

БАКАЛАВРСЬКА РОБОТА

БР. ІІ - 72.00.00.000 ІЗ

Група ІІ-22-3

Луцик Роман

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Луцик Роман Васильович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

БАКАЛАВРСЬКА РОБОТА

Застосування нейромереж для генерації процедурного контенту в іграх

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Луцик Р.В.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Крихівський Михайло Васильович, к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІПЗ
доцент. В.В. Бандура

“ ” 2026 р.

**ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ**

Луцик Роман Васильович

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) " Застосування нейромереж для генерації процедурного контенту в іграх "

керівник проекту (роботи) Крихівський Михайло Васильович, к.т.н., доцент

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення

2. Аналіз вимог і постановка задачі

3. Проектування системи

4. Реалізація проекту

5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1 Класи процедурно створюваних об'єктів (рис. 2.1, ст. 20)

2 Структурна схожість між трикутником (рис. 1.8, ст. 25)

4. Блок-схема, що ілюструє простий діалог. (рис. 2.6, ст. 40)

4 Зернистість типу поверхні додає випадковий фактор або вибирає текстуру бетону. (рис. 3.9, ст. 59)

5 Кількість абзаців для сюжетів з однією, чотирма та сімома діями (рис. 3.14, ст. 72)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання 2026 р.

Керівник

_____ (підпис)

Завдання прийняв до виконання

_____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент

_____ Луцик Р.В.
(підпис)

Керівник роботи

_____ Крихівський М.В.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 83 сторінок, 33 рисунки, 3 таблиці, список використаних джерел із 40 найменувань.

Метою роботи є дослідження застосування нейромереж для генерації процедурного контенту в іграх та аналіз підходів до створення адаптивного, різноманітного та масштабованого ігрового середовища.

Об'єкт дослідження: процеси генерації процедурного контенту в комп'ютерних іграх.

Предмет дослідження: методи, моделі та інструменти використання нейромереж і машинного навчання для генерації ігрового контенту.

Результати дослідження: проведено аналіз сучасних підходів до процедурної генерації контенту, досліджено можливості використання нейромереж для створення ігрових об'єктів, а також реалізовано приклади генерації контенту з подальшою оцінкою їх ефективності.

У першому розділі розглянуто теоретичні основи процедурної генерації контенту та нейромереж, принципи роботи та роль у сучасній індустрії.

У другому розділі досліджено методи та інструменти генерації ігрового контенту, зокрема генерацію ландшафтів, архітектурних об'єктів та застосування нейромереж для створення складних ігрових структур.

У третьому розділі розглянуто реалізацію системи генерації процедурного контенту, включаючи створення ігрових об'єктів та інтерактивних сценаріїв, а також проведено оцінку якості та ефективності.

Висновок: застосування нейромереж у процедурній генерації контенту дозволяє значно підвищити різноманітність, адаптивність і реалістичність ігрових середовищ, автоматизувати процес створення контенту та зменшити витрати часу на розробку.

КЛЮЧОВІ СЛОВА: ПРОЦЕДУРНА ГЕНЕРАЦІЯ КОНТЕНТУ, НЕЙРОМЕРЕЖІ, МАШИННЕ НАВЧАННЯ, ГЕНЕРАТИВНІ МОДЕЛІ, ІГРОВА РОЗРОБКА, GAN, VAE, ІНТЕРАКТИВНІ СЦЕНАРІЇ.

ANNOTATION

The thesis consists of 83 pages, 33 figures, 3 tables, and a reference list with 40 sources.

The aim of the work is to study the application of neural networks for procedural content generation in games and to analyze approaches to creating adaptive, diverse, and scalable game environments.

Research object: processes of procedural content generation in computer games.

Research subject: methods, models, and tools for using neural networks and machine learning in automatic generation of game content.

Research results: modern approaches to procedural content generation were analyzed, the capabilities of neural networks (including generative models) for creating game objects, levels, and scenarios were studied, and examples of content generation with further evaluation of their effectiveness were implemented.

The first section describes the theoretical foundations of procedural content generation and neural networks, their classification, operating principles, and role in the modern game industry.

The second section analyzes methods and tools for generating game content, including landscape generation, architectural object creation, and the use of neural networks for building complex game structures.

The third section covers the implementation of a procedural content generation system, including the creation of game objects and interactive scenarios, as well as evaluation of the quality and effectiveness of the obtained results.

Conclusion: the use of neural networks in procedural content generation significantly improves the diversity, adaptability, and realism of game environments, automates content creation, and reduces development time.

KEY WORDS: PROCEDURAL CONTENT GENERATION, NEURAL NETWORKS, MACHINE LEARNING, GENERATIVE MODELS, GAME DEVELOPMENT, GAN, VAE, INTERACTIVE NARRATIVES.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ КОНТЕНТУ ТА НЕЙРОМЕРЕЖ	10
1.1 Основні поняття процедурної генерації контенту та нейромереж	10
1.2 Підходи до генерації контенту в іграх та аналіз сучасних досліджень	15
1.3 Класифікація типів ігрового контенту, що генерується процедурно	19
1.4 Висновок по розділу	24
РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ ГЕНЕРАЦІЇ ПРОЦЕДУРНОГО КОНТЕНТУ	25
2.1 Методи генерації ігрових середовищ (ландшафти та рівні)	25
2.2 Генерація архітектурних об'єктів та структур у іграх	30
2.3 Застосування нейромереж та методів машинного навчання для генерації контенту	36
2.4 Висновок по розділу	47
РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ ТА ОЦІНКА ЕФЕКТИВНОСТІ	48
3.1 Реалізація генерації процедурних об'єктів (на прикладі будівель)	48
3.2 Генерація інтерактивних історій та сценаріїв із використанням нейромереж	62
3.3 Оцінка якості згенерованого контенту та перспективи розвитку	77
3.4 Висновок по розділу	82
ВИСНОВКИ	83
СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА	85

					БР.ІП – 72.00.00.000 ІПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Застосування нейромереж для генерації процедурного контенту в іграх				Літ.	Арк.	Акрушів
Розроб.	Луцик Р.В.										
Перевір.	Крихівський М.В.									7	
Реценз.	Зікратий С.В.								ІФНТУНГ ІП-22-3		
Н. Контр.	Піх М.М.										
Затверд.	Бандура В. В.										
					Пояснювальна записка						

ВСТУП

У сучасній індустрії відеоігор спостерігається стрімкий розвиток технологій, що спрямовані на підвищення якості, реалістичності та варіативності ігрового контенту. Одним із ключових напрямків є процедурна генерація контенту, яка дозволяє автоматизувати створення ігрових рівнів, об'єктів, сценаріїв та цілих світів. Зростання складності ігор, необхідність створення великих відкритих світів, а також обмеження ресурсів розробників підкреслюють актуальність застосування інтелектуальних методів для генерації контенту. У цьому контексті особливого значення набуває використання нейромереж, які здатні навчатися на великих обсягах даних і генерувати новий контент із високим рівнем різноманітності та адаптивності.

Актуальність теми зумовлена необхідністю підвищення ефективності процесу розробки ігор шляхом автоматизації створення контенту та зменшення витрат часу і ресурсів. Традиційні методи процедурної генерації часто базуються на жорстко заданих правилах і алгоритмах, що обмежує гнучкість і різноманітність результатів. Використання нейромереж, зокрема генеративних моделей, відкриває нові можливості для створення складних, реалістичних і динамічних ігрових середовищ, здатних адаптуватися до дій користувача. Це робить дослідження даної теми важливим як з наукової, так і з практичної точки зору.

Метою роботи є дослідження та розробка підходів до застосування нейромереж для генерації процедурного контенту в іграх з метою створення різноманітного, адаптивного та якісного ігрового середовища.

Завданнями дослідження є аналіз предметної області та сучасних підходів до процедурної генерації контенту, вивчення теоретичних основ нейромереж та їх застосування в ігровій індустрії, дослідження методів генерації ігрових середовищ, об'єктів і сценаріїв, вибір та обґрунтування інструментів реалізації, проектування архітектури системи генерації контенту, реалізація прототипу із використанням нейромереж, тестування та оцінка ефективності

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

отриманих результатів, а також формулювання висновків щодо перспектив подальшого розвитку.

Об'єктом дослідження є процеси генерації процедурного контенту в комп'ютерних іграх.

Предметом дослідження є методи, моделі та інструменти застосування нейромереж і машинного навчання для автоматичної генерації ігрового контенту, включаючи генерацію ландшафтів, архітектурних об'єктів та інтерактивних сценаріїв.

Методи дослідження включають аналіз наукових джерел для вивчення теоретичних основ і сучасних рішень, порівняльний аналіз підходів до генерації контенту, моделювання процесів генерації, експериментальний метод для реалізації та тестування системи, а також оцінку якості згенерованого контенту за визначеними критеріями.

Розроблена система передбачає можливість генерації різних типів ігрового контенту, зокрема ландшафтів, будівель та інтерактивних сценаріїв, із використанням нейромереж. Особлива увага приділяється якості генерації, різноманітності результатів та можливості адаптації контенту до дій користувача. Очікується, що застосування запропонованих підходів дозволить підвищити ефективність розробки ігор, зменшити витрати ресурсів та покращити користувацький досвід.

Бакалаврська робота містить 83 сторінок, 33 рисунків, 3 таблиці, 3 розділи, висновки та список використаних джерел із 40 найменувань.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ КОНТЕНТУ ТА НЕЙРОМЕРЕЖ

1.1 Основні поняття процедурної генерації контенту та нейромереж

Віртуальні світи стають дедалі важливішими в контексті ігор та симуляцій. Наприклад, якщо поглянути на *Skyrim* [1] від Bethesda, ми бачимо, що складність та зусилля, пов'язані зі створенням правдоподібного та реалістичного 3D-світу, є завданням кількох місяців для великої команди професійних дизайнерів. Але не лише загальна візуальна якість відіграє важливу роль у процесі проектування; також величезна кількість різних об'єктів, необхідних для уникнення повторюваного вигляду, коли такі об'єкти, як меблі, рослини чи будівлі, часто з'являються [2]. Як результат, студії розробки ігор стикаються з трьома проблемами:

- створити реалістичне та надійне середовище у розумні терміни,
- вкластися в бюджет проекту, щоб зробити результат доступним для кінцевого споживача,
- привнести креативність, щоб надати грі індивідуального та інноваційного вигляду.

