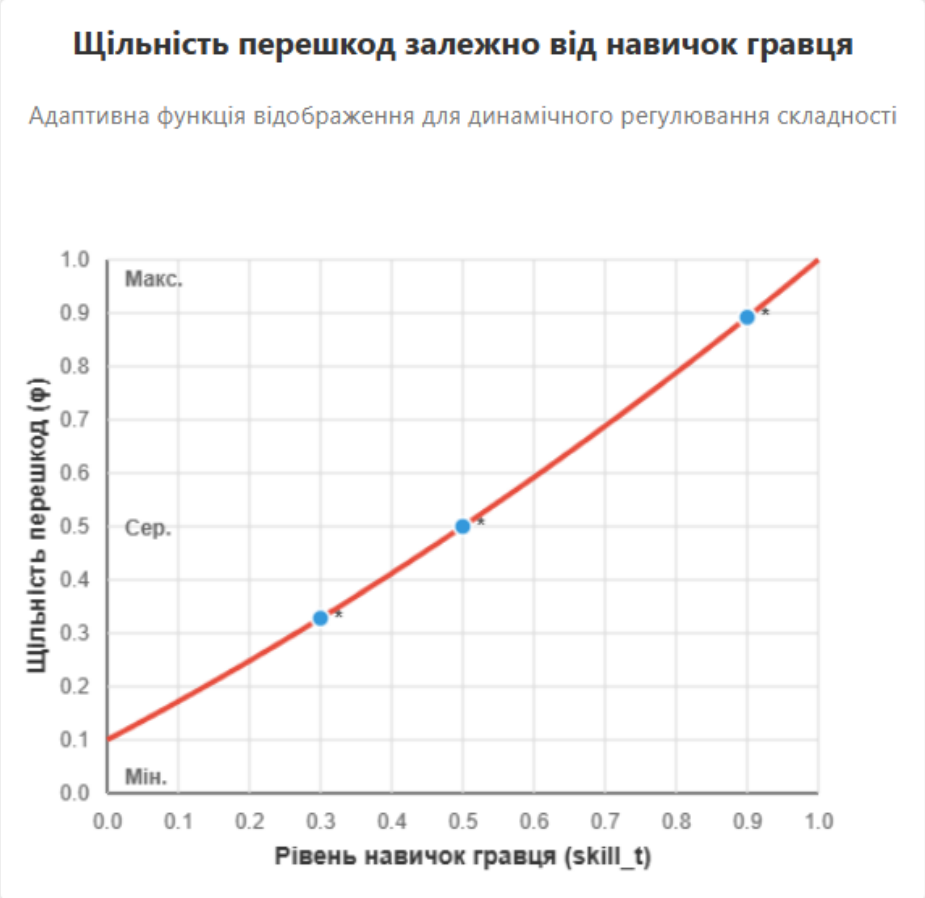

Рисунок 2.5. Приклад функції відображення характеристики гравця у параметр контенту

Приклади параметрів контенту та їх відображення представлені у табл. 2.4.

Таблиця 2.4. Параметри ігрового контенту та функції відображення

Параметр контенту	**Характеристика гравця**	**Тип функції**	**Обґрунтування**
Щільність перешкод	Рівень навичок	Квадратична	Плавний ріст складності для збалансованого виклику
Швидкість рухомих платформ	Час реакції гравця	Лінійна	Пряма залежність між швидкістю реакції та швидкістю елементів
Кількість ресурсів	Ефективність використання ресурсів	Логарифмічна	Менше ресурсів для більш ефективних гравців
Ширина стрибків між платформами	Точність рухів	Сигмоїдна	Різка зміна складності після досягнення певного порогу навичок
Розташування бонусів	Стиль дослідження	Кускова	Різні патерни розміщення залежно від категорії гравця

Ефективна інтеграція системи динамічної та персоналізованої процедурної генерації контенту з ігровим рушієм є ключовою умовою її практичного застосування. Ігровий рушій (Unity, Unreal Engine, Godot) виступає як середовище виконання гри та основне джерело даних про поведінку гравця, а також як кінцевий споживач згенерованого контенту. Для забезпечення чіткої та надійної взаємодії між системою PCG та рушієм необхідно спроектувати спеціалізовані програмні інтерфейси (API), що формалізують обмін даними. Проектування цих інтерфейсів має враховувати вимоги до продуктивності, гнучкості та простоти інтеграції у типові процеси розробки ігор.

Основна мета проектування інтерфейсів полягає у створенні чіткого контракту між системою PCG та ігровим рушієм, що дозволяє їм взаємодіяти без глибокого знання про внутрішню реалізацію один одного. Це сприяє модульній архітектурі і полегшує незалежну розробку, тестування та оновлення як самої системи PCG, так і гри. Необхідно спроектувати два основні типи інтерфейсів: для передачі даних телеметрії від гри до системи PCG та для передачі згенерованого контенту від системи PCG до гри.

Блок-схема алгоритму оцінки того, наскільки згенерований контент відповідає цільовим параметрам, визначеним на основі моделі гравця представлений на рис. 2.6.

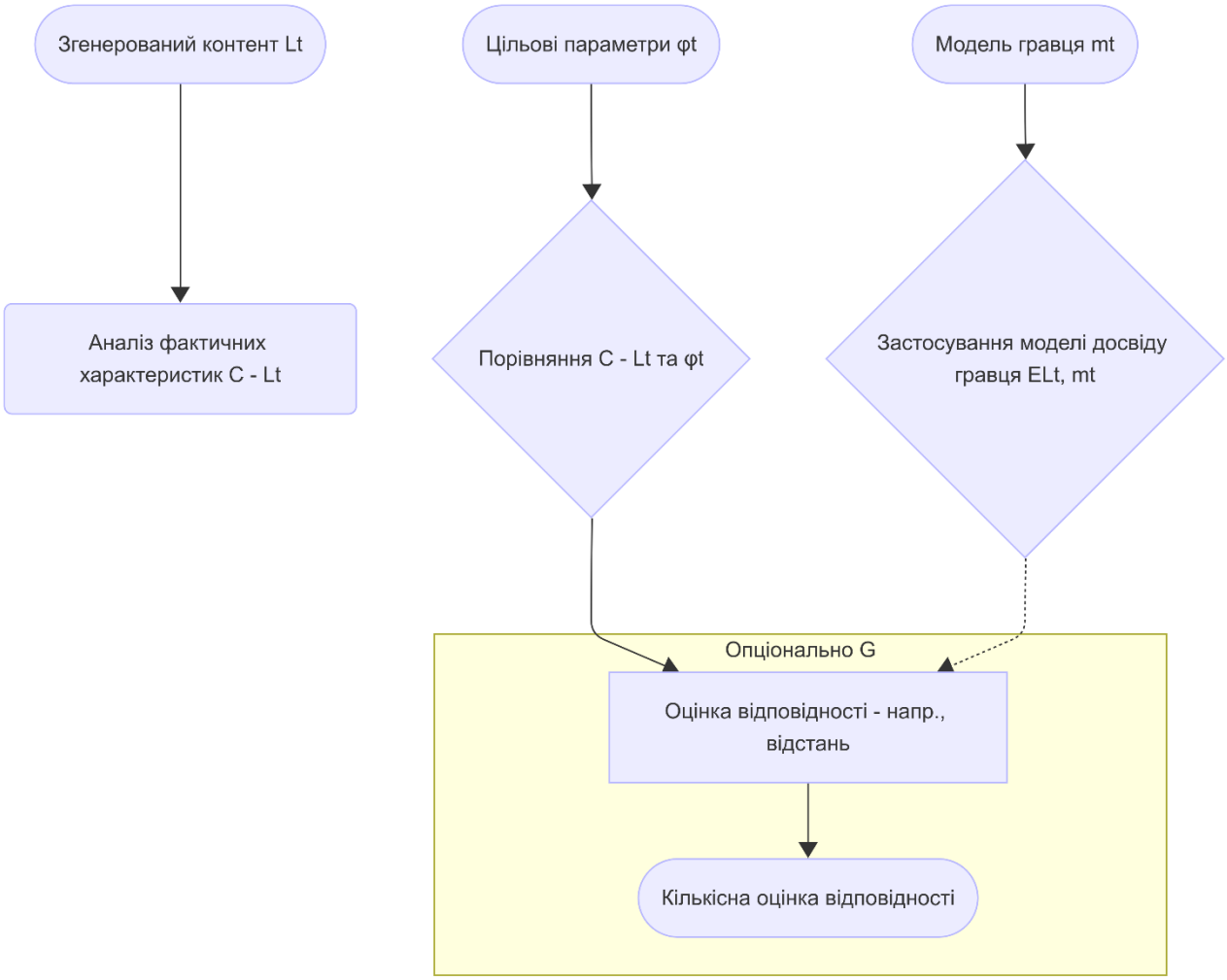

Рисунок 2.6. Блок-схема алгоритму оцінки відповідності контенту профілю гравця

Інтерфейс збору телеметрії (Game-to-PCG Interface) – цей інтерфейс відповідає за передачу даних про поведінку гравця та стан ігрового світу з ігрового рушія до Модуля збору даних системи PCG.

Визначення подій та даних. Необхідно чітко визначити перелік подій (Events) та метрик (Metrics), які мають фіксуватися. Це можуть бути низькорівневі події (натискання клавіш, рух миші/джойстика, позиція гравця), події взаємодії (атака, використання предмету, діалог), події стану (отримання шкоди, смерть, перехід на новий рівень), а також зведені метрики (час проходження, кількість спроб, зібрані ресурси). Кожна подія повинна мати унікальний ідентифікатор, часову мітку та набір релевантних параметрів (наприклад, для події «отримання шкоди» параметрами можуть бути джерело шкоди, її кількість, поточне здоров'я гравця).

Формат даних. Дані телеметрії повинні передаватися у стандартизованому форматі. Популярним вибором є JSON (JavaScript Object Notation) завдяки його гнучкості та читабельності, що зручно для налагодження. Однак для високонавантажених систем, де обсяг даних великий, можуть бути більш ефективними бінарні формати, такі як Protocol Buffers або MessagePack, що забезпечують менший розмір повідомлень та швидше кодування/декодування. Обраний формат повинен підтримувати різні типи даних (числа, рядки, булеві значення, масиви, об'єкти).

Механізм передачі. Дані можуть передаватися синхронно або асинхронно. Асинхронна передача є переважною, щоб не блокувати основний ігровий цикл (game loop). Це може бути реалізовано за допомогою черг повідомлень або викликів API, що не очікують негайної відповіді.

Спосіб інтеграції. На стороні ігрового рушія необхідно реалізувати скрипти або компоненти, що відстежують відповідні події та викликають функції API інтерфейсу для їх відправки. Це може бути реалізовано як вбудований компонент (якщо система PCG інтегрується як бібліотека/плагін) або як клієнт, що взаємодіє із зовнішнім сервісом PCG (якщо використовується архітектура мікросервісів).

Інтерфейс доставки контенту (PCG-to-Game Interface) – цей інтерфейс відповідає за передачу згенерованого контенту L_t (наприклад, опису рівня) від Модуля генерації контенту до ігрового рушія для його інстанціювання та відображення. Ключові аспекти проектування:

- **Формат представлення контенту.** Необхідно визначити структуру даних для опису згенерованого контенту. Для 2D рівня це може бути двовимірна матриця ідентифікаторів плиток, список об’єктів із зазначенням їхнього типу, позиції та властивостей, або більш складне представлення у вигляді графа. Обраний формат повинен бути достатньо детальним, щоб рушій міг однозначно відтворити згенерований контент, але водночас компактним та ефективним для передачі. Знову ж таки, JSON може використовуватися для гнучкості, а бінарні формати – для продуктивності. Важливо передбачити можливість розширення формату для підтримки нових типів контенту у майбутньому.
- **Механізм передачі.** Передача контенту може бути ініційована як системою PCG (push-модель), так і ігровим рушієм (pull-модель, коли рушій запитує новий контент, коли він потрібен). Передача також має бути переважно асинхронною, щоб генерація складного контенту не зупиняла гру. Рушій повинен мати механізм для обробки отриманого контенту та його плавної інтеграції (наприклад, завантаження нового сегмента рівня у фоновому режимі).
- **Інтерпретація контенту.** На стороні ігрового рушія має бути реалізований компонент (наприклад, «Level Builder» або «Content Instantiator»), який отримує дані про контент через інтерфейс, розпаковує їх та створює відповідні ігрові об’єкти (GameObjects в Unity, Actors в Unreal Engine) у сцені. Цей компонент повинен знати, як інтерпретувати ідентифікатори плиток чи об’єктів та які ресурси (префаби, ассети) їм відповідають.

Загальні принципи проектування інтерфейсів:

- **Слабка зв'язність (Loose Coupling).** Інтерфейси повинні мінімізувати залежності між системою PCG та грою.
- **Чіткість та документація.** API має бути добре документованим, з чітким описом функцій, параметрів та форматів даних.
- **Обробка помилок.** Інтерфейси повинні передбачати механізми обробки помилок (наприклад, невалідні дані, помилки передачі, недоступність сервісу) та повідомлення про них.
- **Версіонування.** Для забезпечення зворотної сумісності при подальшому розвитку системи та гри доцільно впровадити версіонування API.
- **Продуктивність.** Вибір форматів даних та механізмів передачі має враховувати вимоги до продуктивності, особливо для систем, що працюють у реальному часі.

Залежно від архітектурного рішення (чи є система PCG вбудованою бібліотекою чи окремим сервісом), можуть використовуватися різні технології:

- **Прямі виклики функцій/методів.** Якщо система PCG реалізована як бібліотека/плагін на тій же мові, що й скрипти рушія (наприклад, C# для Unity, C++ для Unreal Engine). Це найшвидший варіант.
- **Мережеві протоколи (REST API, gRPC, WebSockets).** Якщо система PCG працює як окремий процес або мікросервіс. REST є простим та поширеним, gRPC ефективніший для обміну структурованими даними, WebSockets підходять для двонаправленої комунікації в реальному часі.

Схема, що ілюструє взаємодію між ігровим рушієм та системою PCG через визначені програмні інтерфейси (API) для передачі телеметрії та контенту представлена на рис. 2.7.

