УДК 004.9:004.421.2 DOI https://doi.org/10.35546/kntu2078-4481.2025.4.3.12

Ю. Ю. ІЛЯШ
кандидат технічних наук, доцент,
завідувач кафедри комп'ютерних наук та інформаційних систем
Карпатський національний університет імені Василя Стефаника
ORCID: 0009-0006-5786-7205

В. О. ГОРЄЛОВ
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук та інформаційних систем
Карпатський національний університет імені Василя Стефаника
ORCID: 0000-0002-2106-8704

В. А. РОВІНСЬКИЙ
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук та інформаційних систем
Карпатський національний університет імені Василя Стефаника
ORCID: 0000-0001-8454-8580

АЛГОРИТМИ ТА МЕТОДИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ТА АДАПТИВНОЇ СКЛАДНОСТІ РІВНІВ В ІГРАХ-ПЛАТФОРМЕРАХ

Стаття присвячена дослідженню алгоритмів та методів процедурної генерації ігрового контенту й адаптивної складності в 2D-платформерах. Обґрунтовано актуальність підходів, що поєднують модульну архітектуру з динамічним налаштуванням складності відповідно до дій гравця, щоб забезпечити варіативність, повторюваність проходження та підтримку динамічності ігрового контенту. У роботі запропоновано пошарову (layered) модель проєктування рівнів, у якій послідовно формуються базові елементи середовища (ґрунт, платформи, перешкоди), а надбудови шарів (вороги, пастки, декорації) керуються узгодженими правилами розміщення. Ключову роль у створенні природної геометрії та керованої випадковості відіграють шум Перліна та його фрактальні розширення (octaves): комбінування кількох частот і амплітуд дає змогу поєднати великомасштабні контури з дрібними деталями рельєфу. Запропонована система адаптивної складності інтегрується у генератор шляхом зміни параметрів шуму (scale, persistence, octaves) і поведінкових характеристик ворогів (швидкість, частота атак, кількість), що визначаються індексом продуктивності гравця. Практичну реалізацію виконано у рушії Unity з використанням мови програмування C# та Tilemap-підходу; наведено приклади коду для побудови карти висот і підключення контролера адаптації. Розроблені алгоритми та методи є універсальними й можуть бути легко інтегровані в середовища C++, зокрема для рушіїв Unreal Engine та Cocos2d-x, що підтверджує їхню практичну придатність і міжплатформність. Показано, що такий комбінований підхід зберігає баланс між непередбачуваністю та керованістю, зменшує витрати на ручний дизайн рівнів і підвищує залучення користувачів завдяки персоналізованому ігровому досвіду. Отримані результати можуть бути застосовані у навчальних прототипах, інді-проєктах та як базис для подальших досліджень із використанням методів машинного навчання в генеративному геймдизайні.

Ключові слова: *процедурна генерація; шум Перліна; фрактальний шум; адаптивна складність; 2D-платформери; Unity; C++.*

YU. YU. ILIASH
Candidate of Technical Sciences, Associate Professor,
Head of the Department of Computer Science
Vasyl Stefanyk Carpathian National University
ORCID: 0009-0006-5786-7205

V. O. HORIELOV
Candidate of Technical Sciences, Associate Professor,
Associate Professor at the Department of Computer Science
Vasyl Stefanyk Carpathian National University
ORCID: 0000-0002-2106-8704

V. A. ROVINSKY
Candidate of Technical Sciences, Associate Professor,
Associate Professor at the Department of Computer Science
Vasyl Stefanyk Carpathian National University
ORCID: 0000-0001-8454-8580

ALGORITHMS AND METHODS OF PROCEDURAL GENERATION AND ADAPTIVE DIFFICULTY IN PLATFORMER GAMES

The article focuses on the study of algorithms and methods for procedural generation of game content and adaptive difficulty in 2D platformers. The relevance of approaches combining modular architecture with dynamic difficulty adjustment based on player actions is substantiated, aiming to ensure variability, replay value, and maintain the dynamism of game content. The paper proposes a layered model of level design, where basic environmental elements (ground, platforms, obstacles) are generated sequentially, while higher layers (enemies, traps, decorations) are governed by consistent placement rules. A key role in creating natural geometry and controlled randomness is played by Perlin noise and its fractal extensions (octaves): combining multiple frequencies and amplitudes allows large-scale contours to merge with fine terrain details. The proposed adaptive difficulty system is integrated into the generator by modifying noise parameters (scale, persistence, octaves) and enemy behavioral attributes (speed, attack frequency, quantity), determined by the player's performance index. The practical implementation was carried out in the Unity engine using the C# programming language and the Tilemap approach; code examples for heightmap generation and adaptive controller integration are provided. The developed algorithms and methods are engine-agnostic and can be easily integrated into C++ environments, particularly Unreal Engine and Cocos2d-x, demonstrating their cross-platform applicability and scalability. It is shown that the combined approach maintains a balance between unpredictability and control, reduces the cost of manual level design, and enhances user engagement through personalized gameplay experiences. The obtained results can be applied in educational prototypes, indie projects, and serve as a foundation for further research involving machine learning methods in generative game design.