Відомий геймдизайнер Білл Райт (*The Sims* [3], *Sim City* [4]) довів, що ці проблеми існують не лише в теорії, а й на практиці. Він назвав це «Проблемою Гори Контенту» [5]. *Процедурна генерація* - один із підходів до вирішення цих проблем. Роками розробники винаходили методи та шаблони для створення текстур, 3D-моделей або навіть цілих ігрових рівнів, використовуючи алгоритми для створення унікального середовища, яке не потребує або потребує дуже малого налаштування. Ще однією причиною зростаючого інтересу до процедурної генерації контенту (PCG) є бажання зробити гру варту повторного проходження, змінюючи рівні чи квести та пропонуючи нові історії та враження в кожному новому сеансі. Механіка гри залишається незмінною, щоб гравець міг використовувати набуті здібності та навички. Прикладами ігор, які

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

використовують ці можливості, є *Elite: Dangerous* [6] або *Minecraft* [7], які створюють нові світи на початку кожної гри, якщо гравець цього бажає.

Принцип PCG не обслуговує виключно сферу цифрових ігор. Настільні ігри, такі як «*Поселенці Катану*» [8], вимагають від гравців створювати карту світу, випадковим чином розподіляючи поля ресурсів на столі [9], що дає гравцеві абсолютно новий досвід у кожній грі. Такий підхід призвів до величезного успіху гри.

Крім того, полікоментарна інформативність (ПКГ) вплинула на розвиток *серйозних ігор*, з акцентом на навчанні, окрім розваг, як у класичних іграх [10]. У сфері освіти зміна умов призводить до того, що студенти не лише повторюють вивчені знання чи рухи, але й застосовують свої навички в нових ситуаціях. Ці ситуації можна генерувати випадковим чином за допомогою процедурних механізмів генерації. На прикладі віртуальної автошколи згенеровані дорожні мережі можуть навчити гравців орієнтуватися в реальному світі в невідомій місцевості. Інші сценарії для випадково згенерованих середовищ можна легко вивести для серйозних ігор у різних галузях, таких як транспорт, охорона здоров'я та маркетинг.

Ця робота зосереджена на процедурному створенні віртуальних світів для ігор. Вона містить удосконалення існуючих підходів PCG, а також нові методи з акцентом на практичному використанні. Особливо виділено області *процедурного моделювання та процедурного сторітелінгу*.

Історія створення процедурного контенту

Вже у 1978 році використовувалися прості алгоритми у своїй грі *Beneath Apple Manor* [11] для створення підземель для цієї RPG (рольової гри) [12]. Більш захопливою сенсацією стала гра *Rogue* [13], в якій також використовувалися алгоритми для створення рівнів для цього дуже відомого підземельного краулера [14]. Гра була випущена у 1980 році. Назва *Rogue* послужила епонімом для наступного покоління *Rogue-like* ігор, які мали такі особливості:

- покроковий ігровий процес,
- процедурно згенеровані рівні,

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

- безповоротна смерть (без функції завантаження/збереження).

Всупереч очікуванням, розробники не обрали ні генеративний підхід до створення рівнів - в інтересах того, щоб гра знову стала доступною - ні зіткнулися з *проблемою гори контенту*. Фактично, в ті часи вимоги до пам'яті для гри з багатьма різними рівнями були занадто високими, тому було прийнято рішення генерувати рівні на льоту, а не записувати їх на диск [15].

З точки зору дослідження, стаття «Текстурування твердих тіл», опублікована в 1985 році, була однією з перших публікацій, яка безпосередньо стосується генерації процедурного контенту [16]. Подібно до сучасного *відображення норм*, Пічі запропонував техніку, яка дозволяла двовимірним текстурам виглядати тривимірними. За цією ранньою статтею послідували різні інші публікації, що стосувалися, наприклад, генерації рельєфу [17] або проектування та анімації рослин за допомогою фракталів [18], і таким чином створення генерації процедурного контенту як наукової галузі досліджень.



Рисунок 1.1 - Гра .kkrieger має розмір файлу 96 кб.

Отримані методи використовувалися не лише в індустрії цифрових ігор, але й в анімаційному кіно. Наприклад, анімаційний інструмент Pixar *RenderMan* [19] *пропонував процедурні* функції для алгоритмічного визначення текстур і матеріалів або для генерації безлічі простих примітивів за допомогою рукописних підпрограм. Зі зростанням зрілості досліджень і практики, генеративні методи все частіше використовуються в AAA-іграх (AAA-гра – це

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

широко використовуваний термін для ігор з високими бюджетами на розробку та просування), таких як *Command and Conquer: Red Alert 2* [20], *Diablo* [21] або *The Elder Scrolls II: Daggerfall* [22]. У цих іграх розробники навмисно реалізували PCG як конкретний ігровий елемент, а не через обмежені ресурси, як це було раніше.

У 2000 році EA Games випустила *The Sims* [3], яка не лише представила абсолютно нову ігрову концепцію, але й дуже інноваційний, але простий у використанні редактор персонажів. Гравці могли створювати персонажа в грі у вигляді детальної 3D-моделі, що складається з налаштовуваних частин тіла, обличчя, одягу та особистості. Використання редактора не означало переміщення окремих вузлів та налаштування UV-карти (UV-карта — це проекція поверхні 3D-моделі на 2D-зображення), як це робиться в інструментах моделювання, таких як *Blender* [23]; Maxis розробила редактор, щоб запропонувати набір параметрів для редагування окремих рис людини, таких як талія, розмір або винос очей. Відповідні обмеження допомагають гравцям у розробці своїх персонажів, щоб забезпечити правдоподібний та переконливий вигляд результату. Багато 3D-інструментів реалізували подібний робочий процес для створення гуманоїдних моделей, такі як Autodesk *Character Generator* (раніше відомий як Autodesk Pinocchio) [24], *Fuse* [25] або додаток з відкритим кодом *Makehuman* [26]. Ми розглядаємо цей підхід як процедурну генерацію контенту з сильною взаємодією з користувачем. Через рік, у 2004 році, група .theprodukt випустила простий шутер під назвою *.kkrieger* [27], який виник у демонстраційній сцені (див. рисунок 1.1). Метою цієї сцени є створення візуально або акустично вражаючих додатків, які часто називають цифровим мистецтвом, оскільки вони показують прості, рухомі сцени, подібні до картини [28].

Окрема категорія зосереджена на демоверсіях з максимальним розміром файлу 4, 8 або 64 кб. Такий малий розмір не дозволяє використовувати готові ресурси, такі як графіка, музика чи моделі. Натомість ці ресурси створюються процедурно під час виконання або перед початком демоверсії. Враховуючи, що

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

шахрайські ігри також визначаються пам'яттю, є деякі паралелі. Використовуючи *.kkrieger* як приклад, команда розробників *.theprodukt* генерувала анімацію, рівні, текстури, шейдери та музику на основі набору інструментів з методами обробки. Маючи великий інтерес до фінальної демоверсії (розмір 96 кб), розробники вирішили опублікувати свої інструменти у вигляді редактора *.werkzeug* [29]. Вважається комп'ютерну графіку джерелом процедурної графіки.

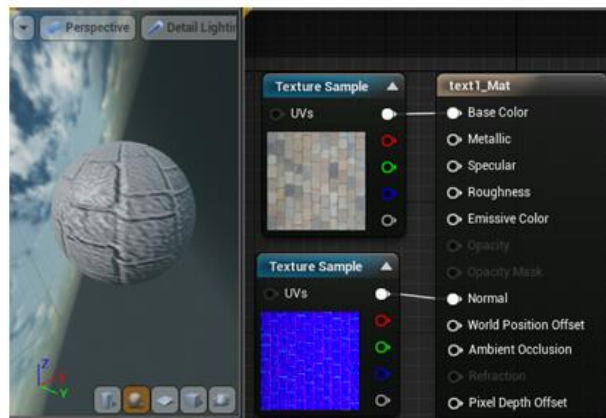


Рисунок 1.2 - Редактор матеріалів движка Unreal Engine

Сміт присвячує цілу статтю історії PCG та говорить про розвиток від модульності дизайну до алгоритмічного складання контенту. Крім того, вона підсумовує мотивації використання PCG, такі як можливість повторного проходження, PCG як виразний засіб або як допоміжна креативна технологія [32]. Сьогодні більшість обговорюваних утиліт з'явилися в сучасних інструментах розробки ігор та сприяють створенню процедурного контенту. 3D-моделі створюються шляхом деформації, вирізання або об'єднання примітивів, а не поєднання їх трикутником за трикутником (наприклад, у *ZBrush* [33]). Це дозволяє розробникам зосередитися на процедурних алгоритмах, що використовуються для створення правдоподібного контенту, а не на технічних проблемах деяких нижчих рівнів. Прикладами є ігрові рушії *Unreal Engine 4* [34], *Unity 2017* [35] та *Cry Engine V* [36], які надають інструменти для легкого маніпулювання місцевістю та малювання рослинності, такої як дерева чи квіти

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

на землі, що значно пришвидшує робочий процес створення рівнів. Це дозволяє розробникам зосередитися на пошуку алгоритму для розподілу об'єктів рослинності по рівню, не турбуючись про зіткнення квітів чи повторно сформовані дерева [30].