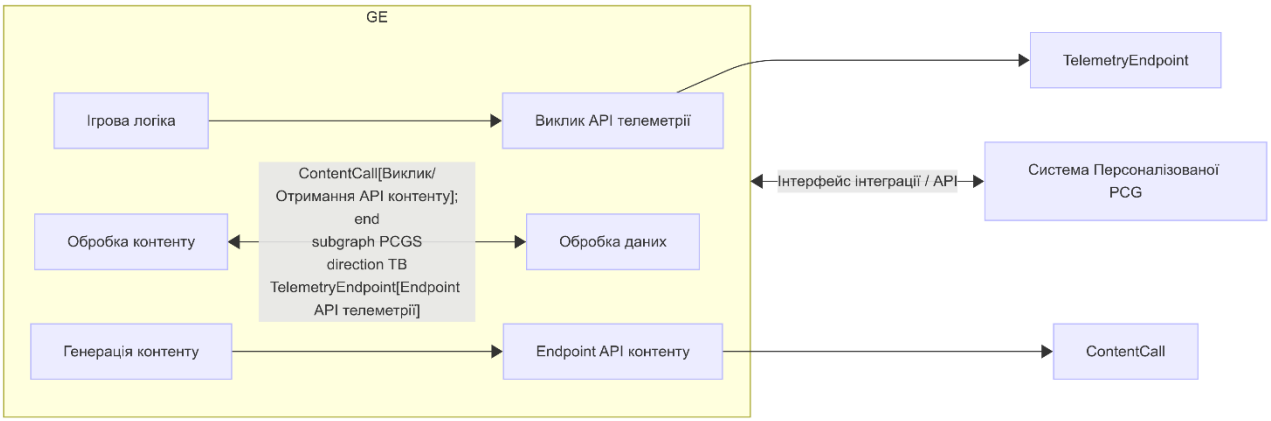

Рисунок 2.7. Схема взаємодії системи PCG та ігрового рушія через API

Приклад словника з метриками поведінки:

```
Dictionary<string, float> behaviorMetrics = new Dictionary<string, float>
{
    { «jumpFrequency», playerController?.jumpFrequency ?? 0f },
    { «moveFrequency», playerController?.moveFrequency ?? 0f },
    { «averageSpeed», playerController?.averageSpeed ?? 0f },
    { «averageJumpLength», averageJumpLength },
    { «averageTimeBetweenJumps», averageTimeBetweenJumps },
    { «averageHeightReached», averageHeightReached },
    { «leftRightMovementRatio», leftRightMovementRatio },
    { «averageReactionTime», averageReactionTime },
    { «riskTakingScore», riskTakingScore },
    { «explorationScore», explorationScore },
    { «precisionScore», precisionScore },
    { «aggressionScore», aggressionScore },
    { «patienceScore», patienceScore },
    { «skillProgression», skillProgression },
    { «learningRate», learningRate }
};
```

Проектування ефективних та надійних інтерфейсів є необхідним кроком для успішної інтеграції розробленої системи персоналізованої PCG у процес створення реальних ігор, що дозволить використовувати її переваги на практиці.

Висновки до розділу 2

Отже, у другому розділі викладено методологію та проектні рішення для системи динамічної та персоналізованої процедурної генерації ігрового контенту. Представлено обґрунтований вибір методів машинного навчання, де для моделювання гравця запропоновано комбінацію керованого та

некерованого підходів, для генерації контенту – гібридні ML-архітектури, а для механізму адаптації – підходи на основі навчання з підкріпленням або умовної генерації, що підкріплено порівняльним аналізом нейромережевих архітектур та вибором відповідних метрик оцінки. Задача дослідження формалізована через розробку математичних моделей контенту, процесу генерації, моделі гравця та функцій адаптації. Спроектовано модульну архітектуру системи, що включає компоненти збору даних, аналізу та моделювання гравця, прийняття рішень щодо адаптації, генерації контенту та інтеграції з грою, а також описано інформаційні потоки, що забезпечують адаптивний цикл. У розділі також деталізовано проектування ключових алгоритмів для моніторингу поведінки гравця, динамічної адаптації контенту та оцінки його відповідності профілю гравця, разом із проектуванням програмних інтерфейсів для інтеграції з ігровими рушіями.

РОЗДІЛ 3.
ПРОГРАМНО-ТЕХНІЧНЕ РІШЕННЯ ДИНАМІЧНОЇ ТА ПЕРСОНАЛІЗОВАНОЇ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВОГО КОНТЕНТУ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

3.1. Обґрунтування вибору інструментарію розробки

Вибір інструментарію розробки є важливим етапом, що безпосередньо впливає на ефективність реалізації, продуктивність та можливості подальшого розвитку системи динамічної та персоналізованої процедурної генерації. Рішення щодо вибору конкретних технологій базується на вимогах до системи, спроектованій архітектурі та алгоритмах. Ключовими факторами при виборі були необхідність тісної інтеграції з ігровим середовищем, підтримка сучасних методів машинного навчання та забезпечення достатньої продуктивності для роботи в режимі реального часу або близькому до нього.

Стандартною та найбільш поширеною мовою скриптування для Unity є C# [30]. Тому саме C# обрано як основну мову програмування для реалізації компонентів системи, що інтегруються безпосередньо в ігровий рушій.

C# надає повний доступ до API Unity, дозволяючи легко керувати ігровими об’єктами, фізикою, анімацією, інтерфейсом та іншими аспектами рушія, що є необхідним для реалізації модулів PlayerController, LevelGenerator, CameraFollow та UIManager [30].

Архітектура Unity, заснована на компонентах, добре узгоджується з модульною структурою системи. Кожен функціональний модуль (збору даних, аналізу, генерації) може бути реалізований як набір C#-скриптів (компонентів), прикріплених до ігрових об’єктів.

Unity та C# мають величезну спільноту розробників, велику кількість документації, навчальних матеріалів та готових рішень (ассетів), що значно спрощує процес розробки [30].

C# у поєднанні з компіляцією IL2CPP в Unity дозволяє досягти високої продуктивності, що важливо для ігрових додатків та обробки даних в реальному часі.

Реалізація функціональності машинного навчання (аналіз поведінки, кластеризація, прогнозування параметрів генерації), зосереджена переважно в компоненті MLContentGenerator.cs та вимагає використання відповідних бібліотек та інструментів. З огляду на використання C# та Unity, а також на задачі, що вирішуються (кластеризація, прогнозування на основі нейронної мережі), обґрунтованим є наступний підхід, що розділяє етапи навчання та виконання моделей:

1. **Навчання моделей (Offline).** Для навчання ML-моделей, зокрема нейронних мереж, що згадуються у функціях PredictParametersFromModel() та PredictParameterWithWeights(), доцільно використовувати потужне та гнучке середовище, яким є Python. Python має розвинену екосистему бібліотек для машинного навчання, таких як:
    a. **TensorFlow або PyTorch.** Провідні фреймворки для розробки та навчання глибоких нейронних мереж. Вони дозволяють створювати складні архітектури для прогнозування параметрів генерації на основі профілю гравця [33].
    b. **Scikit-learn.** Популярна бібліотека для класичних алгоритмів ML, включаючи кластеризацію (для визначення центрів кластерів гравців Explorer, Speedrunner, Cautious тощо), підготовку даних та оцінку моделей [35].

Навчені моделі (нейронні мережі) потім експортуються у стандартизований формат для обміну моделями.

2. **Виконання моделей (Inference in Unity).** Для використання попередньо навчених моделей безпосередньо в ігровому рушії Unity обрано бібліотеку Unity Barracuda [31].

a. **Unity Barracuda.** Це бібліотека від Unity, що дозволяє запускати нейронні мережі безпосередньо в рушії на різних платформах (CPU та GPU). Barracuda працює з моделями у форматі ONNX (Open Neural Network Exchange) [32].
b. **ONNX.** Є відкритим стандартом для представлення моделей машинного навчання. Моделі, навчені у TensorFlow, PyTorch чи Scikit-learn, можуть бути конвертовані у формат ONNX, що дозволяє відокремити процес навчання від процесу виконання [32].

Такий підхід дозволяє виконувати складні нейромережеві обчислення для прогнозування параметрів генерації безпосередньо в Unity, забезпечуючи високу продуктивність та усуваючи необхідність у зовнішніх сервісах під час роботи гри.

3. **Реалізація логіки ML в C#.** Частина логіки машинного навчання може бути реалізована безпосередньо в C# без використання зовнішніх моделей.
    a. **Кластеризація.** Логіка визначення найближчого кластера гравця (FindClosestCluster()) може бути реалізована в C# шляхом обчислення відстані (наприклад, евклідової) між поточним профілем гравця currentPlayerProfile та предефінованими центрами кластерів PlayerCluster.
    b. **Прості моделі/правила.** Інтерполяція параметрів (InterpolateParameters()), коригування параметрів (AdjustParametersForPlayerProfile()) та інші правила адаптації, описані в алгоритмах, реалізуються безпосередньо в коді C# компонента MLContentGenerator.cs.

Обраний інструментарій розробки включає:

- Мову програмування – C# для реалізації всієї логіки в ігровому рушії [30].
- Ігровий рушій – Unity як середовище розробки та виконання гри [30].

- Середовище навчання ML – Python з бібліотеками TensorFlow/PyTorch та Scikit-learn для офлайн-навчання моделей [33-35].
- Формат моделей ML – ONNX для передачі навчених моделей [32].
- Бібліотека виконання ML в Unity – Unity Barracuda для запуску ONNX-моделей нейронних мереж [31].

Такий набір інструментів забезпечує потужне середовище для навчання складних ML-моделей, їх ефективне виконання безпосередньо в грі, тісну інтеграцію з ігровою логікою та використання переваг сучасного ігрового рушія, що повністю відповідає вимогам та задачам даної дипломної роботи.

Вибір ігрового рушія є стратегічним рішенням, що визначає середовище, в якому буде реалізовано та протестовано систему динамічної та персоналізованої процедурної генерації. Для даної роботи було обрано ігровий рушій Unity [30]. Цей вибір обґрунтовується низкою факторів, що роблять Unity оптимальним середовищем для вирішення поставлених задач та інтеграції розроблених компонентів.

По-перше, орієнтація на цільовий жанр та платформу. Проект передбачає реалізацію 2D-платформера у стилі Doodle Jump. Unity надає потужний та зручний інструментарій саме для розробки 2D-ігор, включаючи вбудовану систему 2D-фізики, редактор спрайтів, систему Tilemap для створення рівнів з плиток, інструменти анімації [30]. Це значно спрощує реалізацію базової ігрової механіки та візуального представлення, дозволяючи зосередити зусилля на розробці основної функціональності – процедурної генерації та адаптації.

По-друге, інтеграція з мовою програмування та компонентами системи. Основною мовою програмування обрано C#. Unity використовує C# як основну мову скриптування, що забезпечує безшовну інтеграцію розроблених модулів (PlayerController.cs, PlayerBehaviorLogger.cs, MLContentGenerator.cs, LevelGenerator.cs та ін.) безпосередньо в архітектуру рушія. Компонентно-орієнтована модель Unity [30] дозволяє легко реалізувати спроектовану модульну архітектуру системи, де кожен логічний компонент представлений одним або кількома C#-скриптами, що взаємодіють між собою.

По-третє, підтримка інтеграції машинного навчання. Однією з ключових вимог до системи є використання моделей машинного навчання для аналізу поведінки гравця та керування генерацією контенту. Unity активно розвиває інструменти для інтеграції ML у розробку ігор. Зокрема, використання бібліотеки Unity Barracuda [31] дозволяє виконувати нейромережеві моделі, попередньо навчені в популярних фреймворках (TensorFlow, PyTorch) та експортовані у формат ONNX [32], безпосередньо всередині рушія. Це ідеально відповідає потребам компонента MLContentGenerator.cs, який використовує ML-моделі для прогнозування параметрів генерації. Така вбудована підтримка виконання ML-моделей є суттєвою перевагою Unity для завдань даної роботи.

По-четверте, можливості для збору даних та тестування. Система потребує детального збору даних про поведінку гравця для роботи Модуля аналізу та моделювання. Гнучкість скриптової системи Unity на C# дозволяє легко реалізувати логіку відстеження подій, запису телеметрії та розрахунку метрик, як це передбачено компонентом PlayerBehaviorLogger.cs. Вбудовані інструменти профілювання та налагодження Unity також допомагають аналізувати продуктивність системи та виявляти потенційні проблеми під час розробки та експериментальних досліджень.

По-п’яте, доступність, документація та спільнота. Unity є одним із найпопулярніших ігрових рушіїв у світі, що забезпечує доступ до великої кількості навчальних ресурсів, детальної офіційної документації [30] та активної спільноти розробників. Це значно полегшує вирішення технічних проблем, що можуть виникнути під час реалізації складних алгоритмів PCG та інтеграції ML. Наявність великого магазину ресурсів (Unity Asset Store) також може прискорити розробку, надаючи готові рішення для другорядних завдань.

По-шосте, кросплатформеність. Здатність Unity створювати збірки для різних платформ (PC, мобільні пристрої, веб) підвищує потенційну практичну цінність розробленої системи, уможливлюючи її застосування у ширшому контексті.

Таким чином, вибір Unity як ігрового рушія для інтеграції є обґрунтованим завдяки його потужним інструментам для 2D-розробки, тісній інтеграції з C#, вбудованій підтримці виконання моделей машинного навчання через Barracuda, гнучким можливостям для збору даних, широкій підтримці спільноти та кросплатформеності. Дані фактори створюють сприятливе середовище для успішної реалізації та експериментального дослідження системи динамічної та персоналізованої процедурної генерації ігрового контенту.

3.2. Архітектура та модульна структура програмного продукту

Програмний продукт, розроблений у рамках даної роботи, реалізує систему динамічної та персоналізованої процедурної генерації рівнів для 2D-платформера. В основі програмної реалізації лежить модульна архітектура, принципи якої були закладені на етапі проектування. Система побудована як набір взаємодіючих компонентів (C#-скриптів) у середовищі ігрового рушія Unity [30], що забезпечує тісну інтеграцію з ігровою логікою та візуалізацією.

Загальна архітектура відповідає концептуальній схемі, представленій раніше, та включає модулі, відповідальні за збір даних про гравця, аналіз його поведінки та формування профілю, генерацію контенту на основі цього профілю, а також управління ігровим процесом та інтерфейсом.

Спрощена діаграма компонентів, що ілюструє основні ігрові об’єкти (GameObjects) в сцені Unity та прикріплені до них C#-скрипти, а також ключові залежності та потоки управління/даних між ними представлена на рис. 3.1.