Key words: *procedural generation; Perlin noise; fractal noise; adaptive difficulty; 2D platformers; Unity; C++.*

Постановка проблеми

У XXI столітті індустрія відеоігор перетворилася на один із найдинамічніших напрямів цифрової культури та інновацій. Вона об'єднує мистецтво, програмування, психологію, моделювання поведінки й системи штучного інтелекту. Завдяки широкій доступності рушіїв, таких як *Unity*, *Unreal Engine*, *Cocos2d-x*, створення власних ігор стало можливим не лише для великих компаній, а й для незалежних розробників і студентів-ентузіастів.

Серед різних жанрів особливої уваги заслуговують 2D-платформери – ігри, що поєднують динаміку, простоту керування та високий потенціал для творчих експериментів. Вони є відмінним полігоном для вивчення основ ігрового дизайну, алгоритмічних методів і концепцій адаптивних систем.

Сучасні тенденції розвитку ігор засвідчують, що користувачі очікують не лише красивої графіки чи захопливого сюжету, а насамперед – варіативності та неповторності ігрового досвіду. Гравець прагне, щоб кожне нове проходження було унікальним, створюючи ефект «живого світу», який розвивається самостійно. Одним із найефективніших шляхів досягнення цього є процедурна генерація контенту (Procedural Content Generation, PCG) – автоматичне створення рівнів, об'єктів, перешкод чи навіть наративів за допомогою алгоритмів. Такі технології не лише знижують витрати часу на розробку, а й дозволяють створювати фактично нескінченні варіації ігрового середовища.

Паралельно актуальним напрямом стає адаптивна складність (Dynamic Difficulty Adjustment, DDA), мета якої – підтримувати баланс між викликом і комфортом, динамічно реагуючи на дії, рівень майстерності та стиль гри користувача. Поєднання процедурної генерації з адаптивними механізмами відкриває можливість створення гнучких систем, здатних забезпечити персоналізований досвід і підтримувати мотивацію гравця протягом усього процесу гри.

Аналіз останніх досліджень і публікацій

Питання оптимізації ігрових даних та ефективної обробки інформації розглядалися у працях українських і зарубіжних авторів. Зокрема в [1], було запропоновано методи ущільнення та структурного перетворення інформації, що підвищують ефективність обробки й передачі даних у реальному часі. Ідеї оптимізації структури ігрового контенту, висвітлені в цій статті, стали основою для подальших досліджень алгоритмів динамічного формування середовища та адаптивної складності рівнів.

Поняття процедурної генерації відоме ще з 1980-х років, проте саме сучасні обчислювальні можливості дали змогу піднести цей підхід на рівень складних систем зі штучним інтелектом, здатних реагувати на дії гравця [2].

Водночас не менш важливою тенденцією є поява адаптивних систем складності [3]. Їхня суть полягає в тому, щоб гра не залишалася статичною, а реагувала на успішність, стиль гри та емоційний стан користувача. Гравець, який грає впевнено, може стикатися з більш жорсткими викликами, тоді як початківець – із помірними, щоб

уникнути втрати мотивації. Перші концепції Динамічного Регулювання Складності описані Робіном Ханіком у класичній праці [4], а подальші дослідження [3, 5] доводять ефективність таких систем у збереженні стану "flow" – оптимального балансу між викликом і навичками.

Поєднання процедурної генерації з адаптивною складністю є одним із найперспективніших напрямів у сучасному геймдизайні. Такий підхід дозволяє не лише створювати унікальні рівні, але й налаштовувати їхню складність у процесі гри, забезпечуючи персоналізований досвід для кожного користувача. Наприклад, дослідження [6] демонструє, як алгоритм може генерувати рівні, виходячи з показників успішності гравця – тривалості життя, частоти помилок, часу реакції тощо. Це дозволяє створювати динамічну криву складності без ручного налаштування сценаріїв.