Визначним досягненням сучасних інструментів розробки ігор є процедурна генерація текстур і матеріалів. Подібно до *.theprodukt*, який був представлений у *.werkzeug* кілька років тому, процедурна генерація текстур і матеріалів зараз використовується більшістю великих гравців на ринку ігрових двигунів. Крім того, замість ручного написання шейдерів (на GLSL (OpenGL Shading Language) або HLSL (High-Level Shader Language)), графічним дизайнерам були запропоновані модульні інструменти для побудови матеріалів шляхом комбінування зображень, математичних операцій або функцій шейдерів, наприклад, для створення ефекту відображення нормалей, як показано на рисунку 1.2.

Наступне покоління ігрових двигунів почало реалізовувати функції для створення ще складніших об'єктів [31]. Наприклад, Еріс включав простий інструмент для генерації 3D-будівель.

1.2 Підходи до генерації контенту в іграх та аналіз сучасних досліджень

Визначається генерацію процедурного контенту як «будь-який вид автоматично згенерованого ресурсу на основі обмеженого набору вхідних параметрів, визначених користувачем» [39]. Вони також посилаються на авторів, які називають ці типи алгоритмів алгоритмами *ампліфікації*, беручи невеликий набір вхідних параметрів для перетворення їх на більший набір вихідних даних. Формулюється визначення шляхом антитези, стверджуючи, що процедурно згенерований контент не відповідає контенту, згенерованому користувачами, навіть якщо вони використовують процедурні алгоритми, оскільки їх необхідно параметризувати вручну. Розглянуто процедурну генерацію як альтернативу

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

ручному проектуванню, але наголошують на необхідності можливої параметризації, щоб дизайнери могли впливати на згенерований об'єкт. Визначають PCG, використовуючи приклади того, що таке PCG (наприклад, програмний інструмент для створення випадкових підземель без втручання користувача), а що ні (редактор карт, який дозволяє користувачам розміщувати елементи) [35].

На цьому етапі ми хотіли б надати власне визначення ПХГ:

Процедурна генерація контенту - це автоматичне створення цифрових ресурсів для ігор, симуляцій або фільмів на основі заздалегідь визначених алгоритмів і шаблонів, що потребують мінімального втручання користувача.

ППК є предметом дослідження не лише в інформатиці. Наголошують на зростаючому інтересі до інших спільнот, що виникає внаслідок міждисциплінарності та впливає як на природничі науки, так і на біологію. Цей сильний інтерес до інших галузей досліджень є показником присутності теми. Однак гармонізація всіх цих дисциплін, таких як біологія, архітектура, урбаністика, психологія тощо, та пошук правильних формалізмів та структур даних є величезними зусиллями. Перелік уражених галузей інформатики до:

- граматики,
- L-Системи,
- граматики форм,
- мови програмування.

Він також зазначає, що програмування є найгнучкішим, але схильним до помилок методом автоматичної генерації процедурного контенту. Вводиться аббревіатуру PCG-G (procedural content generation for games - генерація процедурного контенту для ігор), щоб відрізнити PCG для ігор від інших галузей, таких як симуляції або (анімаційні) фільми. Це показує, що методи, алгоритми та інструменти на основі PCG можуть бути застосовані в різних галузях, таких як міське планування (наприклад, Esri CityEngine) або (анімаційна) кіноіндустрія [37]. Pixar - це компанія, яка використовує генерацію процедурного контенту в *RenderMan*; крім того, Disney Research надає публікації, в яких згадуються

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

процедурні методи, наприклад, для редагування віртуальної місцевості. Це показує, що переваги автоматизованої генерації контенту добре відомі в кіноіндустрії.

Теоретичні міркування

Інформатика часто вимірює ефективність та зрілість програмного забезпечення або алгоритмів, щоб забезпечити їхню якість та застосовність. Широко використовуваною метрикою є проста суб'єктивна оцінка того, наскільки *реалістичним виглядає згенерований контент*; це не так, якщо людина може легко ідентифікувати його як автоматично згенерований. Ми пропонуємо балансувати між продуктивністю та точністю в PCG: якщо алгоритм повертає точний результат (наприклад, ліс природного вигляду), алгоритму потрібна більша обчислювальна потужність, більше пам'яті або більше сховища для створення більшої кількості варіацій дерев, текстур з вищою роздільною здатністю, більш детальних сіток або більшої щільності посадки. Залежно від бажаного результату, користувач повинен вибирати між продуктивністю та реалізмом, щоб досягти оптимуму для даної системи та вимог віртуального світу.

Часто згенеровані об'єкти можна класифікувати як ручної роботи або як створені природою. Центральне питання полягає в тому, чи можна визначити, чи можна створити одну чи іншу категорію з меншими зусиллями, ніж іншу. Наприклад, складність завдань зі створення 3D-дерев можна порівняти зі складністю завдань зі створення 3D-будівель. Якщо значна частина видимого контенту створюється автоматично, порядок його створення стає актуальним. Тому ми порівнюємо дуже спрощену проєкцію природного утворення Землі (див. рисунок 1.3b) з процедурною генерацією віртуального світу (див. рисунок 1.3a). У концепції багатьох розробників PCG про природне створення віртуального світу ландшафт (включаючи воду) на планеті служить основою світу, а потім рослинність, така як дерева та рослини. Пізніше люди будують будівлі на цьому ландшафті та з'єднують їх дорожніми мережами. Потім з роками поселення росте або гине. Коли на шляху з'являються рослинність або гори, людство прагне видалити їх, щоб побудувати дороги або будівлі.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

Відображення комп'ютерної графіки віртуального світу на спрощене природне творіння Землі призводить до деяких додаткових ітерацій, де ліси вирубуються для будівництва доріг, а місцевість вирівнюється.

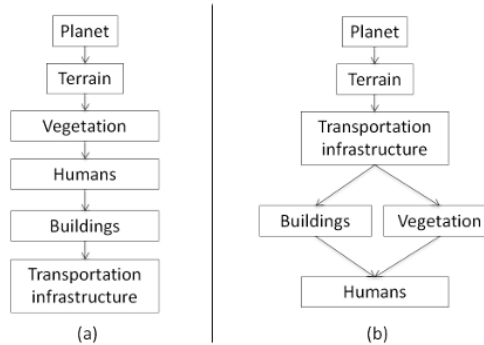


Рисунок 1.3 - Побудова віртуального світу може відбуватися або у спрощеному природному порядку (а), або в оптимізованому процедурному (б).

щоб розмістити на ньому міста, або річки перенаправляються, щоб місто розвивалося в потрібному напрямку. Як альтернативу, ми пропонуємо використовувати порядок, показаний на рисунку 1.3b, для створення місцевості, потім транспортну інфраструктуру, а потім будівлі та рослинність. Ця процедура, найімовірніше, не відповідатиме ідеї природного зростання, але вона може полегшити комп'ютерну генерацію віртуальних світів.

Генератори випадкових чисел

Генератори випадкових чисел (ГВЧ) можуть бути як апаратним пристроєм, так і програмним забезпеченням. Вони генерують детерміновану та періодичну послідовність (псевдо)випадкових чисел. Їхнє існування та функціональність передбачається в багатьох публікаціях з ГВЧ [32]. Не всі дослідники погоджуються з їхньою повсюдністю в ГВЧ. Деякі зазначають, що чисто випадкова генерація призвела б до хаосу. Ми вважаємо, що використання випадковості залежить від контексту кожного процесу генерації. На прикладі дорожньої мережі розміщення доріг та перехресть не обов'язково має відповідати розумній схемі, але алгоритм генерації сіток та текстур дорожньої мережі має бути належним чином реалізований заздалегідь

1.3 Класифікація типів ігрового контенту, що генерується процедурно

Тепер ми пропонуємо класифікацію типів об'єктів, які найчастіше є предметом процедурної генерації контенту (див. рис. 1.4). Ми взяли специфікацію *CityGML* (особливо схему *CityGML Core*) за основу та додали до неї живих істот, щоб задовольнити потреби опису об'єктів у процедурно згенерованому світі [30]. У наступному розділі ми представимо рослинність, воду, дорожні мережі, будівлі, істот, людей та історії як типові приклади для цих класів. Робота, описана тут, була опублікована в [21].

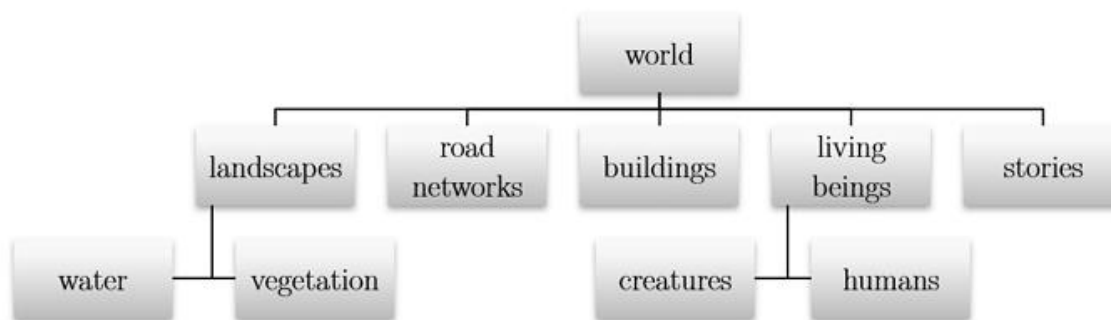


Рисунок 1.4 - Класи процедурно створюваних об'єктів

Рослинність та ландшафт

Процедурне створення об'єктів, що зустрічаються в природі, таких як ландшафти чи рослини, належить до найбільш досліджених галузей у процедурній генерації контенту. Теорії та програмні інструменти існують вже багато років і досягли високого рівня зрілості.

Генерація рослинності

Якщо розглядати зокрема рослини, то однією з причин сильного інтересу до їхньої будови є теоретична біологія: натхненні їхньою красою, дослідники намагалися знайти математичні моделі їхнього росту. Угорські дослідники Прусінкевич і Лінденмайєр були піонерами в цій галузі. Ще в 1968 році вони запропонували граматику під назвою *L-Системи* (Системи Лінденмайєра) для

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

полягає в тому, що ріст рослин базується на поділі клітин, і це відбувається паралельно для всіх клітин. Якщо ми візьмемо наш приклад вище з 'а' як початковим рядком, ми можемо згенерувати такі слова нашою мовою:

а
аб
АБА
абааб
абаабаба.