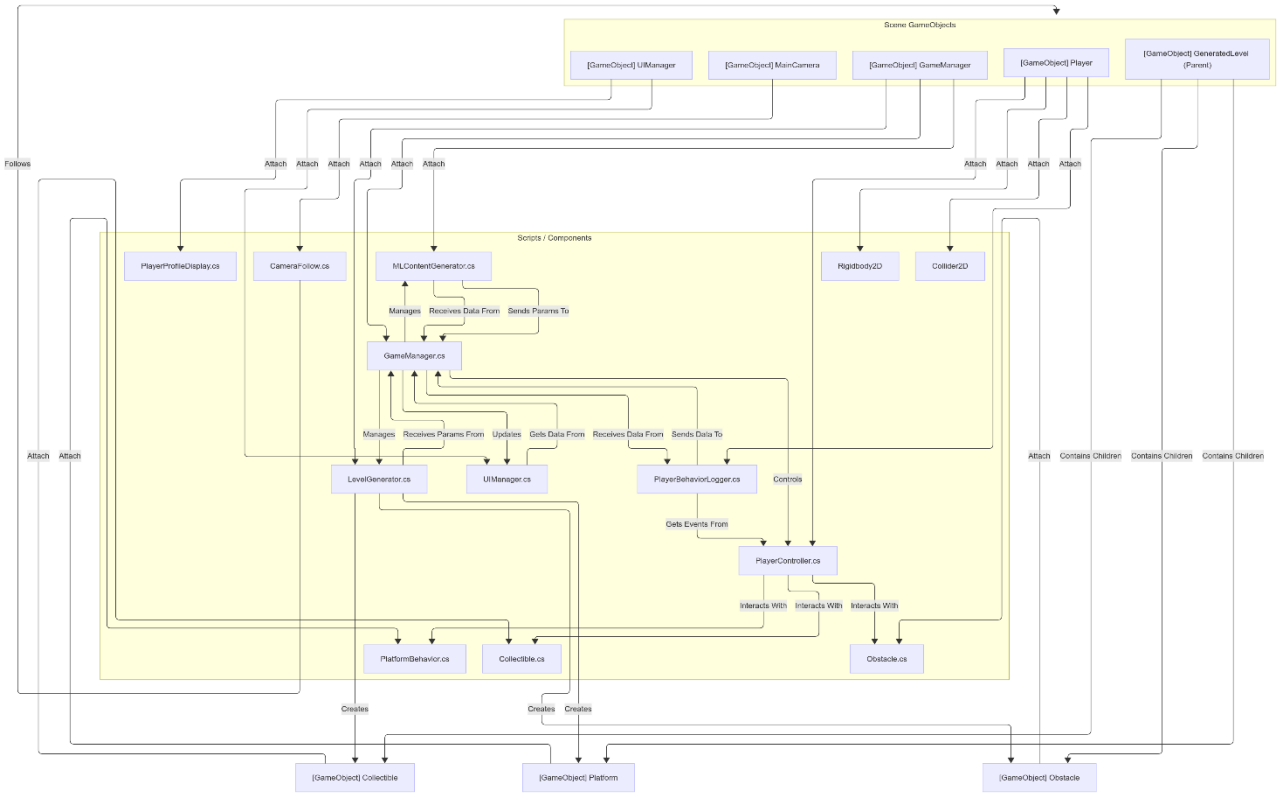

Рисунок 3.1. Архітектура програмного продукту в Unity

Центральним координуючим компонентом системи виступає GameManager. Він відповідає за управління загальним ігровим циклом, ініціалізацію та взаємодію інших ключових модулів. GameManager отримує дані про поведінку гравця, агреговані компонентом PlayerBehaviorLogger, і передає їх до аналітичного модуля MLContentGenerator. На основі результатів аналізу, отриманих від MLContentGenerator у вигляді параметрів генерації, GameManager ініціює створення нового рівня за допомогою компонента LevelGenerator.

Взаємодія між компонентами реалізує адаптивний цикл зворотного зв’язку, що є ядром системи. Цей цикл виглядає наступним чином:

1. PlayerController обробляє введення користувача та взаємодіє з ігровим світом (платформами PlatformBehavior, перешкодами Obstacle, колекційними предметами Collectible), одночасно генеруючи базові події та метрики.

2. PlayerBehaviorLogger отримує ці події, записує детальну інформацію про дії, траєкторію, взаємодії, розраховує розширений набір метрик поведінки та передає їх до GameManager.
3. GameManager пересилає дані про поведінку до MLContentGenerator.
4. MLContentGenerator аналізує отримані дані, оновлює внутрішню модель (профіль) гравця, визначає його ймовірну належність до одного з предефінованих кластерів (Explorer, Speedrunner, Cautious, Beginner, Expert) та, використовуючи свою внутрішню ML-логіку (включаючи потенційне виконання ONNX-моделі через Barracuda [31]), генерує адаптований набір параметрів для наступного рівня.
5. Ці параметри (LevelParameters) передаються через GameManager до LevelGenerator.
6. LevelGenerator використовує отримані параметри для процедурної генерації структури нового рівня, включаючи розміщення платформ, перешкод та колекційних предметів, адаптуючи тип секцій та їх характеристики під профіль гравця.
7. Згенерований рівень стає доступним гравцю, керованому PlayerController, камера CameraFollow адаптує своє положення, а UIManager (через PlayerProfileDisplay) може відображати актуальну інформацію про профіль гравця.
8. Гравець взаємодіє з новим, адаптованим контентом, і цикл повторюється, забезпечуючи постійне пристосування ігрового досвіду до гравця.

Така архітектура, реалізована за допомогою компонентів Unity, забезпечує необхідну гнучкість та ефективність для реалізації динамічної персоналізації в ігровому процесі.

Розглянемо детальніше функціональність основних програмних модулів (компонентів Unity), що складають систему:

GameManager – виконує роль центрального диспетчера (рис. 3.2). Відповідає за запуск та завершення ігрових рівнів, управління станом гри

(наприклад, підрахунок очок, номер рівня). Координує потік даних між логером поведінки (PlayerBehaviorLogger), ML-аналізатором (MLContentGenerator) та генератором рівнів (LevelGenerator). Зберігає та надає доступ до глобальних параметрів гри та поточного стану гравця. Також взаємодіє з UIManager для відображення актуальної інформації.

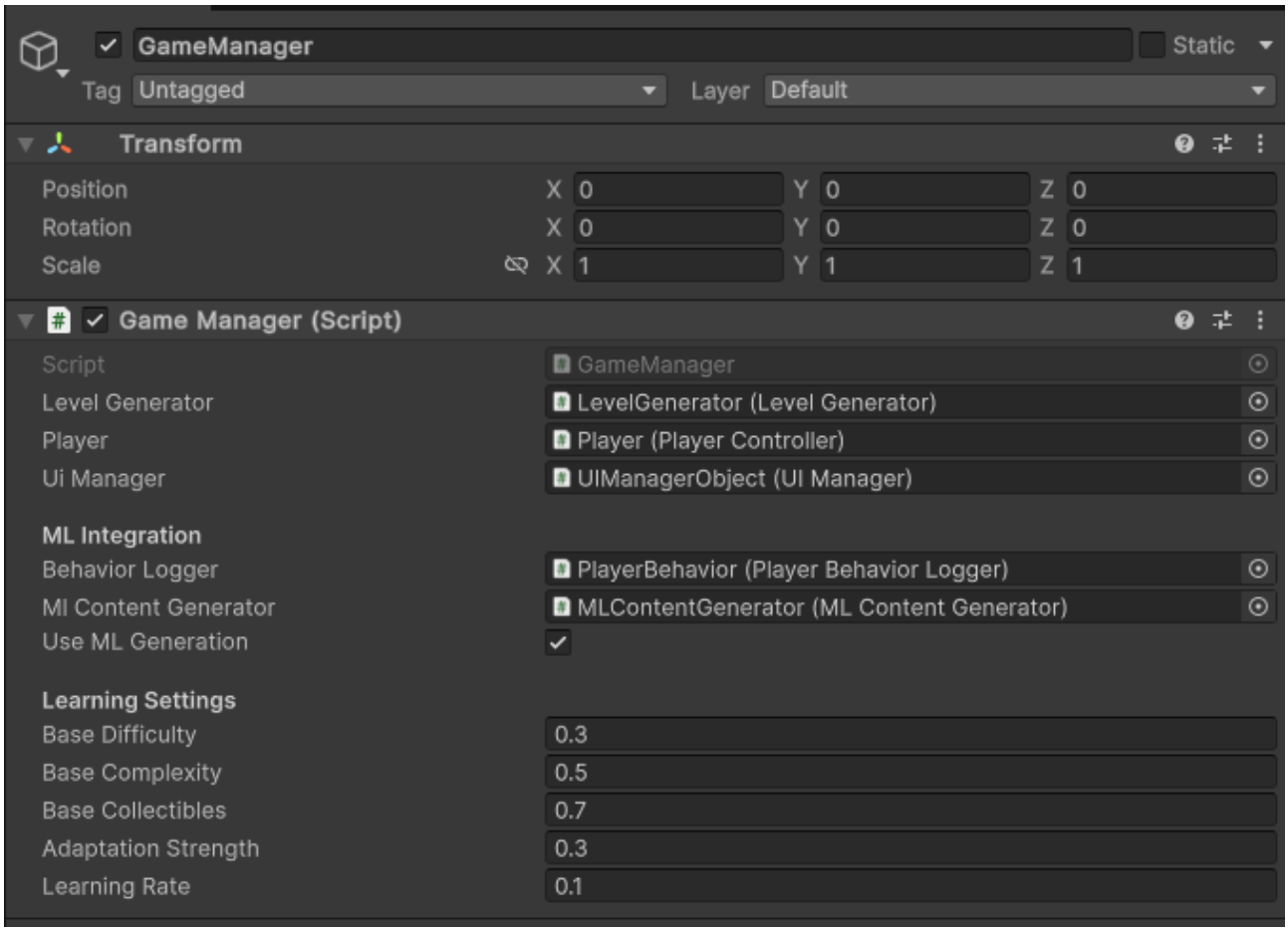

Рисунок 3.2. Компонент GameManager в Unity

PlayerController – реалізує всю логіку керування персонажем гравця (рис. 3.3): обробку введення, переміщення, стрибки, взаємодію з фізичним світом (платформами, перешкодами). Відповідає за збір базових метрик, таких як частота стрибків (jumpFrequency), частота руху (moveFrequency), кількість зібраних предметів (collectiblesPicked), кількість зіткнень (obstaclesHit) тощо. Ці первинні дані передаються для подальшого аналізу.

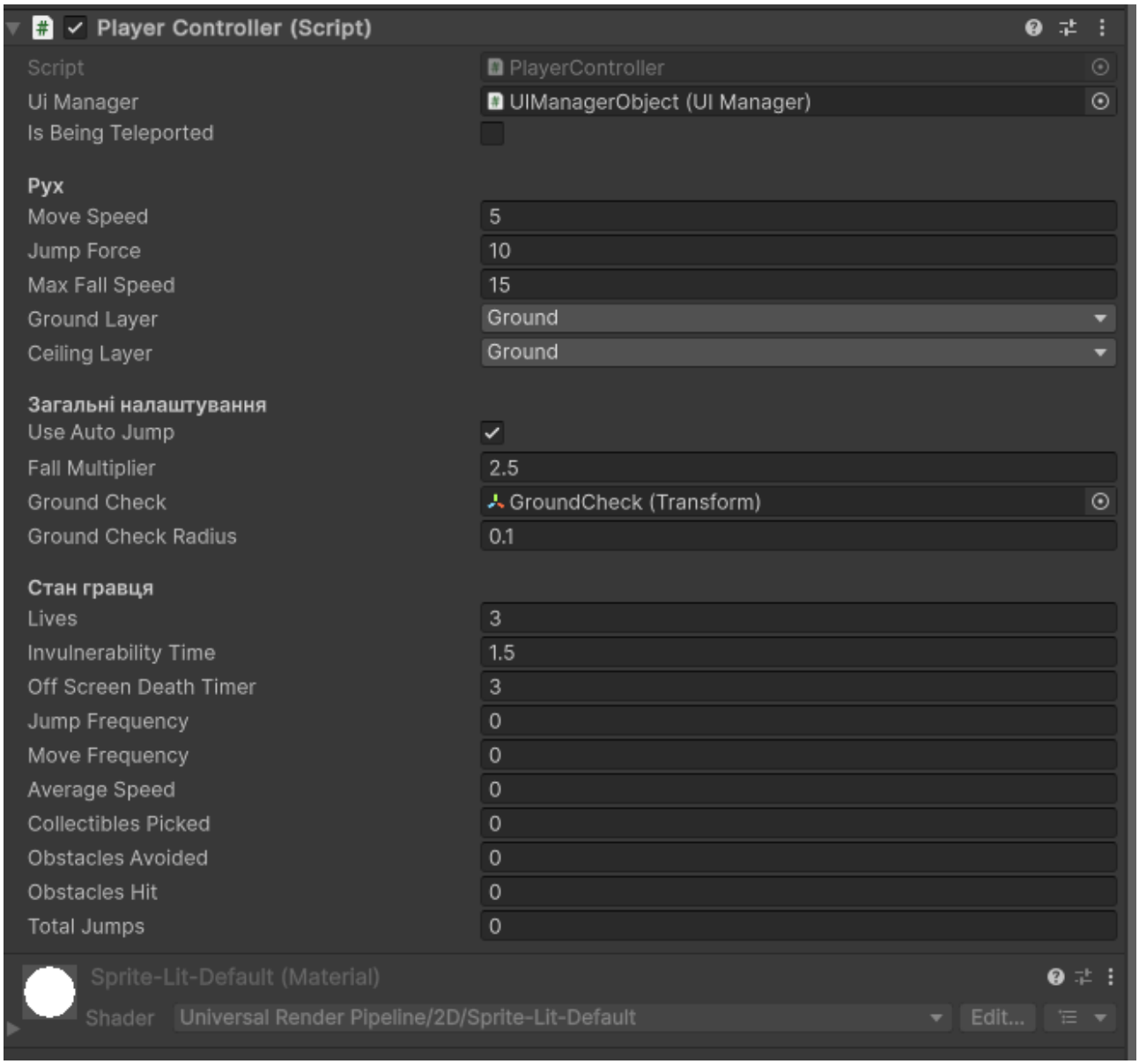

Рисунок 3.3. Компонент PlayerController в Unity

PlayerBehaviorLogger – є ключовим компонентом для реалізації функцій Модуля збору та аналізу даних (рис. 3.4). Цей скрипт детально реєструє дії гравця (PlayerAction) та його взаємодії з оточенням (PlatformInteraction), відстежує траєкторію руху (trajectoryPoints) та формує карту тепла (heatmapData). На основі зібраних даних проводить поглиблений аналіз, розраховуючи складні метрики профілю, такі як averageJumpLength, averageReactionTime, riskTakingScore, explorationScore, precisionScore, aggressionScore, patienceScore. Результати аналізу передаються до MLContentGenerator.