Особливо актуальним застосування цих технологій є саме для жанру платформерів. На відміну від стратегій або RPG, де складність часто визначається кількістю ресурсів чи ворогів, у платформерах вона прямо пов'язана з геометрією рівня, розміщенням платформ, глибинами ям, швидкістю ворогів і частотою перешкод. Саме тому автоматизована генерація рівнів із контролем складності має практичну цінність – вона дозволяє масштабувати контент, створювати адаптивні виклики та уникати монотонності. Застосування шумових, клітинних автоматів, граматичних систем, еволюційних і пошукових алгоритмів стало стандартом у дослідженнях цієї тематики [7, 8]

Із теоретичного погляду, процедурна генерація та адаптивна складність поєднують у собі елементи класичних алгоритмів, стохастичних процесів і методів машинного навчання. Зокрема, пошукові підходи (search-based PCG) використовують еволюційні алгоритми для пошуку найкращої конфігурації рівня відповідно до функції оцінювання. Граматичні методи будують рівні на основі формальних правил, подібно до синтаксичного аналізу в мовах програмування. А сучасні підходи, засновані на підкріплювальному навчанні (Reinforcement Learning), дозволяють навча́ти агента-генератора створювати контент, який оптимально відповідає стилю гри користувача [9, 10].

Практична цінність дослідження полягає в тому, що впровадження подібних алгоритмів у реальні рушії (як-от Unity, Unreal Egine або Cocos2d-x) відкриває можливість створювати ігри, що «живуть» разом із гравцем. Кожне нове проходження формує власний маршрут, власний набір викликів і вражень. Це не лише підвищує залученість користувачів, але й формує основу для розвитку адаптивних навчальних і симуляційних систем, де гра може виступати моделлю навчального середовища.

Попри численні переваги, використання процедурної генерації та адаптивної складності пов'язане з рядом викликів. Одним із них є оцінка рівня «якісності» згенерованого контенту – адже алгоритм може створити рівень, який технічно коректний, але ігрово незбалансований. Іншим викликом є питання «відчуття контролю»: якщо гравець помічає, що гра сама підлаштовується під нього, це може знизити відчуття чесного виклику. Тому важливим напрямом є розробка систем, які залишають адаптацію непомітною, зберігаючи природність процесу.

Формулювання мети дослідження

Таким чином, розвиток алгоритмів і методів процедурної генерації та адаптивної складності рівнів в іграх-платформерах є одним із найбільш перспективних напрямів сучасної ігрової розробки. Він поєднує математичні моделі, аналітику поведінки, штучний інтелект і креативний дизайн, відкриваючи нові можливості як для наукових досліджень, так і для індустріальної практики. У цьому контексті створення прототипу 2D-платформера з процедурною генерацією рівнів і адаптивною складністю може стати не лише демонстрацією теоретичних принципів, а й практичним внеском у розвиток технологій інтелектуального геймдизайну.

Викладення основного матеріалу дослідження

Під час розробки гри-платформера з процедурною генерацією рівнів особливу увагу приділяється створенню збалансованого ігрового середовища, яке поєднує зручність керування, гнучкість механік, доступність для широкого кола гравців і здатність адаптуватися до різних стилів проходження. Врахування потреб як новачків, так і досвідчених користувачів дозволяє побудувати динамічну систему складності, що підтримує природний темп гри та зберігає інтерес протягом усього процесу.

Пропонується реалізовувати процес проєктування на основі пошарового підходу до процедурної генерації рівнів, який розглядається як один з етапів архітектурного проєктування ігрового контенту. Його сутність полягає у послідовному формуванні середовища – від базових елементів (ґрунту, платформ, зон пересування) до компонентів вищого рівня (ворогів, пасток, декоративних об'єктів). Така структура дозволяє забезпечити логічну впорядкованість, контроль над взаємодією об'єктів і запобігання конфліктам при розміщенні.

Запропонована багаторівнева модель генерації характеризується передбачуваністю, гнучкістю та масштабованістю. Вона спрощує налагодження, адже помилки легко локалізуються в межах окремих шарів, і дозволяє розширювати гру шляхом додавання нових модулів – наприклад, шарів «скарбів», погодних ефектів чи декоративних об'єктів. Крім того, модель інтегрує механізми адаптивної складності, які динамічно змінюють параметри генерації залежно від дій гравця: система може збільшувати кількість ворогів або пасток при швидкому проходженні, чи навпаки, спрощувати геометрію рівня при численних помилках. Саме на цьому етапі доцільним стає використання шумових функцій, зокрема шуму Перліна, що забезпечує природність, когерентність і варіативність ігрового простору, створюючи плавний перехід до наступного етапу генерації – побудови шумових карт рівня.

Одним із ключових інструментів, що використовується для створення природних, плавних і варіативних ігрових середовищ, є шум Перліна – математична функція, розроблена Кеном Перліном у 1983 році [11]. Її основна ідея полягає у створенні когерентного (узгодженого) шуму, тобто такого, у якому сусідні значення мають локальну кореляцію та змінюються плавно, на відміну від випадкових (білих) шумів, де кожна точка не залежить від сусідніх. Саме ця властивість робить шум Перліна надзвичайно корисним для задач комп'ютерної графіки, процедурної генерації ландшафтів, текстур, хмар, морських хвиль, а також для побудови рівнів у 2D та 3D іграх.