У L-системі цей процес виробництва описує, як росте рослина. У більш поширеному записі ми позначаємо можливі заміни дужками навколо відповідного підрядка. Наприклад, ми могли б визначити L-систему з перезаписом наступним чином (n – кількість застосувань правила перезапису, 'F' у другому рядку – початковий рядок):

$$n = 1, \delta = 25^\circ$$

$$F$$

$$F \rightarrow F[+F]F[-F]F.$$

Для графічного представлення ми знову використовуємо модель черепахи. '[' інтерпретується як натискання на стек, ']' як виштовхування зі стеку. Вищезгадана L-система потім створює рослину, показану на рисунку 1.6а.

Тепер ми застосовуємо правило перезапису двічі, тобто змінюємо нашу продукційну систему.

Тоді ми отримуємо рослину, зображену на рисунку 1.6b. Якщо використовувати $n = 5$, то отримуємо реалістичну рослину, зображену на рисунку 1.6с.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

$$n = 2, \delta = 25^\circ$$

$$F$$

$$F \rightarrow F[+F]F[-F]F.$$

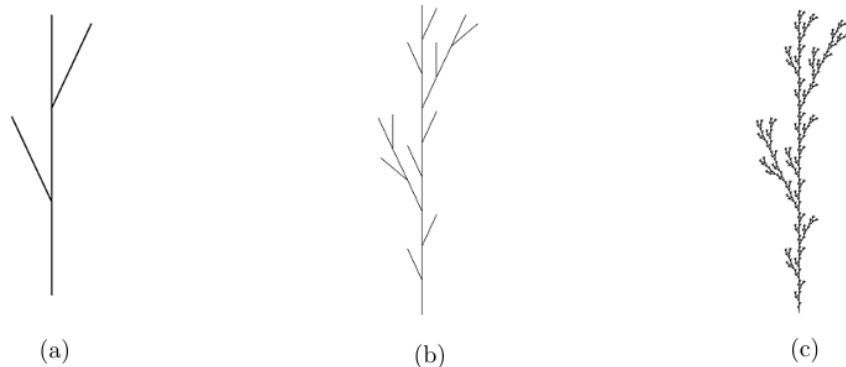


Рисунок 1.6 - Прості рослини, згенеровані за допомогою L-системи з дужками та різною кількістю ітерацій правила перезапису. (а) одна ітерація; (б) дві ітерації; (в) п'ять ітерацій.

Якщо ми використовуємо детерміновані L-системи, то вже отримали можливість створити велику кількість рослин для нашого віртуального світу, які всі виглядають однаково. Це здається неприродним. Таким чином, нам потрібно ввести стохастичний компонент: кожен крок у побудові рослини виконується із заданою ймовірністю. Давайте введемо ймовірності в L-систему зверху:

$$n = 5, \delta = 25^\circ$$

$$F$$

$$p_1 = 0.33: F \rightarrow F[+F]F[-F]F$$

$$p_2 = 0.33: F \rightarrow F[+F]F$$

$$p_3 = 0.34: F \rightarrow F[-F]F.$$

Числа p_i – це ймовірності утворення; вони в сумі дають 1. Ця стохастична L-система може давати рослини, показані на рисунку 1.7. Зауважте, що всі вони, здається, належать до одного виду, просто є варіаціями на різних рівнях росту.

Досі ми описали, як можна генерувати 2D-рослини. Нашу граматику L-системи легко поширити на 3D-рослини: ми просто додаємо операції «нахилити вниз», «нахилити вгору», «повернути ліворуч» та «повернути праворуч» до

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

початкових операцій «повернути праворуч, повернути ліворуч» у кожній точці прийняття рішення. Це дозволяє нам описувати велику різноманітність 3D-рослин за допомогою коротких правил. Читач може уявити, що також легко розширити L-системи з різною шириною та кольорами для різних гілок. Аналогічно, L-системи були адаптовані до кущів та інших типів рослин. Було опубліковано багато статей, присвячених деталям. До сьогодні варіації ранніх L-систем є найбільш широко використовуваними методами процедурної генерації рослин.



Рисунок 1.7 - Кілька рослин, згенерованих за допомогою стохастичної L-системи, описаної вище.

Окрім чистих L-систем, існують різні інші системи. Наприклад, пропонують систему моделювання дерев на основі ескізів, яка використовує базу даних дерев для 3D-макета. Користувач малює ескіз базової структури гілок бажаного дерева та, за бажанням, контур крони кількома штрихами у 2D. Потім система шукає відповідні 3D-структури дерев у базі даних з 20 моделей дерев. Найкраще збіг витягується та використовується для створення бажаного дерева. У пов'язаних роботах інші автори пропонують використовувати фотографії дерев для пошуку відповідної моделі в базі даних.

Самоподібність рослин також часто використовується для їх автоматичної генерації. *Множина Мана -дельброта* [33], *сніжинка Коха* або *дерево Піфагора* є прикладами математичних моделей, які можна візуалізувати як форми, що нагадують природні структури (див. рис. 1.8). Якщо розглянути

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

практичне застосування методів генерації дерев і рослин , можна побачити, що найпоширеніші інструменти *Xfrog* або *Speedtree* досягають вражаючих результатів.

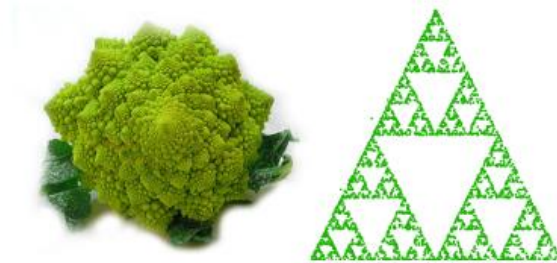


Рисунок 1.8 - Структурна схожість між трикутником

Пропонується підхід на основі процедурного графу гілок (PBG), який створює різноманітні дерева з однаковою структурою гілок на різних рівнях деталізації (LOD). Подальші сучасні дослідження, підтверджують високий рівень зрілості цієї галузі досліджень.

1.4 Висновок по розділу

У першому розділі було розглянуто теоретичні основи процедурної генерації контенту та використання нейромереж у розробці ігор. Проаналізовано основні поняття процедурної генерації контенту та принципи роботи нейромереж, що використовуються для автоматизованого створення ігрових елементів. Досліджено сучасні підходи до генерації контенту в іграх та проведено аналіз актуальних наукових досліджень у цій сфері. Також розглянуто класифікацію типів ігрового контенту, який може генеруватися процедурно, зокрема рівнів, ландшафтів, персонажів і сценаріїв. Отримані результати формують теоретичну основу для подальшого дослідження методів генерації контенту із застосуванням нейромереж.

РОЗДІЛ 2 МЕТОДИ ТА ІНСТРУМЕНТИ ГЕНЕРАЦІЇ ПРОЦЕДУРНОГО КОНТЕНТУ

2.1 Методи генерації ігрових середовищ (ландшафти та рівні)

Як і процедурна генерація рослинності, генерація ландшафтів у вигляді карт висот належить до складніших тем у PCG. У більшості випадків карти висот складаються з сіроградових растрових зображень, на яких висота представлена відтінком сірого.

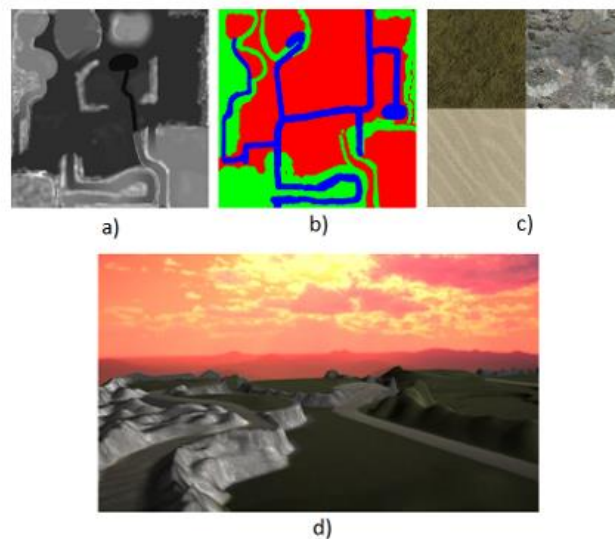


Рисунок 2.1 Створення ландшафту (d) за допомогою карти висот (а), карти кольорів (b) та текстур (c).

пікселя растрового зображення (див. рис. 2.1а). Розрахована висота потім проектується на плоску 3D-сітку. Зазвичай, чим яскравіші пікселі, тим вища висота. Після створення сітки рельєф потрібно розфарбувати. Для цього завдання існує три усталені методи:

- ручне малювання текстури,
- використання вручну створеної карти кольорів для проектування текстур на певні регіони,
- створення текстури шляхом аналізу схилів та висот сітки рельєфу.

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

Ці три підходи відрізняються якістю та застосовністю. Процес ручного малювання текстури є простим, але призводить до величезного растрового зображення або недостатньої роздільної здатності. Малювання карти кольорів (див. рисунок 2.1b), яка використовується в грі або симуляції для відображення набору текстур (див. рисунок 2.1c) на певні кольори (зазвичай за допомогою шейдера), є поширеною практикою та повертає візуально вражаючі результати (див. рисунок 2.1d), але все ще є ручним процесом.