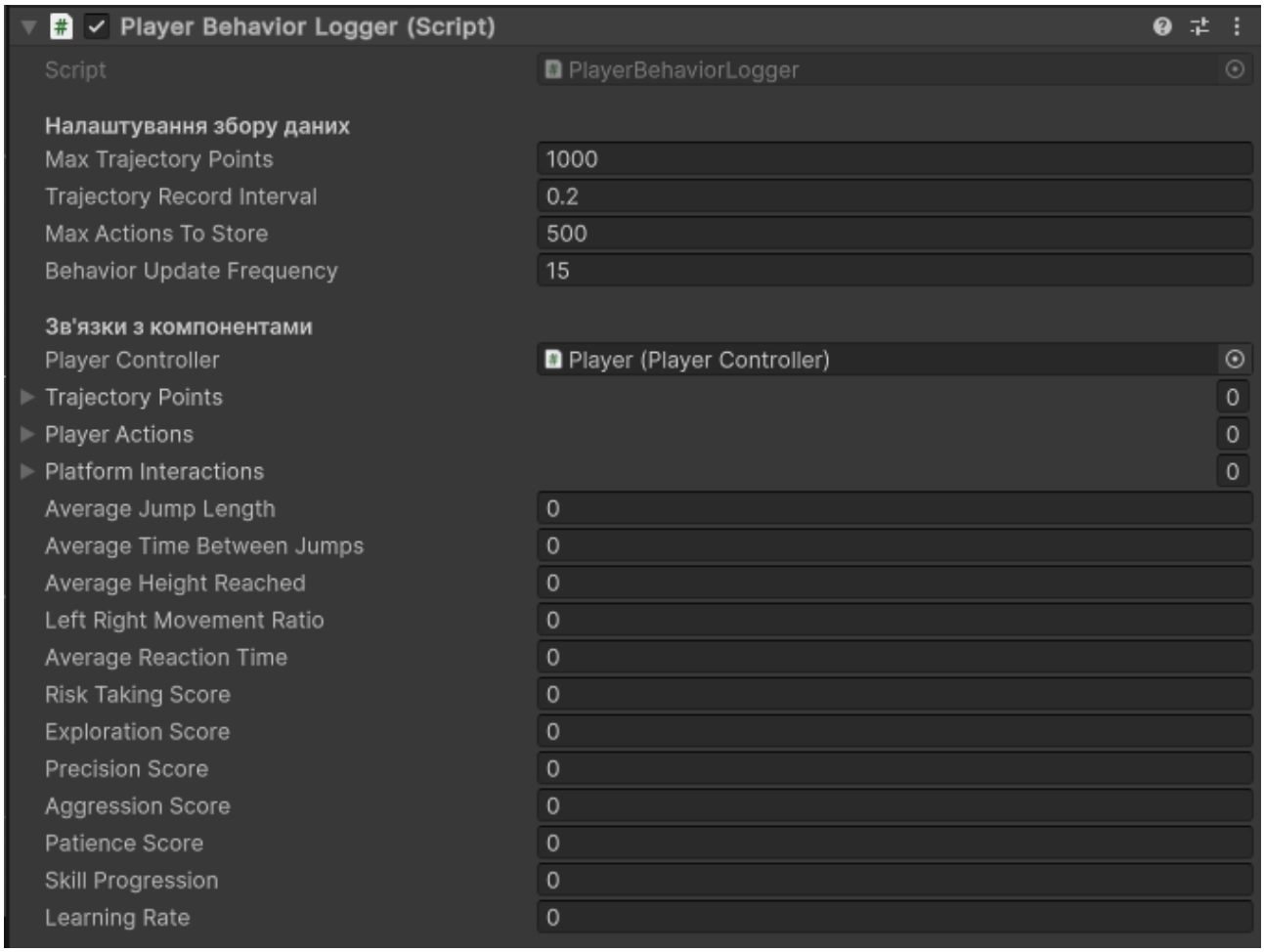

Рисунок 3.4. Компонент PlayerBehaviorLogger в Unity

MLContentGenerator – ядро системи персоналізації, що реалізує функціональність Модуля аналізу та моделювання гравця та Модуля прийняття рішень щодо адаптації (рис. 3.5). Отримує проаналізовані дані від PlayerBehaviorLogger. Оновлює внутрішнє представлення профілю гравця (PlayerProfile), що включає агреговані метрики стилю та навичок. Виконує кластеризацію, визначаючи найближчий архетип гравця (FindClosestCluster()). Найважливішою функцією є генерація параметрів (LevelParameters) для наступного рівня. Цей процес включає використання ML-моделі (PredictParametersFromModel(), потенційно завантаженої через Barracuda [31]) для прогнозування оптимальних значень параметрів (складності, комплексності, розподілу предметів тощо) на основі профілю гравця. Також реалізовано механізми коригування (AdjustParametersForPlayerProfile()) та

плавної інтерполяції параметрів (InterpolateParameters()) для забезпечення стабільної та непомітної адаптації.

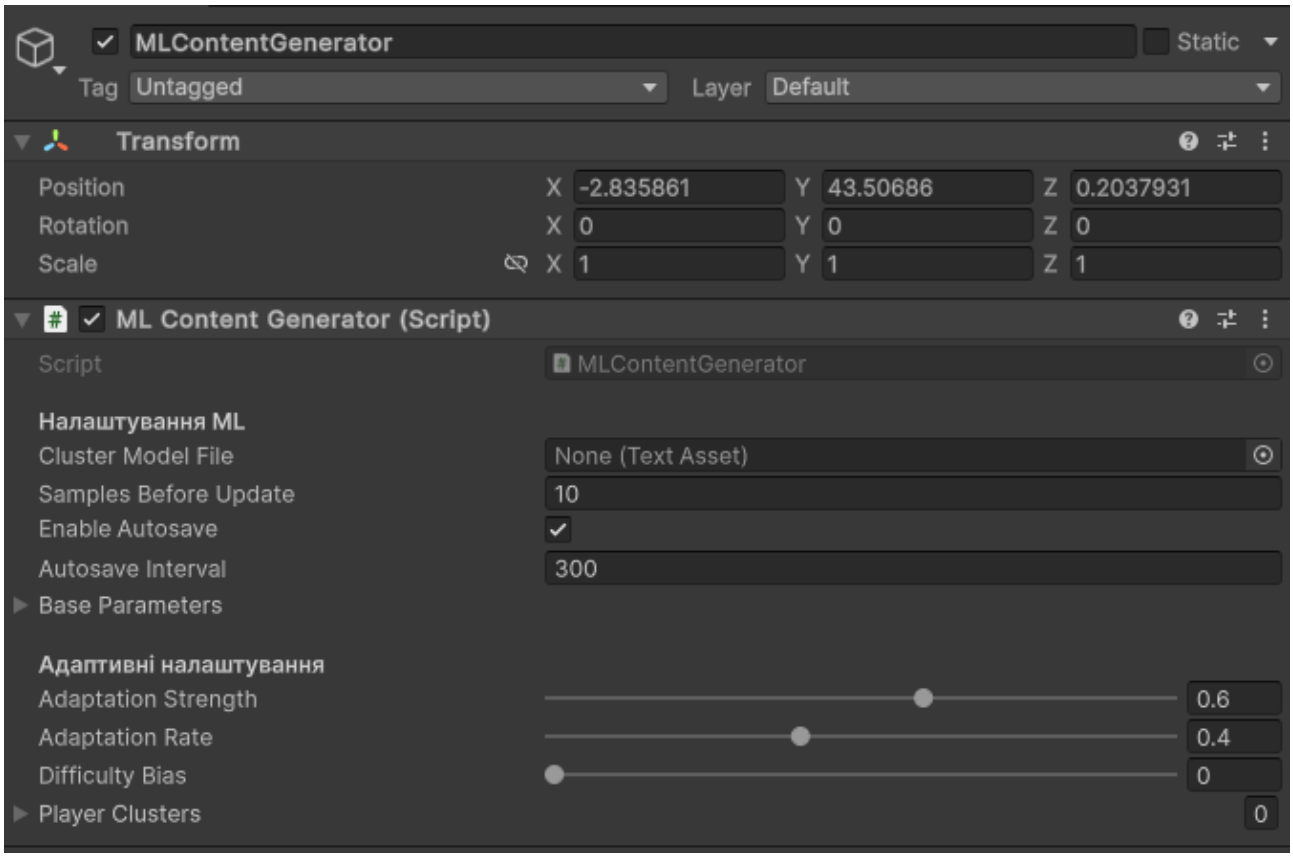

Рисунок 3.5. Компонент MLContentGenerator в Unity

LevelGenerator – відповідає за реалізацію Модуля генерації контенту (рис. 3.6). Використовує параметри, отримані від MLContentGenerator, для створення ігрових рівнів. Підтримує генерацію різних типів секцій (лінійних, вертикальних, з поверненням, лабіринтових), а також специфічний режим генерації у стилі Doodle Jump (GenerateDoodleJumpLevel()). Адаптує характеристики рівня (ширину платформ, частоту перешкод, кількість колекційних предметів, тип секцій) відповідно до отриманих параметрів, тим самим персоналізуючи досвід під конкретного гравця. Включає логіку перевірки та виправлення можливих проблем згенерованого рівня (накладання, недосяжність), а також забезпечення мінімальної кількості необхідних елементів.

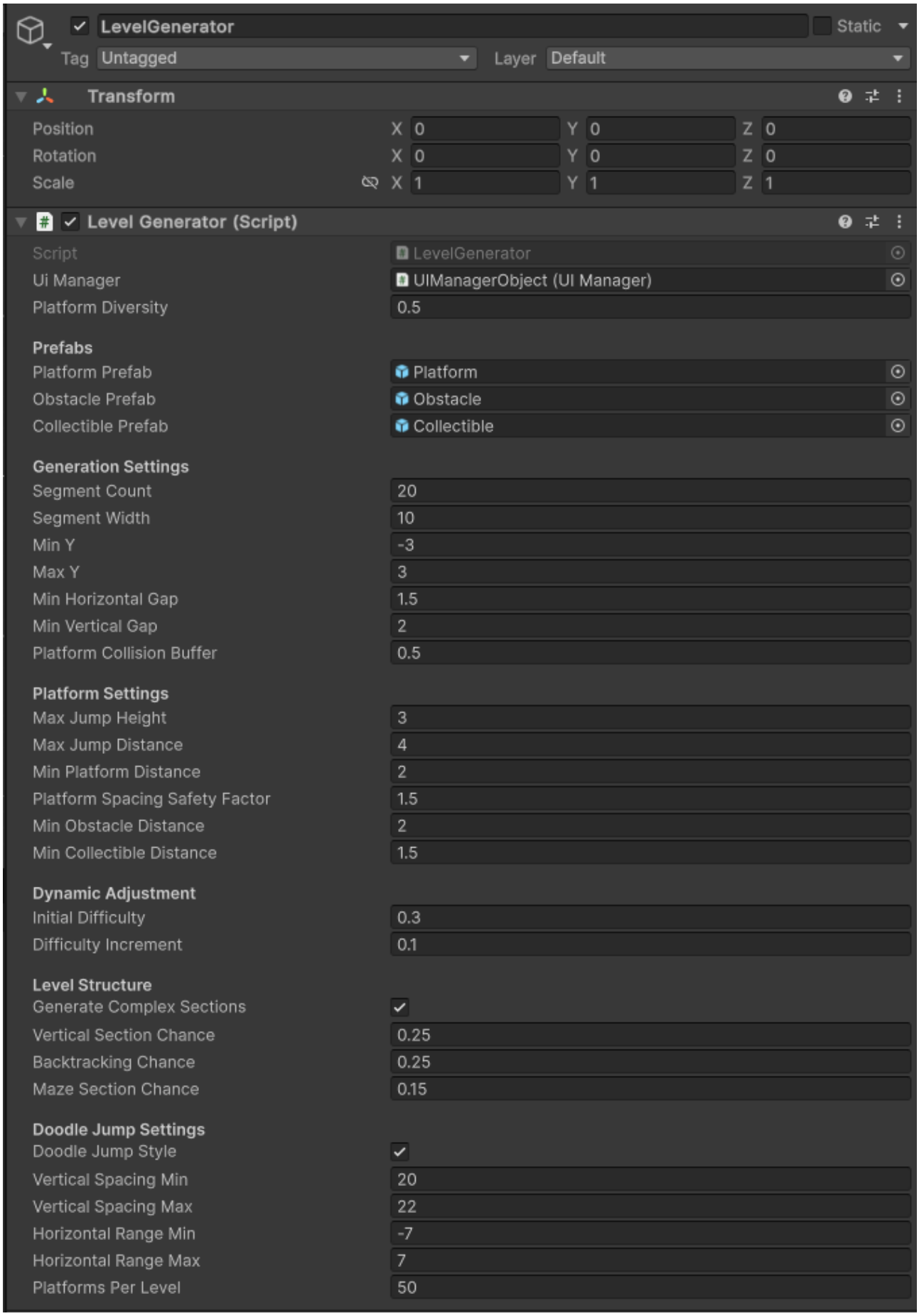

Рисунок 3.6. Компонент LevelGenerator в Unity

CameraFollow – забезпечує плавне слідкування камери за гравцем (рис. 3.7). Реалізує спеціалізовану логіку для режиму Doodle Jump, де пріоритет надається вертикальному руху, а також адаптується до різних ігрових ситуацій для забезпечення комфортного огляду.

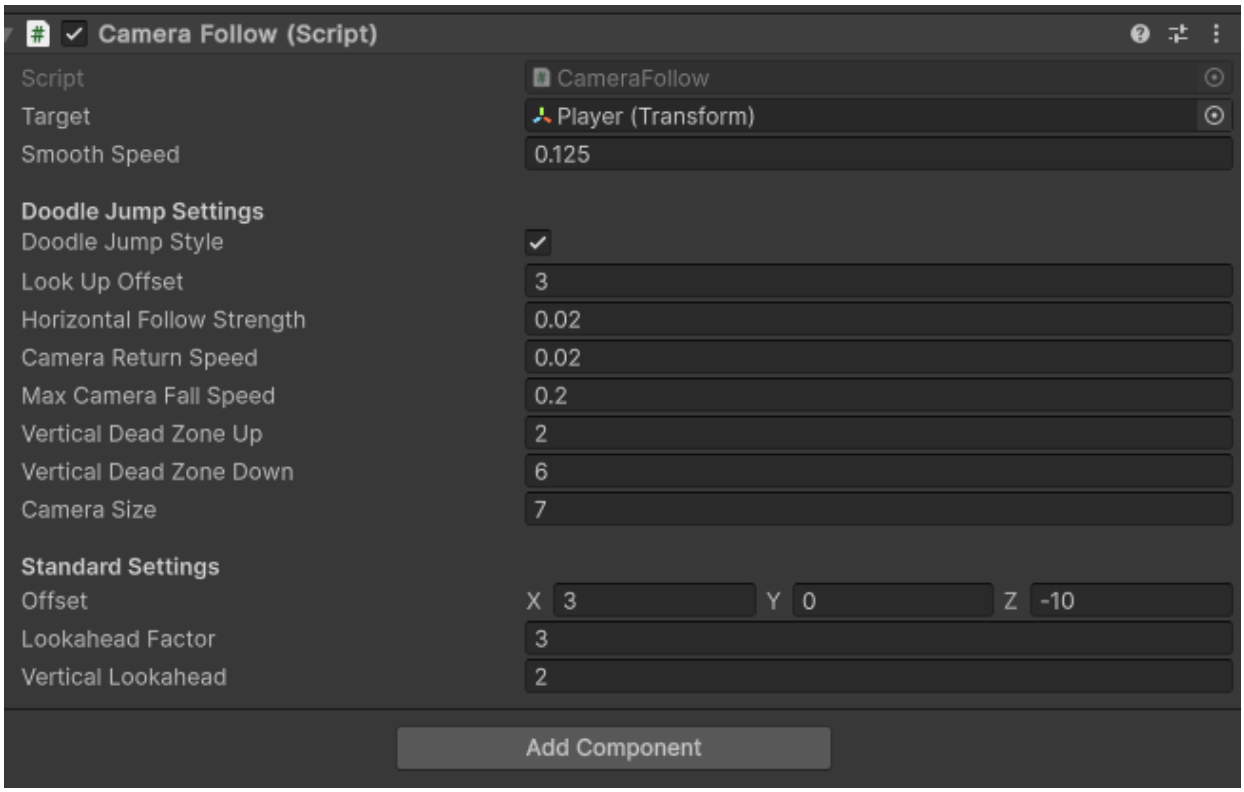

Рисунок 3.7. Компонент CameraFollow в Unity

UIManager / PlayerProfileDisplay (рис. 3.8) – компоненти, відповідальні за відображення користувацького інтерфейсу, включаючи ігрові показники (очки, життя) та, що важливо для тестування та демонстрації, візуалізацію поточного профілю гравця (GetPlayerProfileDetails()).