Якщо випадкові числа між 0 та 1 застосувати безпосередньо для створення карти висот, отримаємо так званий білий шум, у якому відсутня структура – сусідні точки змінюються різко й хаотично. (рис. 1) Такий результат виглядає неприродньо. Натомість шум Перліна формує «плавні» хвилеподібні структури, які око людини сприймає як природні візерунки – пагорби, долини, хмари або поверхню землі (рис. 2). Це досягається завдяки математичній побудові функції, у якій випадковість поєднана з локальною згладженістю.

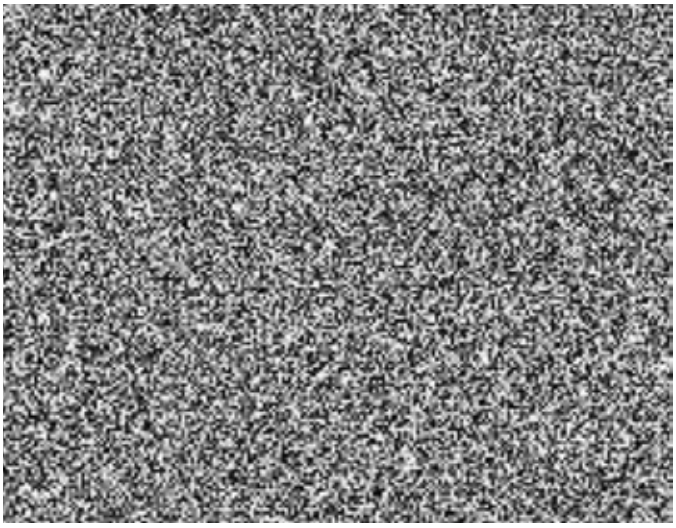

Рис. 1. Білий шум

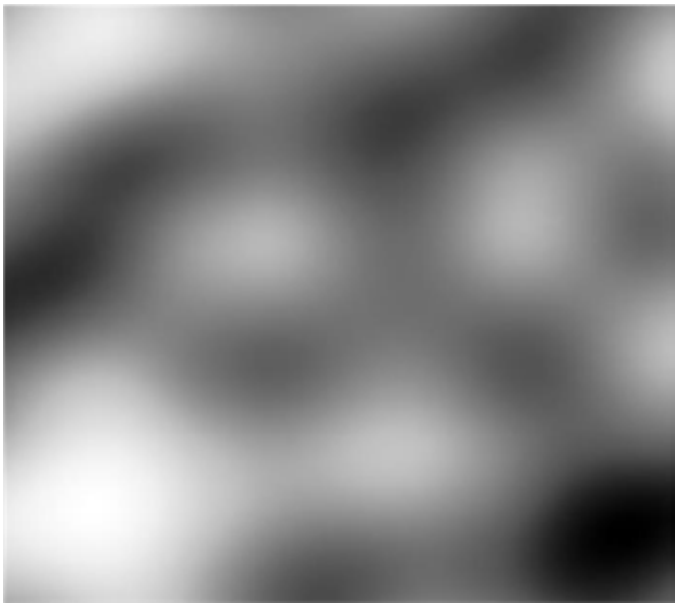

Рис. 2. Шум Перліна

Формально одномірний шум Перліна описується як зважена сума локальних градієнтів:

$$P(x) = \sum \text{fade}(t_i) \cdot G_i$$

де:
x – координата,
i – індекс решітки навколо x,
G_i – випадковий напрямок градієнта в точці решітки i,
$t_i = x - i$ – локальне зміщення від точки решітки,
fade(t) – функція згладжування, що забезпечує плавні переходи (зазвичай $fade(t) = 6t^5 - 15t^4 + 10t^3$).

Завдяки функції fade() отримується неперервне значення, а градієнти вносять контрольовану варіативність. Результатом є функція, що виглядає випадково, але зберігає локальну гармонію – саме це і є когерентний шум.

Шум Перліна має низку переваг, що роблять його одним із найпоширеніших інструментів у процедурній генерації контенту. Його головною особливістю є плавність і неперервність, адже значення змінюються поступово, без різких переходів, завдяки чому створюються природні форми й структури. Водночас шум Перліна є детермінованим – при однакових вхідних параметрах він завжди дає ідентичний результат, що дозволяє відтворювати однакові рівні або ландшафти. Ще однією перевагою є контрольована випадковість: отримані патерни виглядають природно, не маючи явно вираженої періодичності. Крім того, шум відзначається масштабованістю – його можна використовувати з різними частотами (scale), поєднуючи кілька рівнів для формування складних фрактальних структур (octaves). Завдяки цим властивостям шум Перліна широко застосовується для генерації ландшафтів і визначення висот поверхні, розподілу води, ресурсів чи біомів, створення варіативності ігрових об'єктів за формою, розміром або кольором, а також для імітації природних текстур – таких як хмари, трава чи камінь.