Третій метод використовує дані сітки та застосовує висоти й схили для розрахунку відповідного відображення текстури. Дуже низькі ділянки зазвичай розглядаються як океан і тому текстуруються текстурою океанічного ґрунту, середні висоти текстуруються за допомогою трави, ділянки з великою висотою розглядаються як гори і отримують текстуру каменю, а дуже високі регіони отримують текстуру снігу [38]. Схили можна використовувати для визначення крутих ділянок, які часто представлені каменем (наприклад, скелі), і тому отримують текстуру каменю. Оскільки перші два методи вимагають від дизайнера створення або карти кольорів, або текстури, вони не вважаються застосовними для суто процедурного підходу. Навпаки, третя процедура добре підходить, оскільки вона повністю спирається на сітку місцевості. У новітніх іграх, таких як Minecraft [7], біоми вступають у гру та містять кліматичну інформацію, таку як вологість і температура. Біоми - це області суші, і, залежно від їхніх характеристик, місцевість формується відповідно до висоти та текстур. Суміжні біоми можуть або змішуватися, або мати різкі межі залежно від дизайнерських рішень. Знову виникає питання, як процедурно генерувати карту висот. Існують різні техніки для цього:

- фрактальний шум, наприклад, шум Перліна,
- зміщення середньої точки (здебільшого використовується для створення 2D-ландшафтів),
- алгоритм ромбоподібно-квадратного типу (адаптує зміщення середньої точки для створення 3D-рельєфу).

Це лише три приклади величезної різноманітності алгоритмів. Для

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

практичної реалізації існують деякі відкриті бібліотеки, такі як *libnoise*, що містять модулі для створення шуму або інших шаблонів, таких як шахова дошка або *діаграми Вороного* [19]. Останні можна використовувати для досягнення неоднорідного вигляду форми місцевості, на відміну від алгоритмів шуму. Враховуючи, що віртуальний світ вимагає мільйонів квадратних метрів, необхідно узгодити спосіб досягнення оптимальної продуктивності процесу рендерингу. Рекомендується використовувати LOD. Його слід використовувати для додавання більшої деталізації місцевості у важливих та часто відвідуваних областях місцевості, які знаходяться близько до користувача, та для зменшення деталізації в менш важливих регіонах, наприклад, у віддалених гірських районах [39]. Як покращення статичного рішення, запропоновано алгоритм оптимізації в реальному часі під назвою *Real-time Optimally Adapting Meshes* (ROAM), який оптимізує тріангуляцію сітки під час виконання залежно від кута зору гравця. Створили систему створення ландшафту лабіринту (MAVE) для розрахунку скінченного ландшафту лабіринту на основі різних шаблонів лабіринту.

Обмеженням поширених алгоритмів генерації рельєфу є створення печер або нависів, які можна реалізувати за допомогою шаруватих рельєфів або воксельних рельєфів. Пропонується не лише метод створення печер з різними характеристиками, але й ефективне зберігання їхніх даних у структурі даних октодерев. Використовують 3D-моделі для створення печер на основі шаблонів, використовуючи попередньо створені зображення шаблонів.

Розміщення рослинності в ландшафті

Після створення набору рослин і дерев необхідно відповісти на питання їхнього правильного розміщення у віртуальному ландшафті. Загалом, слід розрізняти розміщення рослин у великій кількості на певній ділянці та індивідуальне розміщення. Подібно до описаних вище карт висот, можна використовувати метод, заснований на зображенні в сірій шкалі, в якому щільність рослинності визначається відтінком сірого. Подібно до підходу текстуровання місцевості шляхом аналізу висот і схилів, пропонує процедуру розміщення рослин і дерев на основі шкали сірості місцевості під ними. Інша ідея

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

полягає у використанні кольорових карт, де кожен колір відповідає певному типу рослинності. Цікаву альтернативу запропоновано моделювати природні ресурси, доступні для рослин, і конкуренцію між ними, щоб визначити їхню щільність. У моделі *FON*, кожна рослина має кругову зону впливу на сусідні рослини. Розмір цієї зони може залежати, наприклад, від вологості ґрунту, родючості ґрунту, а також типу та розміру рослини. Вони складають ландшафт з плиток, кожна з яких представляє задану щільність рослин відповідно до моделі *FON*. Переходи між плитками згладжуються методами релаксації. Властивість ґрунту, а отже, і вибрані плитки Ванга залежать від висоти, наявності води тощо. Таким чином, вони можуть автоматично створювати дуже реалістичні області рослинності з різною щільністю.

Інтегровану систему моделювання місцевості та рослин. Описано цілий набір інструментів для цього процесу. Він дозволяє як ручне редагування карт висот і рослин, так і їх процедурну генерацію. Цікава ідея полягає в тому, щоб вручну забезпечити початковий розподіл рослин, а потім алгоритмічно моделювати їх ріст і загибель протягом певного часу, враховуючи конкуренцію рослин; кінцевий результат потім представляється графічно. Інша ідея полягає в зменшенні геометричної складності сцени шляхом *приблизного створення екземплярів*, замінюючи подібні рослини, групи рослин і частини рослин репрезентативними об'єктами перед рендерингом. Кілька вражаючих прикладів демонструють реалізм їхнього підходу. *Розподіл Пуассона* - це ще один підхід до розміщення рослин [40]. Тут ймовірна кількість рослин розподіляється в частковій області сітки. *Розподіл Пуассона за диском* дозволяє уникнути росту рослин занадто близько одна до одної, визначаючи зовнішній радіус, в якому не можна розмістити жодну іншу точку розподілу.

Вода

Хоча вода як елемент завжди однакова, створення річок, океанів, озер та водоспадів багато в чому відрізняється. У той час як океани та озера більш-менш спокійні, річки та водоспади постійно рухаються. Створення річок часто обговорюється двома способами: або вони генеруються під час створення

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

місцевості, або річкові русла розміщуються пізніше в ландшафті на окремому кроці [39]. Інший варіант - звернутися до рівня моря та припустити наявність води всюди у віртуальному світі під цією заздалегідь визначеною висотою. На противагу цьому, пропонується процедура природного зростання шляхом розтікання потоку води. Цей потік потім ітеративно формує річкові русла. Еберт посилається на використання *дилатаційної симетрії* для досягнення реалістично виглядаючих річок, де кожна менша річкова гілка виглядає точно так само, як більша гілка в меншому масштабі. Процедурне створення морів та річок рідко досліджується [39] і зосереджується головним чином на формі та руслі річок. У кількох статтях автори часто розрізняють підхід на основі сітки та підхід на основі сітки при створенні річок; Сітчастий підхід досягає візуально вражаючих результатів, оскільки 3D-сітка генерується окремо вздовж річки, тоді як сітчастий підхід зосереджується на існуючому макеті.

Згадується агентів берегової лінії для створення реалістичних форм островів. Створення берегових ліній або дії прибою та хвиль можна знайти у фреймворках у вигляді конкретних реалізацій. Наприклад, інструмент *Mystymood* автоматично генерує берегові лінії, підводні каустики та берегові розриви, використовуючи простий алгоритм зіткнень та кольорові карти.

Дорожні мережі

Зусилля, докладені для процедурного створення транспортної інфраструктури. У цьому контексті основна увага приділяється створенню дорожніх мереж, включаючи пішохідні доріжки. Оскільки повітряний та морський рух потребують лише незначної фізичної інфраструктури, ці дві теми виключені з нашого обговорення.

Генерація моделей дорожньої мережі залежить від автоматизованої генерації 3D-сіток, про що буде коротко згадано тут. Автоматизована генерація сіток для складних форм все ще вважається вузьким місцем в області процесів моделювання методом скінченних елементів. Визначається складність генерації сіток, називаючи три основні вимоги, які необхідно виконати під час побудови трикутних сіток з довільних форм. Ці сітки повинні:

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

1. відповідають формі змодельованого об'єкта
2. складатися з елементів (трикутників або тетраедрів) потрібного розміру та форми, уникаючи при цьому перекриття фігур
3. допускати гострі кути.

Хороший огляд основ генерації сіток та різноманітних типів даних, що формують сітку, а також інших можливостей маніпулювання існуючою сіткою.

2.2 Генерація архітектурних об'єктів та структур у іграх

Процедурне моделювання будівель, як одна з найкраще розроблених галузей PCG , все ще дуже присутнє в дослідженнях і варіюється від граматик форм до підходів, керованих даними, самонавчання, або реконструкції моделей з фотографій та відео реальної архітектури. Незважаючи на багаті дослідження, існує лише кілька конкретних реалізацій.

Грубо кажучи, більшість робіт можна виділити за створенням будівель з інтер'єром та без нього. Невиключні підсфери досліджень:

- розташування кімнат на фіксованому поверсі,
- створення форми та фасаду зовнішнього вигляду будівель.

Поєднання обох аспектів, а саме створення цілих доступних будівель, обговорюється лише побіжно, як і подальші кроки, такі як розміщення дверей чи вікон або з'єднання поверхів за допомогою сходів.

Крім того, мета створення будівель для реальної симуляції віртуального світу вимагає ще одного фактора: часу. Час впливає на зовнішній вигляд будівель щодо

- вік (час будівництва, заселення, реконструкція, занепад),
- час доби або пора року (освітлені вікна та ввімкнене зовнішнє освітлення вночі або взимку, дим з димаря в холодну пору року та відкриті вікна влітку).

Спочатку ми коротко представимо пов'язані роботи про будівлі без інтер'єру, а потім розглянемо будівлі з інтер'єром.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

Будівлі без інтер'єру

Представлено граматику форм CGA *shape* для створення корпусів будівель з довільним рівнем деталізації. Їхній підхід на основі правил дозволяє послідовне моделювання мас шляхом простої адаптації відповідної граматики. Створюють наступника CGA під назвою CGA++ , який надає більше контекстної інформації про інші форми під час генерації та дозволяє редагувати цілі дерева форм або навіть створювати нові.

Використано процедурну екструзію для моделювання будівель з високою деталізацією, приймаючи контур та профіль будівлі як вхідні дані. Вони досягають вражаючих результатів, коли йдеться про моделювання дахів.

Розширено L-систему, замінюючи символи функціями, які можна виконувати в будь-який час під час процесу генерації. Оскільки ці функції можна параметризувати, користувач миттєво бачить вплив кожної зміни на результуючу архітектуру будівлі.