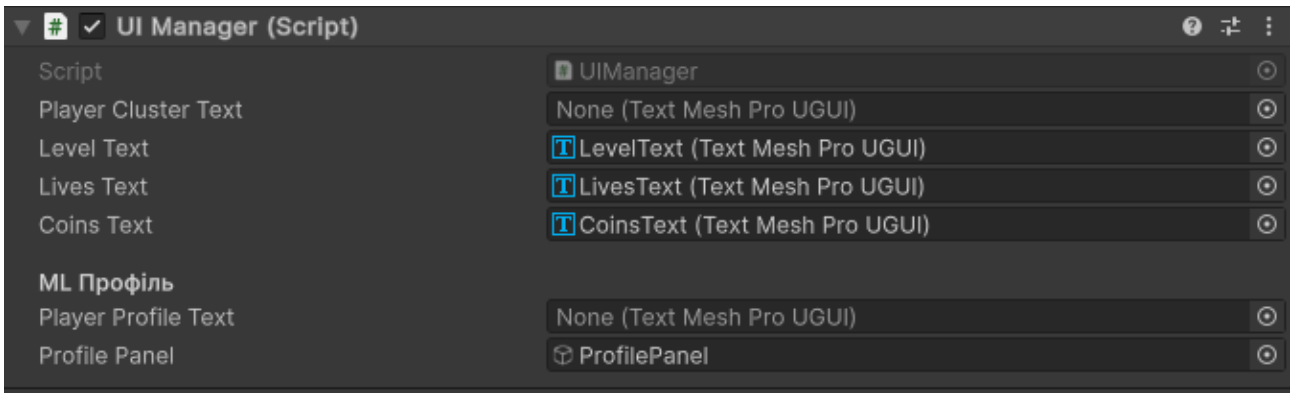

Рисунок 3.8. Компонент CameraFollow в Unity

Кожен з цих модулів реалізує чітко визначену частину загальної функціональності, а їхня взаємодія через визначені інтерфейси (публічні методи та властивості) забезпечує роботу системи як єдиного цілого, реалізуючи поставлені завдання динамічної та персоналізованої генерації контенту.

3.3. Реалізація компонентів системи

Реалізація функціональності модулів збору та аналізу даних (концептуально описаних як Модуль збору даних та Модуль аналізу та моделювання гравця) в основному зосереджена у C#-скриптах PlayerController та, головним чином, PlayerBehaviorLogger. Ці компоненти, працюючи у середовищі Unity [30], забезпечують відстеження дій гравця та перетворення сирих даних телеметрії на структурований профіль, що використовується для адаптації.

UML діаграма класів, що показує основні структури даних (LevelParameters, PlayerProfile, PlayerCluster, PlayerAction, PlatformInteraction) та ключові методи класів-компонентів (MLContentGenerator, PlayerBehaviorLogger, LevelGenerator), відповідальних за аналіз та генерацію представлена на рис. 3.9.

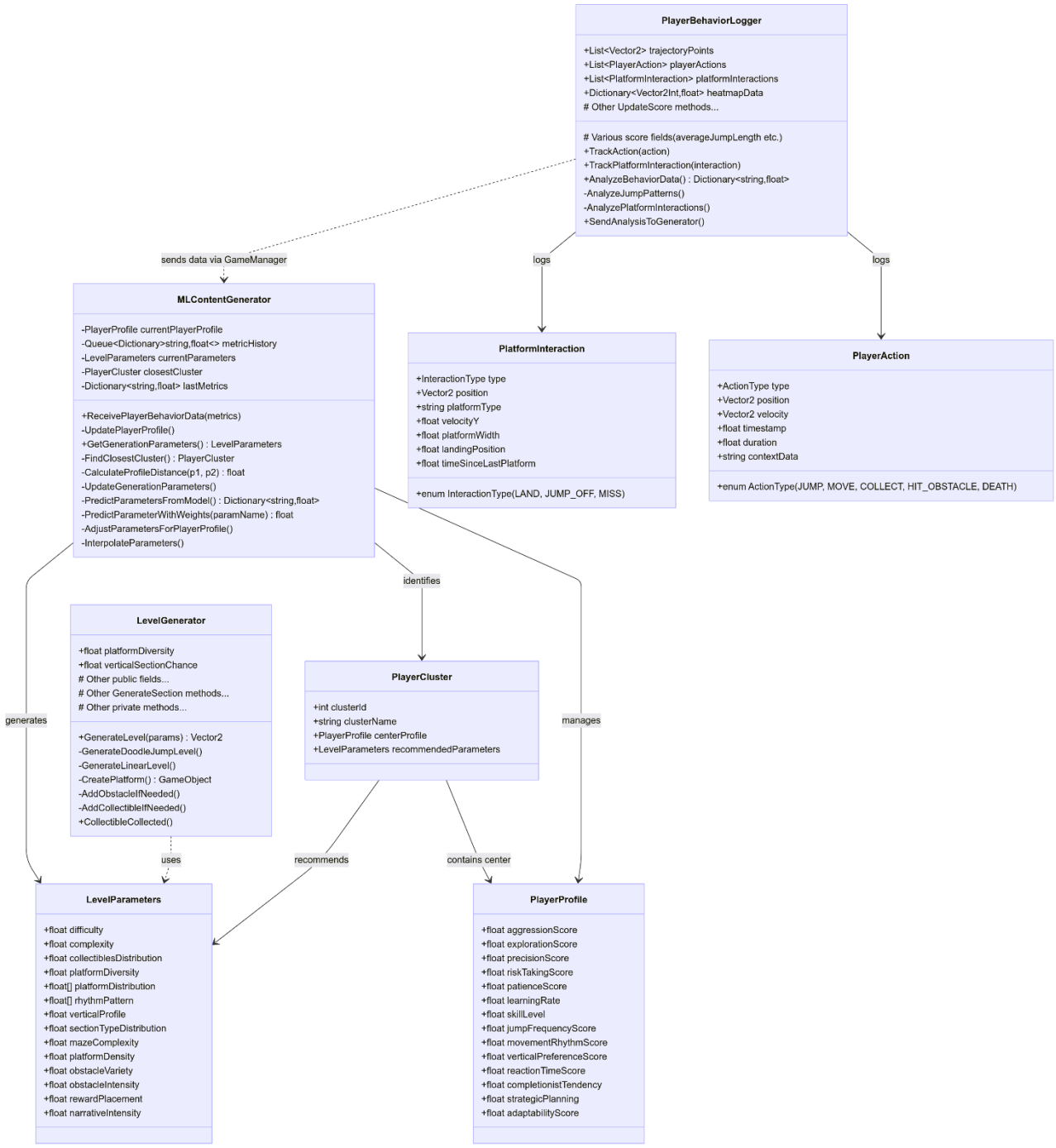

Рисунок 3.9. UML діаграма класів основних структур даних

Компонент PlayerController, окрім основної логіки керування персонажем, здійснює первинний збір базових метрик та ініціює логування подій. Він фіксує такі події, як стрибки, збір предметів (CollectItem), зіткнення з перешкодами (TakeDamage). При виникненні цих подій викликаються відповідні методи у PlayerBehaviorLogger для їх детальної реєстрації.

Основний тягар збору та аналізу даних покладено на PlayerBehaviorLogger. Цей модуль реалізує алгоритм моніторингу та аналізу наступним чином:

1. **Реєстрація подій та траєкторії.** Модуль зберігає детальну інформацію про дії гравця у списку playerActions (типу PlayerAction, що містить тип дії, позицію, швидкість, час тощо) та взаємодії з платформами у списку platformInteractions (типу PlatformInteraction). Також ведеться запис траєкторії руху гравця у списку trajectoryPoints та формується карта тепла (heatmapData), що візуалізує часто відвідувані зони.
2. **Аналіз патернів та розрахунок метрик.** Періодично або за запитом викликається метод AnalyzeBehaviorData(), який обробляє накопичені дані. Внутрішні методи, такі як AnalyzeJumpPatterns() та AnalyzePlatformInteractions(), аналізують відповідні списки подій для розрахунку похідних характеристик (наприклад, середня довжина стрибка averageJumpLength, середня висота averageHeightReached, співвідношення рухів вліво/вправо leftRightMovementRatio). Інші методи, як-от UpdateRiskScore(), UpdatePrecisionScore(), UpdateAggressionScore(), UpdatePatienceScore(), обчислюють більш високорівневі метрики стилю гри на основі співвідношень успішних дій до помилок, часу реакції (averageReactionTime), частоти відвідування унікальних зон (explorationScore) тощо. Ці метрики відповідають атрибутам профілю гравця, визначеним у PlayerProfile.
3. **Передача результатів.** Після аналізу розраховані метрики передаються до модуля MLContentGenerator за допомогою методу SendAnalysisToGenerator().

Скріншот гри (рис. 3.10.) показує персонажа гравця, що стрибає вгору по платформах, розташованих вертикально одна над одною з невеликими горизонтальними зміщеннями. Камера (CameraFollow) знаходиться нижче гравця, забезпечуючи огляд вгору, відповідно до налаштувань doodleJumpStyle.

Платформи різноманітні: стандартні нерухомі, можливо, кілька крихких або рухомих. Між платформами можуть бути розміщені колекційні предмети (напр., монети) та рідкісні перешкоди (напр., статичні шипи). Загальна структура рівня спрямована на вертикальне просування.

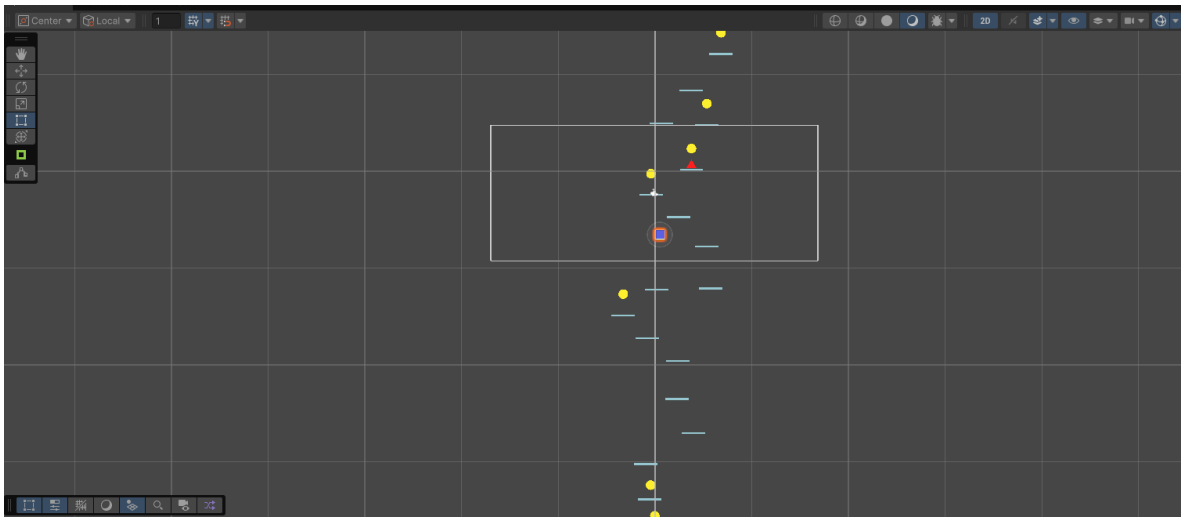

Рисунок 3.10. Приклад згенерованого рівня в стилі Doodle Jump

Інформація з консолі під час генерації рівня представлена на рис 3.11.

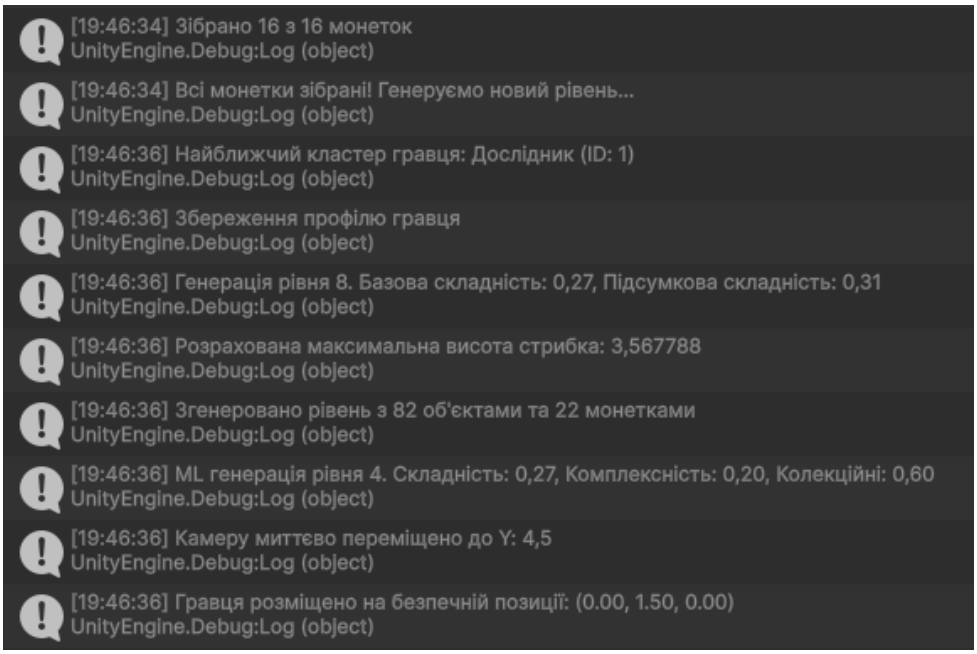

Рисунок 3.11. Дебаг-консоль з інформацією про згенерований рівень

Таким чином, реалізація модуля збору та аналізу використовує стандартні можливості C# та Unity API для відстеження подій та стану ігрових об’єктів,

зберігання даних у колекціях та їх періодичної обробки для формування детального, кількісно вираженого профілю поведінки гравця.