У ігрових рушіях реалізація шуму Перліна доступна через вбудовану функцію (код на Unity):

```
float height = Mathf.PerlinNoise(x * scale, y * scale);  Unity
float Height = FMath::PerlinNoise2D(FVector2D(X * Scale, Y * Scale)) * MaxHeight;  Unreal Engine
```

Параметр scale визначає частоту шуму: менше значення дає плавні великі хвилі, більше – дрібні нерівності. Отримане значення height (у діапазоні [0, 1]) використовується для обчислення висоти поверхні або для прийняття рішення, який тип блоку розмістити у даній точці – землю, воду, платформу чи порожній простір.

Наприклад (код на Unity):

```
float height = Mathf.PerlinNoise(x * 0.1f, seed) * maxHeight;
int y = Mathf.FloorToInt(height);
```

Таким чином, для кожної координати x визначається висота y, що утворює плавний рельєф рівня. Комбінування декількох шарів шуму з різними параметрами частоти та амплітуди дозволяє створювати складніші топографії – від м'яких пагорбів до хаотичних скельних структур.

Використання шуму Перліна у системі пошарової генерації рівнів забезпечує природність форм, варіативність розміщення об'єктів і підґрунтя для подальшої адаптації складності, адже зміна параметрів scale чи maxHeight дає можливість керувати «рельєфом» рівня, створюючи плавні або навпаки – динамічні, складні для проходження ділянки.

Попри свою гнучкість, класичний шум Перліна має обмеження: при використанні одного рівня частоти та амплітуди отримана карта має надто рівномірну структуру – великі плавні хвилі без дрібних деталей. Щоб подолати цю проблему, у практиці комп'ютерної графіки та процедурного моделювання використовується фрактальний шум [12], або шум із «октавами» (octaves), який є комбінацією кількох рівнів шуму Перліна з різними масштабами.

Фрактальний шум можна подати як суму кількох шарів (octaves) базового шуму, кожен із яких має власну частоту (масштаб) і амплітуду (вплив на результат):

$$F(x,y) = \sum_{i=0}^{n-1} amp_i \cdot PerlinNoise\left(x \cdot freq_i, y \cdot freq_i\right)$$

де:
n – кількість шарів,
$freq_i = baseFreq \cdot 2^i$ – частота для i-го шару,
$amp_i = baseAmp \cdot persistence^i$ – амплітуда, що зменшується зі зростанням i,
$persistence \in [0,1]$ – коефіцієнт зменшення амплітуди (зазвичай 0.5–0.7).

Таким чином, перший шар створює великі загальні контури рельєфу, а наступні додають дрібні деталі, текстуру та варіативність. Результат – природній шум, що поєднує плавні макроструктури та дрібномасштабні особливості. Цей принцип аналогічний побудові фракталів, звідки походить і сама назва методу.