Розподіл простору та планування поверхів

Характеризується планування будівлі як основне завдання архітектурного проектування та наголошують на складності оптимального формування, розташування та масштабування кімнат у складній топології з багатьма проміжними зв'язками. Вони представляють агентно-орієнтовану систему пошуку топології, яка працює на регулярній сітці та підтримує еволюційну оптимізацію.

Класифікують планування архітектурних просторів як *проблему просторового розподілу*, метою якої є розташування прямокутників на площині або сітці, та датують її виникнення як допоміжного методу концептуального архітектурного проектування початком 1970-х років.

Перші спроби вирішення *проблеми просторового розподілу* оцінювали всі можливі прямокутні схеми розташування, що призвело до експоненціально зростаючої кількості шаблонів. Навіть якщо цей перший підхід вважався нездійсненним, стратегії на основі сітки залишалися в центрі уваги дослідників, використовуючи генетичні алгоритми, процедурне моделювання або

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

цілочисельне програмування.

Розглядають багатоповерхові будівлі зі сходами та наголошують на важливості розміщення сходової клітки в одному й тому ж місці на кожному поверсі. На відміну від нашої роботи, автоматизоване розміщення зовнішніх дверей та вікон відсутнє, а непрямокутні кімнати не підтримуються.

Використовують *алгоритм квадратної деревоподібної карти* для рівномірного розподілу прямокутних кімнат на прямокутному плані поверху. Вони покладаються на перевагу *квадратної деревоподібної карти* порівняно з оригінальним *алгоритмом деревоподібної карти* для оптимізації співвідношень сторін кімнат, уникаючи довгих, вузьких кімнат. Крім того, вони пропонують оптимізувати алгоритм для виявлення коридорів, що з'єднують усі кімнати на поверсі.

Як видно на рисунку 2.2, кімнати, що відображаються на квадратній деревоподібній карті (с), мають покращене співвідношення сторін між шириною та висотою порівняно з деревоподібною картою (b), що є результатом *алгоритму квадратної деревоподібної карти*. Ми припускаємо прямокутний поверх шириною 6 та довжиною 4 одиниці. Цей поверх має бути розділений на сім кімнат розміром 6, 6, 4, 3, 2, 2 та 1 квадратна одиниця. Кімнати слід розмістити таким чином, щоб кожна кімната мала співвідношення сторін (ширина/довжина), яке є максимально ближчим до 1, щоб уникнути дуже довгих та вузьких кімнат. Алгоритм *квадратної деревоподібної карти* спочатку визначає початкову половину поверху. Оскільки ширина більша за довжину, кімната розміщується в лівій половині (див. рисунок 2.2, крок 1). В іншому випадку вона була б розміщена у верхній половині. Співвідношення сторін першої кімнати становить $8/3$ (або $4/1,5$). На другому кроці наступна кімната розміщується над першою (крок 2). Її співвідношення сторін становить $3/2$, що ближче до 1, ніж до $8/3$, тому ми продовжуємо розміщувати третю кімнату над другою. Співвідношення сторін третьої кімнати погіршується до $4/1$ (крок 3), тому третя кімната переміщується на праву (вільну) половину поверху. Потім співвідношення сторін покращується до $9/4$ (або $3/1,3$) (крок 4). Кімната також переміщується, коли розміщується

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

наступна кімната розміром 3 (крок 5). Розміщення кімнати 5 поруч із кімнатою 4 призводить до гіршого співвідношення сторін (крок 6), і тому вона розміщується вище у вільному верхньому правому куті (крок 7). Кімнати 6 та 7 досягають найкращого співвідношення сторін, якщо їх розмістити поруч одна з одною (кроки 8, 9 та 10).

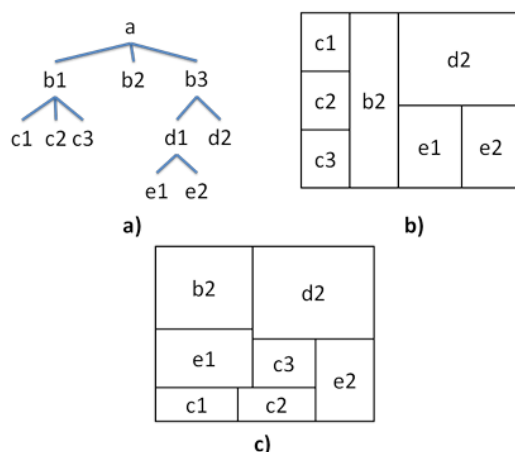


Рисунок 2.2 - Деревоподібна структура (а); деревоподібна карта (б) та квадратна деревоподібна карта (с)

На перший погляд, результат не має практичної зручності, оскільки кожна кімната з'єднана лише безпосередньо до іншої кімнати, тобто коридору немає. Пропонується удосконалення алгоритму квадратної деревоподібної карти, який здатний знаходити коридорний шлях, що з'єднує всі окремі кімнати в будівлі. На основі набору правил (ванні кімнати та спальні можуть не бути з'єднані з кухнею, спальні можуть не бути з'єднані одна з одною тощо), спочатку визначається, чи потрібен коридор. Потім автори пропонують вибрати всі внутрішні ребра (або практично стіни) на плані поверху та з'єднати їх за допомогою алгоритму найкоротшого шляху. Цей шлях потім використовується для створення коридору вздовж вибраних ребер. Шляхом зміщення та розширення шляху генерується коридор. Мірахмаді та Шамі наголошують, що будь-який згенерований коридор потребує ліній, що перекриваються з кімнатами, щоб дозволити розміщення дверей.

Ще одним відомим обмеженням алгоритму деревоподібної карти є те, що він може обробляти лише прямокутні області. Це не стосується іншого підходу, який може збільшити кімнату на будь-якій початковій області. Єдиною передумовою є те, що ця область має бути розкладена плиткою перед застосуванням алгоритму. У сітці плиток кожна кімната спочатку призначається одній клітинці, яка служить відправною точкою. Одна за одною кожна кімната збільшується в одному напрямку по черзі, доки не буде досягнуто її заздалегідь визначеного цільового розміру (рисунок 2.3, кімнати зверху ліворуч та зверху праворуч на кроках е та f). На другій ітерації всі клітинки, які ще не заповнені, призначаються пов'язаній кімнаті (рисунок 2.4, нижня кімната, крок f).

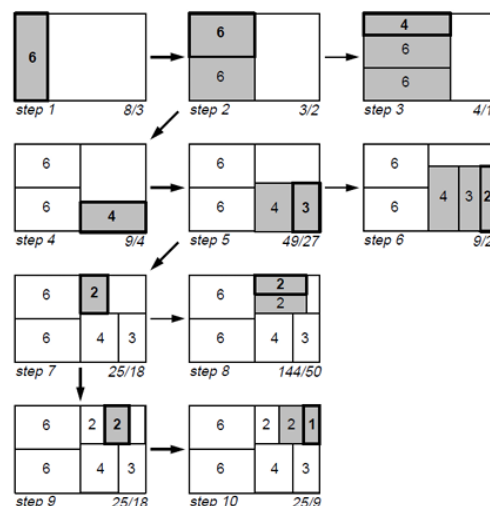


Рисунок 2.3 - Алгоритм квадратної деревоподібної карти.

Оскільки мінімального розміру комірки не існує, алгоритм можна використовувати також для заповнення, наприклад, кола або трикутника, зменшуючи розмір комірки до мінімуму. Чим менший розмір комірки, тим точніший результат для непрямокутних областей.

Оцінюють свою роботу, щоб з'ясувати, чи є комбінація методу поділу та зростання життєздатним підходом для створення різноманітних та правдоподібних планів поверхів на основі плитки .

Представляють метод упаковки n прямокутників для мінімізації їхньої

обмежувальної рамки, водночас витрачаючи мінімум місця у внутрішній частині обмежувальної рамки. Їхній алгоритм спочатку обчислює положення x усіх внутрішніх прямокутників, перш ніж призначати положення у відповідно роблячи це, вони зменшують розмірність до одиниці та застосовують один із попередніх алгоритмів обрізання втраченого простору.

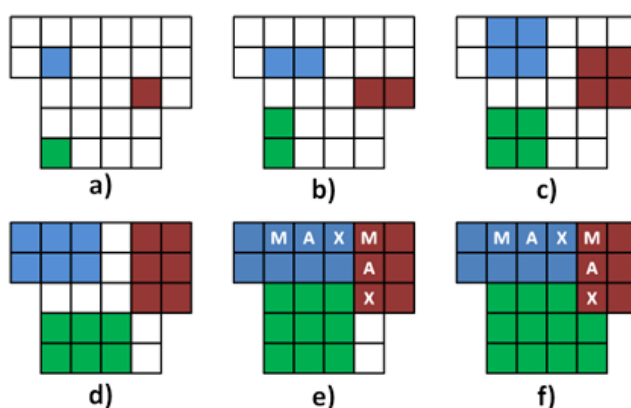


Рисунок 2.4 - Алгоритм зростання здатний заповнювати навіть непрямокутні області кімнатами на різних стадіях: (а) початкове розміщення комірок кімнати, (be) зростання кімнат, (f) з'єднання решти комірок із сусідніми кімнатами.

Хоча *прямокутне пакування* рідко згадується, коли йдеться про генерацію планів поверхів, ми згадуємо його тут, оскільки ми будемо звертатися до нього пізніше, розміщуючи кімнати на заданому плані поверху. Обговорюють різні структури для пакування планів поверхів, такі як *О-дерево* або *В*-дерево*, та порівнюють їхні можливості щодо представлення топології плану поверху.

Щодо *прямокутної упаковки*, *щільна упаковка* була розглянута та застосована і, нарешті, порівняна із поширеними алгоритмами поділу з точки зору швидкості та надійності. Згідно з їхньою роботою, розв'язувач схеми поділу має перевагу в продуктивності, тоді як щільна упаковка демонструє переваги, коли задіяна взаємодія з користувачем. Можливим наступним кроком у створенні будинків було б заповнення створених кімнат меблями та аксесуарами.