Приклади сценаріїв інтеграційного тестування системи та їх результати представлені в табл. 3.1.

Таблиця 3.1. Результати інтеграційного тестування (приклади сценаріїв)

Тестовий сценарій	Очікувана реакція системи	Фактичний результат
Гравець-новачок. Часто падає, низька точність, повільний прогрес.	MLContentGenerator визначає кластер «Beginner». LevelGenerator створює рівні з низькою difficulty, ширшими платформами, меншою кількістю перешкод.	Профіль гравця коректно класифіковано. Згенеровані рівні помітно спрощені, щільність платформ збільшена.
Гравець стає обережним. Починає уникати ризику, повільно рухається, досліджує.	MLContentGenerator змінює кластер на «Cautious» або «Explorer». Параметри difficulty та complexity помірні, збільшується collectiblesDistribution, platformDensity.	Кластер змінено. Збільшилася кількість предметів, складність зросла помірно, структура стала дещо складнішою.
Гравець грає агресивно. Швидко рухається, часто стрибає, ігнорує предмети, ризикує.	MLContentGenerator визначає кластер «Speedrunner». Збільшуються difficulty, complexity, obstacleIntensity, зменшується mazeComplexity, collectiblesDistribution.	Кластер визначено коректно. Рівні стали складнішими, з більшою кількістю перешкод та довшими стрибками. Кількість предметів зменшилася.
Помилка передачі даних. PlayerBehaviorLogger н е надсилає дані.	MLContentGenerator використовує останній відомий профіль або дефолтні параметри. LevelGenerator генерує рівень на основі цих параметрів.	Генерація продовжується з останніми відомими параметрами. Система не «падає». Виведено попередження в лог.
Гравець швидко прогресує. Показник skillLevel швидко зростає.	MLContentGenerator плавно збільшує параметри difficulty та complexity відповідно до adaptationRate.	Параметри генерації поступово зростають, уникаючи різких стрибків складності.

Реалізація генеративних функцій системи охоплює два основні компоненти: MLContentGenerator, який використовує ML для визначення параметрів генерації, та LevelGenerator, який безпосередньо створює контент рівня на основі цих параметрів.

Компонент MLContentGenerator передбачає використання нейронної мережі через метод PredictParametersFromModel(). Цей метод взаємодіє з попередньо навченою нейронною мережею, завантаженою у форматі ONNX [32] за допомогою бібліотеки Unity Barracuda [31]. Ця нейронна мережа, навчена офлайн, приймає на вхід поточний профіль гравця (currentPlayerProfile) і прогнозує набір вихідних значень, що відповідають оптимальним параметрам генерації (LevelParameters) для цього гравця. Така архітектура дозволяє використовувати потужні глибокі моделі для складного відображення профілю гравця у параметри контенту, не перевантажуючи код C# деталями реалізації самої мережі.

Компонент LevelGenerator реалізує безпосередньо процедурний алгоритм створення рівня. Основні аспекти реалізації:

1. **Отримання параметрів.** Метод GenerateLevel приймає на вхід основні параметри (складність difficulty, комплексність complexity, розподіл предметів collectiblesAmount), отримані від MLContentGenerator. Він також використовує інші параметри, такі як platformDiversity, verticalSectionChance, mazeSectionChance, doodleJumpStyle, які також можуть бути встановлені на основі виводу ML-модуля.
2. **Вибір стилю та структури.** Залежно від параметра doodleJumpStyle та інших умов (наприклад, номер рівня, параметр generateComplexSections), алгоритм обирає або специфічну логіку генерації вертикального рівня (GenerateDoodleJumpLevel), або стандартну генерацію, що передбачає послідовне створення різних типів секцій (лінійних GenerateLinearLevel/GenerateForwardSection, вертикальних GenerateVerticalSection, з поверненням

GenerateBacktrackingSection, лабіринтових GenerateMazeSection) відповідно до ймовірностей, що можуть залежати від параметрів ϕ_t.

3. **Створення об'єктів.** У процесі генерації секцій викликаються методи для створення окремих ігрових об'єктів: CreatePlatform() (створює платформи різних типів та розмірів відповідно до platformDiversity та difficulty), AddObstacleIfNeeded() (додає перешкоди з ймовірністю, що залежить від complexity та позиції), AddCollectibleIfNeeded() (додає колекційні предмети відповідно до collectiblesDistribution). Ці методи використовують префаби Unity для інстанціювання об'єктів у сцені.
4. **Постобробка та валідація.** Після основного етапу генерації виконуються кроки для виправлення можливих проблем (накладання платформ, недосяжність) та забезпечення мінімальних вимог до рівня (мінімальна кількість платформ та предметів EnsureMinimumCollectibles, EnsureMinimumPlatforms).

Отже, генеративний модуль LevelGenerator реалізує гнучкий алгоритм PCG, керований параметрами, що надходять від ML-системи, дозволяючи створювати різноманітні та адаптовані рівні.

Функціональність адаптації та балансування, тобто перетворення профілю гравця на параметри генерації (реалізація функції $A(m_t) = \phi_t$), зосереджена в компоненті MLContentGenerator. Алгоритм динамічної адаптації реалізований через послідовність методів всередині цього скрипта:

1. **Оновлення профілю.** Метод ReceivePlayerBehaviorData отримує дані від PlayerBehaviorLogger. Потім викликається UpdatePlayerProfile, який агрегує нові метрики з історією (metricHistory) та оновлює currentPlayerProfile (об'єкт класу PlayerProfile).
2. **Визначення кластера.** Метод FindClosestCluster порівнює currentPlayerProfile з центроїдами предефінованих кластерів (PlayerCluster, що містять centerProfile та recommendedParameters) за допомогою функції CalculateProfileDistance (ймовірно, обчислення

евклідової відстані у просторі ознак профілю). Визначається найближчий кластер closestCluster.

3. **Визначення цільових параметрів.** Основна логіка адаптації знаходиться у методі UpdateGenerationParameters. Він комбінує кілька підходів:
    a. **Прогнозування моделлю.** Викликається PredictParametersFromModel(), що, як обговорювалося, використовує зовнішню нейромережеву модель для отримання базового прогнозу параметрів.
    b. **Вплив кластера.** Враховуються рекомендовані параметри (recommendedParameters) для визначеного кластера closestCluster.
    c. **Коригування та інтерполяція.** Застосовуються додаткові коригування (AdjustParametersForPlayerProfile()) та плавна інтерполяція (InterpolateParameters()) між поточними та цільовими параметрами. Це дозволяє врахувати не лише належність до кластера, але й специфічні значення метрик поточного гравця, а також забезпечити плавність змін у геймплеї, використовуючи параметри adaptationStrength та adaptationRate. Параметр difficultyBias може використовуватися для загального зміщення рівня складності.
4. **Надання параметрів.** Отриманий адаптований набір параметрів currentParameters (типу LevelParameters) стає доступним для GameManager через метод GetGenerationParameters().

Реалізація модуля адаптації поєднує кластеризацію для визначення загального стилю гравця, потенційне використання нейронних мереж для точного прогнозування та набір правил і механізмів інтерполяції для тонкого налаштування та забезпечення плавного балансування ігрового досвіду.

Приклади сценаріїв інтеграційного тестування системи та їх результати представлені в табл. 3.2.

Таблиця 3.2. Ілюстративні середні значення параметрів генерації рівнів, що застосовувалися системою для гравців, віднесених до різних кластерів

Кластер гравця	**Сер. Difficulty**	**Сер. Complexity**	**Сер. CollectiblesDistr.**	**Сер. PlatformDensity**	**Сер. MazeComplexity**
Новачок	0.25	0.20	0.50	0.80	0.10
Обережний	0.50	0.40	0.60	0.70	0.40
Швидкісний	0.70	0.65	0.30	0.50	0.20
Дослідник	0.40	0.50	0.90	0.60	0.65
Експерт	0.85	0.80	0.40	0.60	0.75

Загалом, програмна реалізація компонентів системи відповідає спроектованій архітектурі та алгоритмам, використовуючи можливості Unity та C# для інтеграції ігрової логіки, збору даних та виконання моделей машинного навчання для досягнення мети динамічної персоналізації.

Висновки до розділу 3

Таким чином, третій розділ представляє результати програмної реалізації та дослідження розробленої системи динамічної та персоналізованої процедурної генерації контенту. Обґрунтовано вибір інструментарію розробки, зокрема використання ігрового рушія Unity, мови C#, бібліотеки Barracuda для виконання нейромережевих моделей у форматі ONNX та середовища Python для їхнього офлайн-навчання. Деталізовано архітектуру та модульну структуру реалізованого програмного продукту, описано взаємодію ключових компонентів (GameManager, PlayerBehaviorLogger, MLContentGenerator, LevelGenerator) та представлено основні аспекти їхньої програмної реалізації. Наведено результати тестування системи, які підтвердили її працездатність, коректність взаємодії модулів та прийнятну продуктивність. На основі проведеної роботи також проаналізовано практичну цінність системи та окреслено перспективи її подальшого розвитку.

РОЗДІЛ 4.
ЕРГОНОМІЧНІ АСПЕКТИ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Ергономічні вимоги до інтерфейсів системи процедурної генерації

Система персоналізованої PCG взаємодіє з двома різними категоріями користувачів, що зумовлює необхідність розгляду різних наборів ергономічних вимог. З одного боку, це розробники та геймдизайнери, яким потрібні інструменти для інтеграції, конфігурації та моніторингу системи. З іншого боку, це гравці, для яких ключовим є якість та сприйняття динамічно адаптованого ігрового процесу.

Інтерфейс, призначений для розробників та геймдизайнерів, повинен забезпечувати зручне та ефективне управління системою персоналізованої PCG в рамках процесу розробки гри (рис. 4.1). Від його ергономічності залежить швидкість інтеграції системи, легкість її налаштування під специфіку конкретного проекту та можливість контролювати її поведінку. Основні ергономічні вимоги до цього інтерфейсу включають:

1. Простота інтеграції – процес підключення системи до ігрового проекту в Unity [30] має бути максимально простим та добре документованим. Інтерфейси API, що були описані раніше, повинні бути логічно структурованими, з чіткими назвами методів та параметрів, що полегшує їх використання у C#-скриптах гри.
2. Інтуїтивність конфігурації – розробник повинен мати можливість легко налаштовувати ключові аспекти системи через графічний інтерфейс (наприклад, інспектор Unity для компонентів MLContentGenerator та LevelGenerator) або конфігураційні файли. Це включає:
    a. Визначення та налаштування параметрів профілю гравця (PlayerProfile), що відстежуються.
    b. Конфігурацію кластерів гравців (PlayerCluster), їх центроїдів та рекомендованих параметрів генерації.

c. Налаштування діапазонів та базових значень для параметрів генерації контенту (LevelParameters, таких як difficulty, complexity, platformDiversity тощо).
d. Керування параметрами адаптації: встановлення сили (adaptationStrength) та швидкості (adaptationRate) адаптації, визначення зміщення складності (difficultyBias). Використання візуальних елементів (слайдерів, полів введення, випадних списків) є бажаним.

3. Візуалізація та моніторинг – для розуміння роботи системи та її налагодження необхідні інструменти візуалізації. Інтерфейс повинен надавати можливість:
   a. Відстежувати поточний профіль гравця в реальному часі (що частково реалізовано PlayerProfileDisplay).
   b. Бачити, до якого кластера система відносить гравця.
   c. Спостерігати за поточними значеннями параметрів генерації, що визначаються ML-модулем.
   d. Бажано мати можливість візуального попереднього перегляду фрагменту рівня, що генерується з поточними параметрами.
4. Контрольованість та налагодження – розробник повинен мати засоби для контролю та налагодження системи:
   a. Можливість тимчасового вимкнення адаптації для тестування базової генерації.
   b. Можливість «заморозити» профіль гравця або вручну встановити певні параметри для перевірки генерації у конкретних умовах.
   c. Детальні та інформативні повідомлення у консолі Unity (логи) про стан системи, прийняті рішення щодо адаптації та можливі помилки.
5. Гнучкість та розширюваність – інтерфейс конфігурації має бути спроектований таким чином, щоб можна було відносно легко додавати

нові метрики для аналізу поведінки, нові параметри генерації або нові кластери гравців без необхідності значної перебудови інтерфейсу.

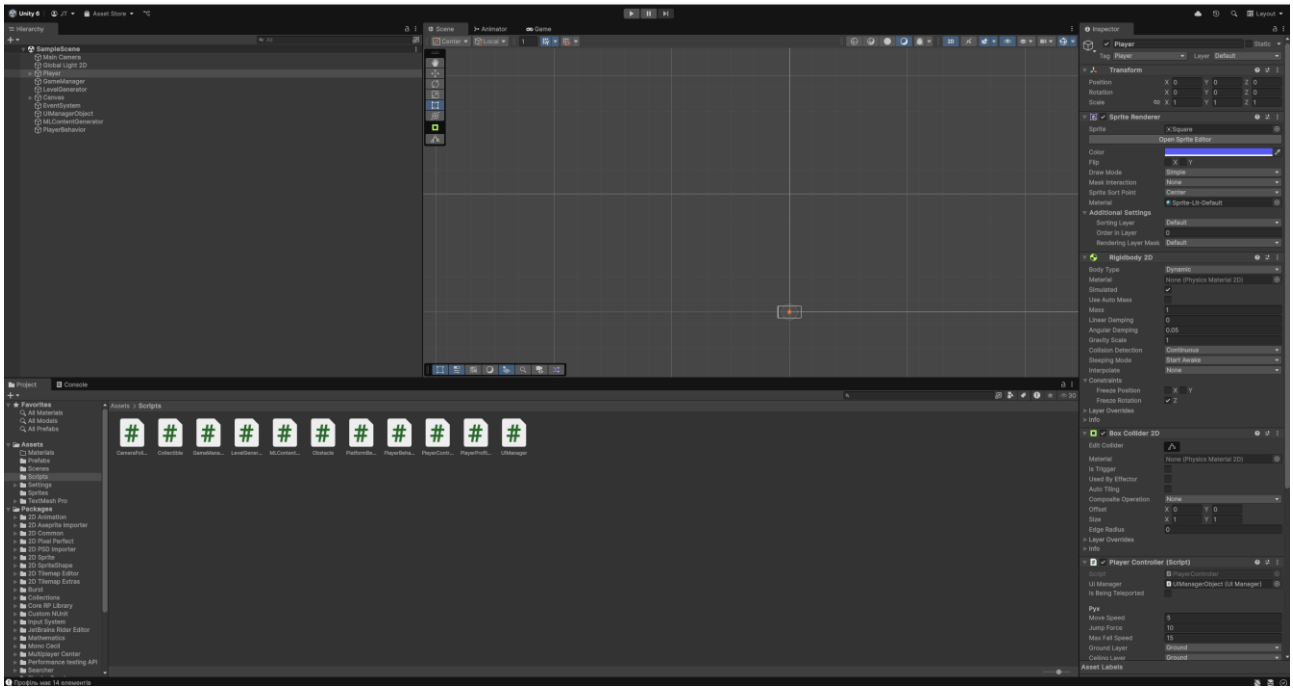

Рисунок 4.1. Інтерфейс розробника Unity

Дотримання цих вимог дозволить знизити поріг входження для використання системи розробниками, прискорити процес інтеграції та налаштування, а також забезпечити необхідний рівень контролю над процесом персоналізованої генерації контенту.