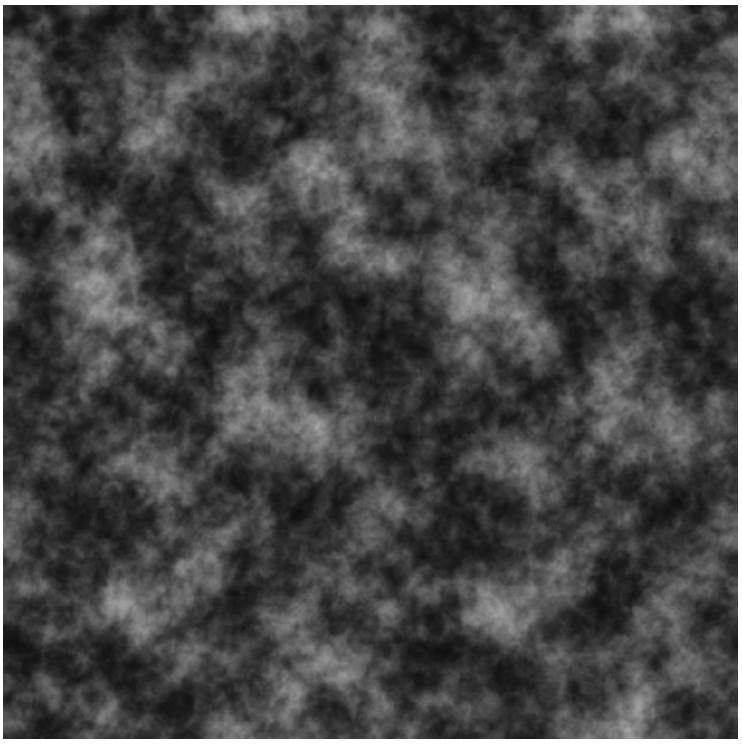

Рис. 3. Фрактальний шум Перліна

У рушії Unity фрактальний шум легко реалізується шляхом сумування кількох викликів Mathf.PerlinNoise() з різними масштабами:

```
float FractalNoise(float x, float y, int octaves, float persistence, float scale)
{
    float total = 0f;
    float frequency = 1f;
    float amplitude = 1f;
    float maxValue = 0f;  // для нормалізації

    for (int i = 0; i < octaves; i++)
    {
          total += Mathf.PerlinNoise(x * frequency * scale, y * frequency * scale) *
amplitude;
        maxValue += amplitude;
        amplitude *= persistence;
        frequency *= 2f;
    }
    return total / maxValue;
}
```

У межах цього дослідження запропоновано новий гібридний підхід до генерації рівнів, який поєднує три ключові принципи:

- **Пошарова архітектура**: забезпечує структуровану, керовану логіку створення рівня – від базової землі до ворогів і декоративних елементів.
- **Фрактальний шум Перліна**: задає основу для кожного шару, формуючи природні переходи між ділянками та визначаючи топографію рівня з різною деталізацією.
- **Адаптивна складність**: динамічно модифікує параметри шуму (наприклад, scale, persistence, кількість octaves) на основі дій гравця – підвищуючи або знижуючи частоту нерівностей, довжину платформ чи щільність пасток.

Таке поєднання дозволяє створити самоадаптивний генератор рівнів, який не лише формує різноманітне середовище, але й регулює складність у реальному часі. Наприклад:

- при швидкому проходженні рівня гра збільшує параметр frequency, створюючи щільнішу структуру платформ і складніші перепади;
- при частих поразках – зменшує амплітуду шуму, роблячи поверхню рівнішою і простішою для навігації.

Завдяки цьому система отримує властивість реактивного дизайну (responsive design), коли рівень не є статично заданим, а формується у взаємодії з гравцем. Це не лише підвищує реіграбельність, але й наближає гру до концепції «інтерактивної творчості», описаної в [13], де користувач виступає співтворцем процесу.

Таким чином, фрактальний шум Перліна у поєднанні з пошаровою архітектурою та адаптивною логікою складності формує новий підхід до процедурної генерації у 2D-платформерах. Він дозволяє створювати природні, непередбачувані, але керовані рівні, що змінюються залежно від досвіду, швидкості та стилю гри користувача. Така система поєднує алгоритмічну строгість з елементами динамічної творчості, що відповідає сучасним тенденціям у галузі інтелектуального геймдизайну.

З урахуванням теоретичних принципів процедурної генерації та фрактального шуму Перліна, реалізовано комплексний алгоритм побудови рівня у середовищі Unity. Його основою є поєднання пошарового підходу з керованими параметрами шуму, що дозволяє створювати унікальні ігрові локації під час кожного запуску гри.

Архітектура генерації побудована за модульним принципом, де кожен компонент відповідає за окрему стадію побудови рівня. Кожен етап працює незалежно, використовуючи спільні параметри, що передаються через конфігураційний об’єкт LevelSettings. Такий підхід забезпечує легку масштабованість і можливість заміни будь-якого модуля без впливу на решту системи.

Основні модулі системи:

NoiseGenerator: обчислює карту висот за допомогою фрактального шуму Перліна (використовуються параметри scale, persistence, octaves, seed).

TerrainBuilder: перетворює карту висот у базову топографію рівня (ґрунт, платформи, порожні ділянки).

ObstaclePlacer: додає перешкоди (шипи, ями, пастки) згідно з логікою пошарової побудови.

EnemySpawner: розміщує ворогів на рівних поверхнях з урахуванням складності гри.

AdaptiveController: аналізує поведінку гравця (кількість смертей, швидкість проходження, середній час реакції) і змінює параметри генерації для наступного рівня.

Нижче наведено спрощений псевдокод алгоритму генерації рівня:

```
function GenerateLevel(seed, difficulty):
    setParametersBasedOnDifficulty(difficulty)
    heightMap = FractalPerlinNoise(seed, scale, persistence, octaves)

    for x in range(levelWidth):
        height = heightMap[x]
        for y in range(levelHeight):
            if y <= height:
                placeBlock("ground", x, y)
            else if y <= waterLevel:
                placeBlock("water", x, y)

    placePlatforms(heightMap)
    placeSpikes(heightMap)
    placeEnemies(heightMap, difficulty)
```

Під час виконання гри, після завершення рівня, система передає дані про продуктивність гравця модулю AdaptiveController, який коригує складність:

```
function UpdateDifficulty(playerStats):
    if playerStats.deaths > threshold:
        scale -= 0.02      // зменшення хвилястості рельєфу
        octaves -= 1       // менше дрібних деталей
    else if playerStats.speed > avgSpeed:
        scale += 0.02      // дрібніші коливання поверхні
        octaves += 1       // більше деталей
```

Таким чином, кожен наступний рівень формується з урахуванням попередніх дій гравця. Це створює ефект еволюції складності, коли гра динамічно підлаштовується до користувача, не потребуючи фіксованих налаштувань.