Це можна було б зробити на основі рекомендацій щодо дизайну інтер'єру,

визначаючи розміщення меблів залежно від доступності, симетрії або адаптивності до структури кімнати.

Граматики

Вонка та ін. представляють *розділену граматику* для автоматизованого моделювання архітектури. Їхня система підтримує ієрархічне визначення правил для маніпулювання заданою поверхнею. Таким чином, вони можуть створювати будівлі з високим або низьким рівнем деталізації.

Процедурні методи важко контролювати, і пропонують підхід до керування процедурними моделями на основі граматики. Їхній алгоритм оптимізує продукт граматики на основі специфікації, що надається разом із самою граматикою. Для полегшення процедурного моделювання на основі граматики Патоу пропонує візуалізувати правила граматики в *орієнтованому ациклічному графі* (DAG), щоб побачити всі залежності вздовж ланцюжка правил і, як наслідок, мати більш зручне для користувача представлення граматики. Детально розглянуто моделювання круглої геометрії та використовують різні системи координат для полегшення створення структур, таких як куполи чи вежі. Щодо створення правдоподібних фасадів, Фінкенцеллер дає вичерпний огляд різноманітних методів, що використовуються для створення фасадів за допомогою *L-систем* або *граматик форми*. Він не тільки пояснює створення дверей та вікон, але й представляє семантичну модель для орнаментів та стінових конструкцій.

2.3 Застосування нейромереж та методів машинного навчання для генерації контенту

Саммервілл та ін. звертають увагу на імпульс, який машинне навчання знову набуло в останні роки. Однак вони чітко зазначають, що PCGML все ще вимагає багато роботи та досліджень, поки генерація предметів, персонажів, правил або 3D-рівнів не буде повністю вивчена.

Застосування машинного навчання та стохастичної оптимізації для

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

створення доступних житлових будівель досліджується. Використовуючи *баєсівські мережі*, автори навчають алгоритм, використовуючи реальні архітектурні дані. Алгоритм підбирає заданий набір визначених користувачем змінних (наприклад, розмір та планування будівлі) до результуючої топології кімнат, використовуючи їхні індивідуальні розміри та суміжність кімнат. За допомогою стохастичної оптимізації план поверху генерується та оптимізується за довільну кількість ітерацій.

Застосовується *баєсівське об'єднання моделей*, техніку з обробки природної мови, до позначених навчальних фасадів. Отримана модель використовується для вибірки контекстно-вільних граматики для імітації існуючого стилю будівлі.

Шайю навчив три моделі, використовуючи генеративно-змагальні нейронні мережі (GAN), для проектування, розділення та надання планів поверхів. Використовуючи 256×256 та архітектуру генератора / дискримінатора, автор досягає хороших результатів для широкого спектру форм і стверджує, що нейронні мережі відіграватимуть певну роль у майбутньому генерації будівель.

Оцінювання процедурних генераторів

Провели дослідження користувачів, щоб дослідити реалізм процедурно згенерованих будівель, і виявили, що якість текстур, наявність нерівностей та правдоподібність структури позитивно впливають на сприйняття моделей будівель.

Загальний огляд оцінювання генераторів контенту, зосереджуючись на візуалізації діапазону результатів генерації та використанні анкет для вивчення сприйняття генераторів гравцями.

Досліджено діапазон виразності генератора процедурного рівня шляхом візуалізації його генеративного простору. Ця візуалізація дозволяє нам проаналізувати вплив різноманітних параметрів алгоритму.

Усі розглянуті нами статті представляли інноваційні ідеї та робили внесок у загальне уявлення про методи процедурної генерації моделей віртуальних будівель. Однак вони не вдаються до детального опису фактичної генерації

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

тривимірних сіток і не називають пасток, яких слід уникати після розгляду згенерованої архітектури, таких як перешкоди, оптимальний вибір та розміщення текстури, а також проблеми генерації різних рівнів деталізації.

Фасади

Тепер ми коротко обговоримо створення фасадів, яке на перший погляд здається досить простим, оскільки воно містить лише завдання зі створення стіни та додавання до неї текстури. Фінкенцеллер може не погодитися з цією точкою зору; він пропонує дуже вичерпний та компетентний огляд різноманітних методів алгоритмічного створення фасадів, таких як граматика, L-системи або граматика форм. Не тільки фасади, вікна та двері є частиною його роботи. Він радше розробляє семантичну модель для карнизів, орнаментів та стінових конструкцій (наприклад, виготовлених з випаленого каменю). Таким чином, розглядається не лише моделювання фасадів, але й процедурна генерація для них текстур - ще один величезний розділ теорії комп'ютерної графіки, який впливає на створення ландшафту, персонажів та об'єктів.

Інший підхід до створення правдоподібних зовнішніх фасадів був опублікований під назвою «Роздільна граматика». Він пропонує техніку ієрархічного поділу стіни та позначення отриманих частин, наприклад, дверей, вікна або стіни. На наступному ієрархічному рівні ці частини можна знову розділити, а також описати за допомогою граматики, щоб отримати дедалі більше деталей, чим частіше застосовується поділ. Щоб віртуальні будівлі виглядали більш реалістичними та естетичними, рекомендується широко використовувати орнаменти та декор.

Живі істоти

Генерацію живих істот можна розділити на дві частини. З одного боку, є люди, які двоногі та ходять прямо. З іншого боку, є тварини, які можуть бути як хребетними, так і безхребетними. Ці класи знову ж таки можна розділити, наприклад, на риб, птахів, комах або павукоподібних. Тут ми розглядаємо цих істот як істоти. Баррето та Рок уточнюють, що генерація статичних сіток та скінів істот включає генерацію 3D-моделей, анімацію, поведінку та звуки. Тут ми

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

зосереджуємося на створенні статичних сіток та шкінів для людей та істот.

Люди

Генерація гнучкої моделі людини, на відміну від статичної, характеризується деякими додатковими кроками, включаючи генерацію скелета, процес ригінгу та анімацію [5]. Ригінг займається призначенням кісток та суглобів кінцівкам моделі. Кожен рівень деталізації моделі вимагає окремого ригінгу. Перевага використання скелетної анімації порівняно з вершинною анімацією очевидна: після створення анімаційної послідовності (ходьба, біг, розмова тощо) її можна просто призначити сотням моделей. Рух відрізняється лише залежно від віку та статі людини. Незначні зміни (наприклад, довжина кроку, напруження тіла та повнота) можуть забезпечити індивідуальний вигляд та поведінку для кожної людини.



Рисунок 2.5 - MakeHuman дозволяє створювати окремих персонажів шляхом параметризації базової моделі.

На цьому етапі вже можна здогадатися, що можуть бути деякі обмеження, коли йдеться про питання, чи можна повністю створити віртуальний світ процедурно. З одного боку, детальну гуманоїдну модель, утворену реалістичним скелетом, тканинами шкіри та м'язами, важко створити з нуля, а з іншого боку, так само важко знайти алгоритм для генерації правдоподібного двоногого руху людини.

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

Існують інструменти для створення та налаштування універсальної моделі персонажа, такі як Autodesk Character Generator [24], MakeHuman [26] (див. рис. 2.5) або Unity Multipurpose Avatar (UMA). Використання фреймворку, що діє таким чином, може бути гарним способом швидкого створення моделей людини, і це питання варте подальших досліджень. Для досягнення вищого ступеня реалізму потрібно не лише рухати кінцівки відповідно до реального світу, але й відображати міміку та рухи губ під час взаємодії. Остання тема розглядається в пропозиції механізму перетворення тексту в мовлення, який здатний не тільки перетворювати письмовий текст на чутне мовлення, але й автоматично обчислювати відповідні рухи губ.

Істоти

Як уже згадувалося, дослідження генерації істот проводяться рідко. Видатна гра під назвою *Spore* була розроблена *Maxis*, розроблена *Віллом Райтом* та видана *EA Games* у 2008 році, представляючи ігровий процес, що показує розвиток мікроскопічного організму в високорозумну та соціальну істоту. Щоб створити величезну кількість істот різного вигляду, дизайнер Вілл Райт запропонував процедурну генерацію як засіб вирішення проблеми генерації тисяч ресурсів, які *Maxis* мала під час розробки *Sims 2*. На основі розробки для *Spore*, Хекер та ін. представили нову систему для анімації істот з невідомою формою тіла, в якій узагальнені дані про рух можна було б застосовувати під час виконання для досягнення несподіваної, але реалістичної анімації. Представлено триетапну систему:

1. користувач визначає набір змінних для покоління істот (які називаються *генами*),
2. потім інструмент перетворює ці гени у візуальну модель,
3. Модель налаштована таким чином, щоб мати скелет, готовий до анімації.

Імітований рух

Основний аспект моделювання руху у віртуальному життєвому середовищі зосереджений на візуальному враженні від життя, яке можна

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

підсумувати для мешканців та дорожнього руху. Звичайно, багато інших симуляцій можуть працювати одночасно, наприклад, економічний розвиток, розваги (спорт, телебачення), погода або еволюція істот. Контроль поведінки сотень або тисяч мешканців належить до найскладніших завдань. Деякі статті уникають обговорення ШІ (штучного інтелекту) в цьому контексті, оскільки процедури, що використовуються в іграх, зазвичай не призначені для імітації реалістичної людини. Моделювання натовпу – це окрема, добре досліджена тема в галузі моделювання великої кількості людей. Моделювання натовпу – це скоординований рух та дії кількох персонажів у заданому середовищі та в його межах.

Рух у віртуальному світі та взаємодія з ним однаково важливі. Ранні ігри з відкритим світом -ігри, в яких гравець може отримати доступ до всього світу з самого початку гри - такі як *Grand Theft Auto 3*, вводили віртуальних громадян, щоб зробити світ більш живим, але через відсутність взаємодії один з одним та відсутність їхньої індивідуальності вони виглядають трохи нудними. Кращі підходи сучасних ігор, такі як *Watch Dogs*, воліють надавати кожному NPC (неігровому персонажу) особистість, роботу та спеціальні риси характеру.