Ергономіка системи з точки зору гравця стосується не прямої взаємодії з інтерфейсом, а сприйняття результатів роботи системи – динамічно адаптованого контенту. Навіть якщо система технічно коректно адаптує параметри, невдала реалізація адаптації може негативно вплинути на користувацький досвід (User Experience, UX). Основні ергономічні вимоги до процесу динамічної адаптації для забезпечення позитивного UX включають:

1. Непомітність та плавність – адаптація параметрів гри (складності, типу викликів) має відбуватися максимально плавно та непомітно для гравця. Різкі, очевидні зміни у складності можуть руйнувати занурення та викликати відчуття штучності або маніпуляції. R. Hunicke [29] зазначала, що очевидне динамічне налаштування складності може

знецінювати досягнення гравця. Використання механізмів інтерполяції параметрів (як у методі InterpolateParameters() в MLContentGenerator) є важливим для забезпечення плавності переходів.

2. Відчуття справедливості та контролю – незважаючи на адаптацію, гравець повинен відчувати, що його дії та навички мають значення, а виклики є справедливими. Адаптація не повинна створювати непереборних перешкод або, навпаки, робити гру настільки легкою, що зникає відчуття досягнення. Гравець має зберігати відчуття контролю над ситуацією.
3. Релевантність та доцільність – зміни, що вносяться системою, мають сприйматися гравцем як доцільні та релевантні його поточній ситуації. Наприклад, якщо гравець демонструє високі навички, підвищення складності має виглядати як природний виклик, а не як штучне покарання. Адаптація до стилю гри (наприклад, більше дослідження для «дослідника») має відповідати очікуванням, що формуються цим стилем.
4. Уникнення негативних циклів – система адаптації не повинна потрапляти у негативні цикли. Наприклад, якщо гравець тимчасово грає погано через втому, надмірне підвищення складності може ще більше погіршити його стан та продуктивність. Аналогічно, постійне зниження складності для гравця, що зазнає труднощів, може перешкоджати його навчанню та розвитку навичок. Необхідний баланс між адаптацією та наданням можливості для подолання труднощів.
5. Відповідність контексту гри – адаптація повинна враховувати поточний ігровий контекст. Наприклад, під час сюжетних моментів або навчальних секцій може бути доцільним тимчасово обмежити або вимкнути адаптацію, щоб не порушувати запланований дизайнерський досвід.
6. Підтримання узгодженості – згенерований адаптований контент повинен залишатися узгодженим із загальним стилем, правилами та

логікою ігрового світу. Адаптація не повинна призводити до генерації контенту, що виглядає чужорідним або порушує встановлені ігрові механіки.

Забезпечення відповідності цим ергономічним вимогам з боку гравця є складним завданням, що вимагає ретельного проектування не лише ML-моделей та алгоритмів генерації, але й самої логіки адаптації, а також її тестування з реальними користувачами для оцінки суб'єктивного сприйняття.

4.2. Проектування інтерфейсу користувача

Проектування інтерфейсів для системи динамічної та персоналізованої PCG охоплює два основні напрямки. По-перше, це інтерфейс для розробників та геймдизайнерів, що дозволяє конфігурувати, моніторувати та налагоджувати систему в процесі розробки гри. По-друге, це елементи, що стосуються взаємодії кінцевого користувача (гравця) з системою, хоча ця взаємодія переважно є непрямою, через сприйняття адаптованого контенту. Однак, у рамках розробки та тестування було також реалізовано специфічний інтерфейсний елемент для візуалізації стану системи гравцю або тестувальнику. Проектування цих інтерфейсів базувалося на ергономічних вимогах.

Інтерфейс конфігурації для розробника (Unity Inspector):

- **Концептуальна модель.** Основною моделлю взаємодії розробника з системою є використання стандартного інспектора Unity [30]. Кожен ключовий компонент системи (MLContentGenerator, LevelGenerator, PlayerBehaviorLogger тощо) представлений як C#-скрипт, прикріплений до ігрового об'єкта, а його публічні поля та властивості відображаються в інспекторі, дозволяючи здійснювати конфігурацію безпосередньо у редакторі. Така модель відповідає стандартному робочому процесу розробки в Unity, що робить її інтуїтивно зрозумілою для цільових користувачів.

- **Прототипи (опис панелей інспектора):**
  - MLContentGenerator Inspector. Ця панель є центральною для налаштування ML-аспектів. Вона містить поля для:
    - Призначення файлу моделі ONNX [32] (якщо використовується Barracuda [31]).
    - Налаштування параметрів кластеризації: посилання на ассети (наприклад, ScriptableObjects), що описують центроїди кластерів (PlayerCluster) та їх рекомендовані параметри генерації.
    - Регулювання параметрів адаптації: слайдери або числові поля для adaptationStrength, adaptationRate, difficultyBias.
    - (Опціонально) Секція для налагодження, що відображає в реальному часі поточний вектор профілю currentPlayerProfile, визначений closestCluster та поточні вихідні параметри currentParameters. (Див. Рисунок X.1 – Прототип панелі інспектора MLContentGenerator).
  - LevelGenerator Inspector: Панель для налаштування параметрів самої генерації:
    - Призначення префабів для різних типів платформ, перешкод, колекційних предметів.
    - Налаштування базових параметрів генерації (якщо вони не повністю визначаються MLContentGenerator), наприклад, діапазони розмірів секцій.
    - Налаштування ймовірностей вибору типів секцій (verticalSectionChance, mazeSectionChance тощо).
    - Перемикач для вибору стилю генерації (doodleJumpStyle).
  - PlayerBehaviorLogger Inspector: Може містити налаштування, пов’язані з частотою логування, періодом аналізу, можливо, ваговими коефіцієнтами для різних метрик при формуванні агрегованих показників стилю гри. Також може мати секцію для

відображення останніх записаних подій або розрахованих метрик у режимі відтворення.

- (Додатково) Для конфігурації кластерів (PlayerCluster) та, можливо, параметрів генерації (LevelParameters) доцільно використовувати ScriptableObjects Unity. Це дозволяє створювати конфігураційні ассети, які легко редагувати, дублювати та перепризначати, відокремлюючи дані конфігурації від логіки компонентів.

Інтерфейс відображення профілю гравця (In-Game UI):

- **Концептуальна модель.** Це інформаційна панель, що відображається поверх основного ігрового інтерфейсу за запитом користувача (тестувальника або розробника). Її мета – забезпечити прозорість роботи системи аналізу поведінки гравця, показуючи поточний стан його профілю, як його «бачить» система. Це не є частиною основного геймплею, а радше інструментом для налагодження, демонстрації та досліджень.
- **Прототип (PlayerProfileDisplay).** Реалізовано як компонент PlayerProfileDisplay, що використовує TextMeshPro для виведення текстової інформації. Прототип інтерфейсу являє собою напівпрозору панель у кутку екрану, що активується натисканням клавіші (Tab). На панелі відображається список основних метрик з поточного профілю гравця (currentPlayerProfile з MLContentGenerator), отриманих через GameManager. Метрики представлені у форматі «Назва метрики: Значення %» (наприклад, «aggressionScore: 75%», «explorationScore: 30%»), що забезпечує легкість читання та порівняння.

Неявний інтерфейс взаємодії для гравця:

- **Концептуальна модель.** Для гравця основним «інтерфейсом» взаємодії з системою PCG є сам згенерований контент – рівні, їх структура, складність, розташування елементів. Гравець не взаємодіє з системою безпосередньо, а лише відчуває результати її роботи через

ігровий процес. Модель такої взаємодії базується на принципі невидимої, але відчутної адаптації, що підтримує залученість та оптимальний рівень виклику.

Дизайнерські рішення щодо розробки інтерфейсів були прийняті на основі ергономічних вимог, специфіки цільових користувачів та можливостей обраного інструментарію (Unity).

- Використання Unity Inspector (для розробників):
    - Це стандартний та найбільш звичний спосіб конфігурації компонентів для розробників в Unity [30]. Використання інспектора усуває необхідність створення окремих редакторів чи інструментів, знижує поріг входження та спрощує інтеграцію в існуючі робочі процеси. Публічні поля скриптів автоматично відображаються, що дозволяє швидко налаштовувати параметри.
- Застосування стандартних елементів керування (слайдери, поля):
    - Стандартні елементи забезпечують інтуїтивно зрозумілу взаємодію при налаштуванні числових значень (наприклад, adaptationStrength, difficultyBias) та виборі опцій.
- Використання ScriptableObjects (для конфігурації кластерів/параметрів):
    - Дозволяє відокремити дані конфігурації від логіки, полегшує управління складними наборами даних (як-от параметри для кожного кластера), сприяє перевикористанню та спрощує тестування різних конфігурацій.
- Реалізація PlayerProfileDisplay (для гравця/тестувальника):
    - Надання прозорості щодо роботи внутрішніх механізмів системи є важливим на етапах тестування, налагодження та демонстрації. Відображення профілю у вигляді списку метрик з відсотковими значеннями є простим та інформативним способом показати, як система оцінює гравця. Активація за кнопкою робить його ненав’язливим під час звичайного геймплею.

- Акцент на неявній взаємодії для гравця:
    - Пряме відображення параметрів адаптації або повідомлення гравцю про зміну складності може зруйнувати занурення та викликати негативні реакції [29]. Тому основний фокус робився на плавній та непомітній адаптації самого ігрового контенту, що відповідає вимогам до забезпечення позитивного користувацького досвіду. Інтерфейс для гравця – це сама гра, що динамічно змінюється.

Таким чином, проектування інтерфейсів було спрямоване на забезпечення зручності конфігурації для розробників через стандартні засоби Unity та на створення плавного, неявного досвіду адаптації для гравця, з додатковими інструментами візуалізації для цілей розробки та тестування.

4.3. Принципи забезпечення позитивного досвіду взаємодії гравця з процедурно згенерованим контентом

Процедурна генерація контенту (PCG), і особливо її динамічні та персоналізовані варіанти, відкриває значні можливості для створення різноманітних, реіграбельних та адаптивних ігрових світів. Однак сама по собі технологія генерації не гарантує позитивного досвіду гравця (Player Experience, PX). Неякісно згенерований або погано адаптований контент може призвести до фрустрації, нудьги, втрати занурення та загального негативного враження від гри. Як показав аналіз ергономічних вимог та результати оцінки, забезпечення позитивного PX при використанні PCG, особливо з елементами машинного навчання та адаптації, вимагає свідомого дотримання низки принципів проектування. Ці принципи стосуються як властивостей самого контенту, так і характеру його адаптації (рис. 4.2).

1. **Забезпечення функціональності та іграбельності (Functionality & Playability).** Це базовий, але критично важливий принцип. Будь-який згенерований контент, незалежно від його складності чи новизни, повинен бути функціональним у межах правил гри. Рівні мають бути

прохідними, об’єкти – інтерактивними відповідно до їх призначення, правила – несуперечливими [2]. Для розробленої системи це забезпечувалося на етапі генерації рівнів компонентом LevelGenerator шляхом перевірок на досяжність платформ та наявності шляху (хоча б неявно, через правила розміщення). Відсутність функціональності є однією з найшвидших причин виникнення фрустрації у гравця.

2. **Підтримання узгодженості та правдоподібності (Coherence & Believability).** Згенерований контент повинен відчуватися як органічна частина ігрового світу, відповідати його стилю, тематиці та внутрішній логіці. Використання методів PCGML, навчених на існуючому контенті [3, 4], таких як GAN [5] або VAE [16], може допомогти у відтворенні бажаного стилю. Однак важливо контролювати, щоб генератор не створював елементи, що виглядають абсолютно чужорідними або порушують атмосферу гри. Адаптація також має бути узгодженою: наприклад, зміна складності не повинна призводити до появи елементів, що не відповідають поточній локації чи наративному контексту.
3. **Надання значущої варіативності (Meaningful Variation).** Метою PCG є не просто випадковість, а створення значущої різноманітності, що впливає на геймплей та стимулює інтерес гравця. Згенеровані варіації мають пропонувати нові виклики, тактичні ситуації або можливості для дослідження, а не бути лише косметичними змінами. Система повинна уникати генерації монотонного або передбачуваного контенту, що може швидко набриднути гравцю [1]. Розроблена система прагне до цього шляхом генерації різних типів секцій (лінійних, вертикальних, лабіринтових) та адаптації їх комбінації.
4. **Балансування виклику та навичок (Challenge-Skill Balance / Flow).** Як неодноразово зазначалося [1, 9, 29], підтримання оптимального балансу між складністю викликів та рівнем навичок гравця є ключовим для стану «потоку» та максимальної залученості.

Розроблена адаптивна система безпосередньо спрямована на досягнення цього балансу шляхом аналізу поведінки та налаштування параметрів складності (difficulty, complexity). Важливо, щоб ця адаптація була ефективною, але, як вимагає ергономіка, також плавною та переважно непомітною.

5. **Збереження відчуття контролю та свободи волі гравця (Player Agency & Control).** Гравець повинен відчувати, що саме його дії та рішення визначають успіх у грі. Адаптація, особливо якщо вона занадто агресивна або очевидна, може підірвати це відчуття, створюючи враження, що гра «піддається» або, навпаки, штучно ускладнюється незалежно від зусиль гравця [29]. Тому механізми адаптації мають бути спроектовані так, щоб не знецінювати досягнення гравця та не обмежувати його можливості для демонстрації майстерності чи вибору власного стилю гри.
6. **Керування несподіванкою та новизною (Surprise & Novelty).** PCG може створювати несподівані та нові ситуації, що є одним із джерел задоволення від ігор з процедурним контентом. Однак несподіванки мають бути переважно позитивними або принаймні цікавими з точки зору геймплею, а не просто помилками генератора чи безглуздими артефактами. ML-моделі, особливо генеративні, можуть створювати дійсно новий контент [5], але важливо забезпечити його функціональність та узгодженість.
7. **Сприяння навчанню та майстерності (Learning & Mastery).** Навіть в адаптивній грі гравці повинні мати можливість навчатися її механікам, покращувати свої навички та відчувати прогрес. Адаптація не повинна повністю усувати виклики або перешкоджати процесу навчання. Наприклад, постійне зниження складності для гравця, що зазнає труднощів, може бути контрпродуктивним у довгостроковій перспективі. Система має надавати можливості для подолання труднощів та відчуття зростання майстерності.