У практичній реалізації в Unity використовується Tilemap-система, що дозволяє візуалізувати згенеровані блоки у вигляді спрайтів. Основний скрипт LevelGenerator.cs виконує ініціалізацію шуму та побудову ландшафту:

```
for (int x = 0; x < width; x++)
{
    float height = FractalNoise(x, seed, octaves, persistence, scale) * maxHeight;
    for (int y = 0; y < height; y++)
        tilemap.SetTile(new Vector3Int(x, y, 0), groundTile);

    if (height < waterThreshold)
        tilemap.SetTile(new Vector3Int(x, Mathf.FloorToInt(height), 0), waterTile);
}
Модуль AdaptiveController.cs періодично оновлює параметри генератора:
if (player.deaths > 3)
{
    generator.scale *= 0.95f;
    generator.octaves = Mathf.Max(2, generator.octaves - 1);
}
else if (player.score > 1000)
{
    generator.scale *= 1.05f;
    generator.octaves = Mathf.Min(8, generator.octaves + 1);
}
```

У результаті, навіть без зовнішнього втручання, гра створює нові комбінації рельєфу, змінюючи кількість і складність перешкод, плавно формуючи індивідуальний темп гри.

Висновки

У межах цієї роботи реалізовано модель процедурної генерації рівнів у 2D-платформері, що базується на комбінації пошарового підходу, фрактального шуму Перліна та адаптивної системи складності. Такий підхід дозволяє формувати унікальні ігрові рівні з природною топографією, адаптувати параметри генерації відповідно до стилю й навичок гравця, зберігати баланс між випадковістю та керованістю структури гри, а також підвищувати мотивацію до повторного проходження завдяки персоналізованому ігровому досвіду. Практична реалізація у Unity (C#) продемонструвала, що варіювання параметрів шуму (scale, persistence, octaves) у поєднанні з поведінковими метриками користувача дає змогу створювати динамічну систему рівнів, що природно реагує на дії гравця.

Отримані результати підтверджують ефективність комбінованого підходу як інструменту не лише для створення контенту, а й для розвитку концепції інтелектуального геймдизайну, у якому гра виступає адаптивним середовищем взаємодії з користувачем.

Розроблені алгоритми та методи мають універсальну структуру й можуть бути швидко інтегровані в середовища на C++, зокрема в рушії Unreal Engine та Cocos2d-x, без втрати функціональності чи потреби у суттєвій модифікації архітектури. Їхня модульність і незалежність від конкретного рушія дозволяє переносити ключові компоненти – генератор шуму, адаптивний контролер і пошарову логіку побудови рівня – без змін у логіці алгоритмів. Це забезпечує міжплатформність, сумісність із різними системами рендерингу й фізики та відкриває можливості для використання єдиного генеративного ядра в різних проєктах.

Таким чином, розроблена методика поєднує ефективність математичних моделей із практичною гнучкістю реалізації, що робить її придатною як для освітніх і дослідницьких експериментів, так і для професійної розробки ігор із процедурною генерацією та адаптивною складністю.

Подальший розвиток теми процедурної генерації пов’язаний із застосуванням розширених математичних моделей для керування не лише геометрією рівнів, а й поведінковими характеристиками ігрових агентів. Використання стохастичних процесів – таких як фрактальний броунівський рух чи випадкові поля Маркова – може забезпечити динамічну адаптацію складності ворогів у часі, узгоджену з діями гравця. У цьому контексті шумові функції здатні виконувати роль керуючих сигналів інтенсивності виклику, визначаючи щільність або агресивність супротивників. Поєднання цих підходів із методами оптимізації створює основу для єдиної адаптивної системи балансу, у якій структура рівня та поведінка ворогів еволюціонують синхронно, забезпечуючи природну й послідовну динаміку ігрового процесу.

Список використаної літератури

1. Горєлов, В. О., Іляш, Ю. Ю., Ровінський, В. А. (2018). Методи ущільнення даних та перетворення форми інформації у комп'ютерних іграх. Управління розвитком складних систем. № 35 с. 93-104.

2. Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games.: A Textbook and an Overview of Current Research. Springer. DOI: 10.1007/978-3-319-42716-4

3. Sepulveda, G. K., Besoain, F., & Barriga, N. A. (2019, November). Exploring dynamic difficulty adjustment in videogames. In 2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON) (pp. 1-6). IEEE. DOI**:** 10.1109/CHILECON47746.2019.8988068

4. Hunicke, R. (2005, June). The case for dynamic difficulty adjustment in games. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (pp. 429-433). DOI: 10.1145/1178477.1178573

5. Zohaib, M. (2018). Dynamic difficulty adjustment (DDA) in computer games: A review. Advances in Human-Computer Interaction, 2018(1), 5681652. DOI: 10.1155/2018/5681652

6. Biemer, C. F. (2023, October). Dynamic difficulty adjustment via procedural level generation guided by a Markov decision process for platformers and roguelikes. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 19, No. 1, pp. 436-439). DOI: 10.1609/aiide.v19i1.27540

7. Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games, 3(3), 172-186. DOI: 10.1109/TCIAIG.2011.2148116

8. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., … & Togelius, J. (2018). Procedural content generation via machine learning (PCGML). IEEE Transactions on Games, 10(3), 257-270. DOI: 10.1109/TG.2018.2846639

9. Risi, S., & Togelius, J. (2020). Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence, 2(8), 428-436.

10. González-Duque, M., Palm, R. B., Ha, D., & Risi, S. (2020, August). Finding game levels with the right difficulty in a few trials through intelligent trial-and-error. In 2020 IEEE Conference on Games (CoG) (pp. 503-510). IEEE. DOI: 10.1109/CoG47356.2020.9231548

11. Perlin, K. (1985). An image synthesizer. ACM Siggraph Computer Graphics, 19(3), 287-296. DOI: 10.1145/325165.325247

12. Dustler, M., Bakic, P., Petersson, H., Timberg, P., Tingberg, A., & Zackrisson, S. (2015, March). Application of the fractal Perlin noise algorithm for the generation of simulated breast tissue. In Medical Imaging 2015: Physics of Medical Imaging (Vol. 9412, pp. 844-852). SPIE. DOI: 10.1117/12.2081856

13. Risi, S., & Togelius, J. (2020). Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence, 2(8), 428-436.

References

1. Horelov, V. O., Ilyash, Yu. Yu., & Rovinsky, V. A. (2018). Methods of Data Compression and Information Shape Transformation in Computer Games. Management of Complex Systems Development, (35), 93–104.

2. Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games.: A Textbook and an Overview of Current Research. Springer. DOI: 10.1007/978-3-319-42716-4

3. Sepulveda, G. K., Besoain, F., & Barriga, N. A. (2019, November). Exploring dynamic difficulty adjustment in videogames. In 2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON) (pp. 1-6). IEEE. DOI**:** 10.1109/CHILECON47746.2019.8988068

4. Hunicke, R. (2005, June). The case for dynamic difficulty adjustment in games. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (pp. 429-433). DOI: 10.1145/1178477.1178573

5. Zohaib, M. (2018). Dynamic difficulty adjustment (DDA) in computer games: A review. Advances in Human-Computer Interaction, 2018(1), 5681652. DOI: 10.1155/2018/5681652

6. Biemer, C. F. (2023, October). Dynamic difficulty adjustment via procedural level generation guided by a Markov decision process for platformers and roguelikes. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 19, No. 1, pp. 436-439). DOI: 10.1609/aiide.v19i1.27540

7. Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games, 3(3), 172-186. DOI: 10.1109/TCIAIG.2011.2148116

8. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., … & Togelius, J. (2018). Procedural content generation via machine learning (PCGML). IEEE Transactions on Games, 10(3), 257-270. DOI: 10.1109/TG.2018.2846639

9. Risi, S., & Togelius, J. (2020). Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence, 2(8), 428-436.

10. González-Duque, M., Palm, R. B., Ha, D., & Risi, S. (2020, August). Finding game levels with the right difficulty in a few trials through intelligent trial-and-error. In 2020 IEEE Conference on Games (CoG) (pp. 503-510). IEEE. DOI: 10.1109/CoG47356.2020.9231548

11. Perlin, K. (1985). An image synthesizer. ACM Siggraph Computer Graphics, 19(3), 287-296. DOI: 10.1145/325165.325247

12. Dustler, M., Bakic, P., Petersson, H., Timberg, P., Tingberg, A., & Zackrisson, S. (2015, March). Application of the fractal Perlin noise algorithm for the generation of simulated breast tissue. In Medical Imaging 2015: Physics of Medical Imaging (Vol. 9412, pp. 844-852). SPIE. DOI: 10.1117/12.2081856

13. Risi, S., & Togelius, J. (2020). Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence, 2(8), 428-436.

Дата першого надходження рукопису до видання: 27.11.2025
Дата прийнятого до друку рукопису після рецензування: 12.12.2025
Дата публікації: 31.12.2025