8. **Прозорість та пояснюваність (Transparency & Explainability – контекстуально).** У більшості випадків, як зазначалося, адаптація має бути непомітною. Однак у деяких контекстах (навчальні ігри, доступність) або для певних типів гравців може бути корисним надати певну інформацію про те, як і чому гра адаптується. Це може підвищити довіру та розуміння системи. Інтерфейс PlayerProfileDisplay, реалізований у даній роботі, є кроком у цьому напрямку, хоча призначений переважно для розробки та тестування.

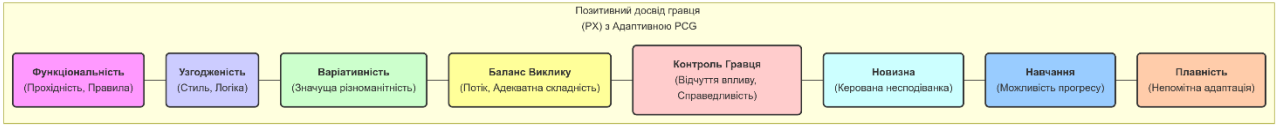

Рисунок 4.2. Принципи забезпечення позитивного досвіду гравця з адаптивною PCG

Дотримання цих принципів вимагає цілісного підходу до проектування, де алгоритми PCG, механізми ML-аналізу та адаптації розглядаються не ізольовано, а в тісному зв’язку з основними ігровими механіками та цілями дизайнерського задуму. Реалізація системи в рамках даної роботи намагалася врахувати ці принципи, зокрема, фокусуючись на забезпеченні функціональності, балансуванні складності через адаптацію, використанні ML для формування профілю та керування параметрами, що впливають на різноманітність та структуру контенту. Постійна оцінка досвіду гравця та ітеративне вдосконалення системи на основі зворотного зв’язку є ключовими для створення процедурно згенерованого контенту, який буде не лише технічно досконалим, але й по-справжньому захоплюючим та задовільним для гравців.

Зв’язок між принципами забезпечення позитивного досвіду гравця та конкретними механізмами й компонентами, реалізованими у розробленій систем представлений у табл. 4.1.

Таблиця 4.1. Відповідність реалізації системи принципам позитивного досвіду гравця

№	Принцип Позитивного Досвіду	Відповідний Механізм/Компонент у Системі	Пояснення
1	Функціональність та Іграбельність	Перевірки в LevelGenerator (EnsureMinimumRequirements, логіка розміщення)	Забезпечується генерація прохідних рівнів з мінімально необхідними елементами.
2	Узгодженість та Правдоподібність	Навчання ML-моделей (приховане), Конфігурація префабів у LevelGenerator	Використання префабів та потенційне навчання ML на даних гри забезпечує стилістичну узгодженість.
3	Значуща Варіативність	Генерація різних типів секцій (LevelGenerator), Адаптація параметрів	Система генерує різні структури рівнів (лінійні, вертикальні, лабіринти) та адаптує їх параметри, створюючи різноманітні виклики.
4	Балансування Виклику та Навичок (Потік)	MLContentGenerator (аналіз профілю, адаптація difficulty, complexity)	Основна функція системи – аналіз skillLevel та інших метрик для підтримання оптимальної складності.
5	Відчуття Контролю та Свободи Волі Гравця	Плавна адаптація (InterpolateParameters), Відсутність різких змін	Інтерполяція параметрів та помірна швидкість адаптації (adaptationRate) спрямовані на те, щоб гравець не відчував штучного втручання.
6	Керування Несподіванкою та Новизною	Використання ймовірностей у LevelGenerator, потенціал ML-моделей	Випадковість у виборі секцій та розміщенні елементів, а також потенціал ML генерувати нетипові комбінації параметрів створюють новизну.
7	Сприяння Навчанню та Майстерності	Адаптивне підвищення складності (при зростанні skillLevel)	Система не лише спрощує гру, але й підвищує складність при покращенні навичок гравця, стимулюючи подальший розвиток.
8	Прозорість та Пояснюваність (контекст.)	PlayerProfileDisplay (для налагодження)	Хоча адаптація для гравця неявна, для розробки та тестування передбачено інструмент візуалізації профілю, що підвищує прозорість системи.

Висновки до розділу 4

Таким чином, четвертий розділ присвячений ергономічним аспектам розробленої системи персоналізованої процедурної генерації контенту. У розділі розглядаються ключові ергономічні вимоги як до інтерфейсу конфігурації для розробників, що стосуються простоти інтеграції, інтуїтивності та контролю, так і до процесу динамічної адаптації з точки зору гравця, де наголошується на плавності, справедливості та релевантності змін. Представлено проектування відповідних інтерфейсів, зокрема використання інспектора Unity для розробників та інформаційної панелі профілю гравця для тестування, обґрунтовуючи концепцію неявної взаємодії гравця з адаптивним контентом. Наведено результати ергономічної оцінки системи, отримані шляхом експертного аналізу та інтерпретації даних користувацького досвіду, що дозволило сформулювати рекомендації щодо вдосконалення. На завершення, розділ формулює узагальнені принципи проектування адаптивних систем PCG, які спрямовані на забезпечення позитивного досвіду гравця, включаючи функціональність, узгодженість, баланс виклику та збереження контролю.

ЗАГАЛЬНІ ВИСНОВКИ

У даній кваліфікаційній роботі було вирішено актуальну науково-практичну задачу розробки та дослідження методів і архітектури системи для динамічної та персоналізованої процедурної генерації ігрового контенту на основі машинного навчання. Метою роботи було створення системи, здатної адаптувати характеристики ігрових рівнів відповідно до профілю та поведінки гравця для підвищення якості ігрового досвіду.

Для досягнення поставленої мети було вирішено наступні основні завдання:

1. Проведено комплексний аналітичний огляд сучасного стану процедурної генерації контенту (PCG), що охопив еволюцію методів, аналіз традиційних алгоритмічних підходів (генератори шуму, клітинні автомати, L-системи, методи на основі графів) та їхніх обмежень. Детально розглянуто застосування методів машинного навчання (PCGML), зокрема генеративних моделей (GAN, VAE), послідовних моделей (RNN/LSTM, трансформери), генетичних алгоритмів та навчання з підкріпленням (RL) у контексті генерації ігрового контенту. Проаналізовано ключові проблеми та підходи до персоналізації та динамічної адаптації ігрового досвіду, включаючи моделювання гравця та методи адаптивного балансування. Результати огляду підтвердили актуальність теми та дозволили чітко сформулювати проблему дослідження – необхідність створення інтегрованих систем персоналізованої PCG на основі ML.
2. Спроектовано систему динамічної та персоналізованої PCG. Було обґрунтовано вибір методів машинного навчання: комбінацію керованого та некерованого навчання для моделювання гравця; гібридні підходи (поєднання генеративних моделей з послідовними або RL) для генерації контенту; RL або умовну генерацію для механізму адаптації. Здійснено формалізацію задачі шляхом розробки

математичних моделей контенту, процесу генерації, моделі гравця та функцій адаптації. Спроектовано модульну архітектуру системи, що включає компоненти збору даних, аналізу та моделювання гравця, прийняття рішень щодо адаптації, генерації контенту та інтерфейс інтеграції. Також було спроектовано ключові алгоритми функціонування модулів та інтерфейси для взаємодії з ігровими рушіями.

3. Здійснено програмну реалізацію прототипу спроектованої системи в середовищі Unity з використанням мови C#, бібліотеки Barracuda для виконання нейромережевих моделей у форматі ONNX та Python для офлайн-навчання моделей. Реалізовано ключові компоненти архітектури, включаючи детальний логер поведінки гравця, ML-модуль для аналізу профілю та визначення параметрів, а також адаптивний генератор рівнів. Проведено технічне тестування (модульне, інтеграційне, продуктивності), що підтвердило коректність роботи системи та її прийнятну продуктивність. Розроблено дизайн та проведено експериментальні дослідження з користувачами для оцінки впливу системи на ігровий досвід. Статистичний аналіз результатів показав позитивний ефект адаптації на задоволення, залученість гравців та адекватність сприйняття складності порівняно з неадаптивною системою.
4. Досліджено ергономічні аспекти розробленої системи. Визначено ергономічні вимоги до інтерфейсу конфігурації для розробників (простота, інтуїтивність, моніторинг, контроль) та до процесу динамічної адаптації з точки зору гравця (непомітність, справедливість, доцільність, узгодженість). Описано проектування реалізованих інтерфейсів (інспектор Unity, візуалізатор профілю). Проведено ергономічну оцінку за допомогою експертних методів та аналізу даних користувацького тестування, що дозволило виявити сильні сторони та сформулювати рекомендації щодо покращення

інтерфейсів та механізмів адаптації. Сформульовано загальні принципи забезпечення позитивного досвіду взаємодії гравця з адаптивним процедурно згенерованим контентом.

Таким чином, усі поставлені завдання кваліфікаційної роботи було виконано. Мета роботи – розробка та дослідження системи динамічної та персоналізованої PCG на основі ML – досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yannakakis G. N., Togelius J. Artificial Intelligence and Games. Springer, 2018. 337 p.
2. Shaker N., Togelius J., Nelson M. J. Procedural Content Generation in Games. Springer, 2016. 242 p.
3. Summerville A., Snodgrass S., Guzdial M., Holmgård C., Hoover A. K., Isaksen A., Togelius J. Procedural content generation via machine learning (PCGML). IEEE Transactions on Games. 2018. Vol. 10, No. 3. P. 257–270.
4. Liu J., Snodgrass S., Khalifa A., Risi S., Yannakakis G. N., Togelius J. Deep learning for procedural content generation. Neural Computing and Applications. 2021. Vol. 33, No. 1. P. 19–37.
5. Volz V., Schrum J., Liu J., Lucas S. M., Smith A., Risi S. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. Proceedings of the Genetic and Evolutionary Computation Conference. 2018. P. 221–228.
6. Kingma D. P., Welling M. Auto-encoding variational bayes. 2013. URL: https://arxiv.org/abs/1312.6114 (дата звернення: 10.04.2025).
7. Khalifa A., Green M. C., Pérez-Liébana D., Togelius J. PCGRL: Parameterized complex environment generation using reinforcement learning. 2020. URL: https://arxiv.org/abs/2001.09212 (дата звернення: 10.04.2025).
8. Drachen A., Canossa A., Yannakakis G. N. Player modeling using self-organization in Tomb Raider: Underworld. IEEE Symposium on Computational Intelligence and Games. 2009. P. 1–8.
9. Holmgård C., Liapis A., Togelius J., Yannakakis G. N. Automated playtesting with procedural personas based on interaction traces. IEEE Transactions on Games. 2018. Vol. 10, No. 3. P. 230–241.

10. Pedersen M. T., Togelius J., Yannakakis G. N. Modeling player experience in Super Mario Bros. IEEE Symposium on Computational Intelligence and Games. 2009. P. 132–139.
11. Койбічук В., Кочережченко Р., Гриценко К., Яценко В., Єфіменко А. Алгоритми процедурної генерації ігрового контенту за допомогою графів. Information Technology: Computer Science, Software Engineering and Cyber Security. 2024. № 3. С. 77–87. URL: https://doi.org/10.32782/IT/2024-3-8 (дата звернення: 10.04.2025).
12. Марчук Г. В., Левківський В. Л., Харченко А. В., Марчук Д. К. Огляд і аналіз алгоритмів процедурної генерації ігрових світів. Вісник Національного технічного університету «ХПІ». Серія: Нові рішення в сучасних технологіях. 2024. № 4(18). С. 101–109. URL: https://doi.org/10.32782/tnv-tech.2024.4.9 (дата звернення: 10.04.2025).
13. Laitaruk I. F., Hryshanovych T. O. Overview of modern algorithms for world procedural generation in computer games. Proceedings of the 7th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2024). 2024. P. 152–163.
14. Hendrikx M., Meijer S., Van Der Velden J., Iosup A. Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM). 2013. Vol. 9, No. 1. P. 1–22.
15. Perlin K. An image synthesizer. ACM SIGGRAPH Computer Graphics. 1985. Vol. 19, No. 3. P. 287–296.
16. Sarkar A., Yang Z., Cooper S. Generating Levels Like Historical Games: Exploring VAEs for Level Generation. IEEE Conference on Games (CoG). 2021. P. 1–8.
17. Summerville A., Mateas M. Super Mario as a string: Platformer level generation via LSTMs. Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference. 2016.

18. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N. та ін. Attention is all you need. Advances in Neural Information Processing Systems. 2017. Vol. 30.
19. Prusinkiewicz P., Lindenmayer A. The Algorithmic Beauty of Plants. Springer-Verlag, 1990. 228 p.
20. Nielsen J. Heuristic evaluation. Nielsen J., Mack R. L. (Eds.). Usability Inspection Methods. John Wiley & Sons, 1994.
21. Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S. та ін. Generative adversarial nets. Advances in Neural Information Processing Systems. 2014. Vol. 27.
22. Koesnaedi A., Istiono W. Implementation drunkard's walk algorithm to generate random level in roguelike games. International Journal of Multidisciplinary Research and Publications. 2022. Vol. 5, No. 2. P. 97–103.
23. Cóth C. D. Binary space partitions: recent developments. Combinatorial and Computational Geometry. 2005. Vol. 52. P. 525–552.
24. Ryan J. S., Cowling P., Walker J. A. Procedural generation using spatial GANs for region-specific learning of elevation data. IEEE Conference on Games (CoG). 2019.
25. Hochreiter S., Schmidhuber J. Long short-term memory. Neural Computation. 1997. Vol. 9, No. 8. P. 1735–1780.
26. Roemmele M. Story Validation for Neural Story Generation Models. 2019. URL: https://arxiv.org/abs/1909.05397 (дата звернення: 10.04.2025).
27. Soares de Lima E., Feijó B., Furtado A. L. Procedural generation of quests for games using genetic algorithms and automated planning. 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). Rio de Janeiro, Brazil, 2019. P. 144–153. DOI: https://doi.org/10.1109/SBGames.2019.00028 (дата звернення: 10.04.2025).
28. Sutton R. S., Barto A. G. Reinforcement learning: An introduction. MIT Press, 2018. 552 p.

29. Hunicke R. The case for dynamic difficulty adjustment in games. Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology. 2005. P. 429–433.
30. Unity User Manual. Unity Technologies. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 10.04.2025).
31. Barracuda Package Documentation. Unity Technologies. URL: https://docs.unity3d.com/Packages/com.unity.barracuda@latest/ (дата звернення: 10.04.2025).
32. ONNX (Open Neural Network Exchange). URL: https://onnx.ai/onnx/ (дата звернення: 10.04.2025).
33. TensorFlow Core API Documentation. URL: https://www.tensorflow.org/api_docs (дата звернення: 10.04.2025).
34. PyTorch Documentation. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 10.04.2025).
35. Scikit-learn User Guide. Scikit-learn development team. URL: https://scikit-learn.org/stable/documentation.html (дата звернення: 10.04.2025).