Тим не менш, такі основи, як пересування містом, увага до руху, використання тротуарів тощо, добре працюють в іграх AAA. Вони показали, що простого алгоритму пошуку шляху недостатньо, щоб поведінка виглядала реалістичною. Чотири дисципліни, які має охоплювати симулятор натовпу:

- вибір цілі,
- просторові запити,
- обчислення плану,
- адаптація плану.

У безперервному процесі встановлюються цілі кожного NPC (*Вибір цілі*), визначаються шанси на дії та рух (*Просторові запити*), а також розраховується шлях досягнення вибраної мети (*Обчислення плану*) та ітеративно адаптується після кожної дії (*Адаптація плану*). Отриману симуляцію можна покращити, враховуючи два інші фактори, які значно

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

покращують загальне враження:

- довіра,
- можливості взаємодіяти.

Фактор *достовірності* розглядається як поверхнева вимога. Його можна розділити на кілька підумов, таких як особистість, емоційність, рішучість та зовнішній вигляд. Деякі автори роблять обережні кроки до процедурної анімації. Описує свій фреймворк руху *Twig*, заснований на фізичному моделюванні, щоб мати змогу створювати рухи, такі як рух до цілі або обійми. Однак його метою є не створення реалістичної анімації, а лише тієї, яка робить персонажа живим. Представляють систему пересування для багатонігих персонажів, яка базується на алгоритмі, який розміщує сліди вздовж шляху, а потім обчислює положення ніг персонажа вздовж шляху. У своєму висновку автори наголошують, що модель персонажа, особливо апарат руху, аж до форми ніг, має бути дуже деталізованою, щоб створити правдоподібний рух.

Взаємодія з NPC

Можливість взаємодії з іншими людьми, істотами або NPC зараз детально обговорюється. Взаємодію можна розбити на вербальне та невербальне спілкування (тут порушується класична аксіома вченого-комунікаціоналіста Пола Вацлавіка, який стверджує, що не можна не спілкуватися; це насправді можливо для NPC, якщо він явно не запрограмований на це). Вона може включати двох або більше акторів. У цьому контексті не має значення, чи бере участь гравець-людина (представлений аватаром) чи ні. У вербальному спілкуванні планування та управління діалогом означають центральну проблему. Один з найпростіших та найяскравіших способів її вирішення - використання кінцевих автоматів (FSM), відомих з класичної інформатики. На рисунку 2.6 показано такий FSM, в якому NPC задає гравцеві просте запитання, а саме: «Скільки буде 2 помножити на 2?» і пропускає його, коли гравець відповідає правильно. Дається набір із двох заздалегідь визначених варіантів - 4 та *інше число*. Якщо відповідь - *інше число*, NPC повторить запитання. Якщо відповідь 4, NPC скаже гравцеві пасувати, і діалог завершиться.

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

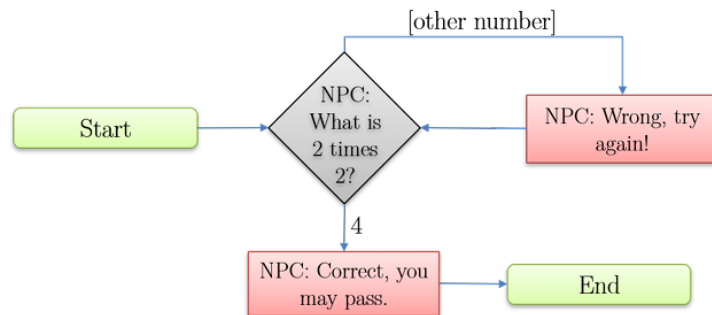


Рисунок 2.6 - Блок-схема, що ілюструє простий діалог.

Цю ідею можна посилити аспектами особистості, стосунків та настроїв. Розглянувши такі інструменти, як *articy:draft*, можна побачити, що візуальне редагування діалогів (і цілих сюжетних ліній) досягло високого рівня зрілості, оскільки Maxis' *The Sims* яскраво показує, як симулювати кількох NPC з сильним акцентом на інтерактивні аспекти. Усі персонажі в грі, включаючи тих, якими керує гравець, мають зважене відношення один до одного та індивідуальну особистість. Обмеження полягає у вербальній взаємодії, яка досі (з першого випуску гри) відбувається за допомогою псевдомови під назвою *Simlish*. Тим не менш, емоції дуже добре виражаються за допомогою інтонації, міміки та жестикуляції, що дозволяє негайно інтерпретувати настрої (див. рисунок 2.7 для проекції настроїв на модель NPC).



Рисунок 2.7 - NPC, створений у Mixamo Fuse, виражає емоції мімікою (нейтральний, злий та щасливий).

У грі The Sims обмежено певною кількістю акторів, тому доцільним є дослідження того, як масштабувати цей підхід до багатьох взаємодіючих NPC. Термін «рівень деталізації», який використовується для опису деталей 3D-моделі, також може бути застосований до області штучного інтелекту та поведінки акторів.

У цій роботі пропонується процедура, в якій різноманітні моделі поведінки структуровані ієрархічно. Ці моделі можна зменшити або розширити певними шарами, залежно від рівня деталізації. Недоліком цього підходу є додаткові витрати на моделювання відповідного дерева поведінки. Автор описує таверну як приклад, у якому бармен та кілька гостей моделюються на різних рівнях деталізації. Якщо гравець відсутній, сцена в таверні приблизно змінюється;

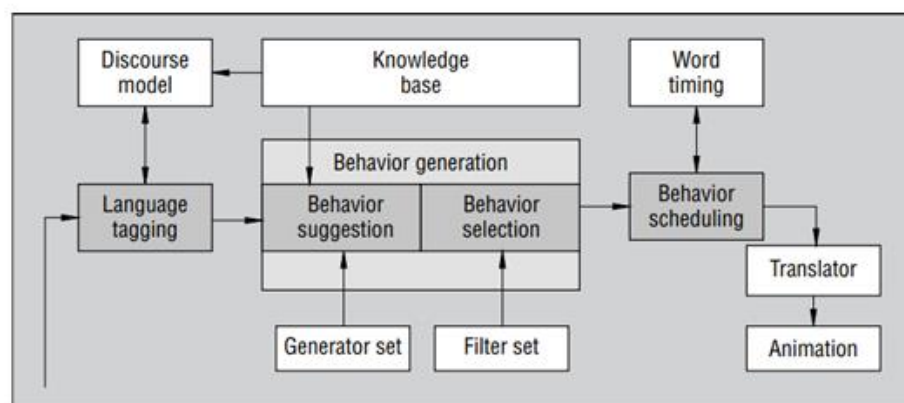


Рисунок 2.8 Інструментарій анімації вираження поведінки (BEAT) перетворює текст на невербальну поведінку.

миттєво спорожніють, гості не переходять від столу до столу, а миттєво виходять або заходять до бару. Якщо гравець заходить до таверни, рівень деталізації симуляції підвищується, і гості починають взаємодіяти один з одним, бармен пересувається та прибирає столи. Розробили інструментарій BEAT (Behavior Expression Animation Toolkit) для генерації невербальної поведінки та синтезованого мовлення для віртуального персонажа на основі друкованого тексту. Їхня розширювана система на основі правил була отримана з реальних

людських розмов (див. рисунок 2.9).

Детальніше розглянуто, як BEAT анотує текст за допомогою жестів рук, рухів очей і брів, а також інтонації, і планують ці вирази на часовій шкалі мовлення. Представляють розмовного агента, який імітує людську поведінку під час розмови. Так званий інтерфейс «Розум-Тіло» збагачує план дискурсу значенням і дозволяє проектувати відповідні емоції на тіло персонажа. Говорять про використання *реляційних агентів* для встановлення довгострокових стосунків між людиною (наприклад, гравцем) та віртуальною особистістю (наприклад, NPC). Вони зосереджуються на ситуаціях (таких як освіта чи бізнес), в яких людина взаємодіє зі своїм колегою, та надають уявлення про їхню мотивацію.

Процедурно згенеровані історії

Багато ігор базуються на історіях. Вони забезпечують природну структуру наративу, допомагаючи гравцям зрозуміти менталітет та цілі, яких потрібно досягти, встановлюючи контекст та мотивуючи гравців. Це особливо стосується рольових ігор. Гравці зазвичай мають основну сюжетну лінію, яка веде їх до мети, та додаткові побічні історії, які підвищують задоволення від гри. Прикладами таких ігор є The Elder Scrolls V: Skyrim [1] та серія The Legend of Zelda.

Процедурна генерація історій має багато переваг. По-перше, створення історій є дорогим; зазвичай задіяно багато художників, дизайнерів контенту, програмістів та звукорежисерів. Ці зусилля можна значно зменшити, якщо використовувати процедурну генерацію історій. По-друге, коли нові варіації історій створюються під час виконання гри, це може підтримувати мотивацію гравців під час повторного проходження гри. По-третє, стає можливим створювати історії, які автоматично адаптуються до навичок та вподобань гравців. Описують процес створення історії як двоетапний процес: планування послідовності подій (планування історії) та формулювання фактичного тексту, який бачить гравець (поверхнева реалізація). Фактично, складність обох етапів поділяється на багато підпроцесів, особливо за допомогою адаптивних підходів.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

Фасад, наприклад, це система, яка створює драматичну напругу шляхом об'єднання подій та ретельної реакції на дії гравця – обидва дуже детальним способом (так називають автори).

Граматики

Один з найочевидніших підходів полягає у використанні попередньо створених граматик, часто в поєднанні з вручну згенерованими структурними блоками (шаблонами). Відомим представником є *Tracery* (див. лістинг 2.1). Його основним елементом є список правил продукування, які створюють речення шляхом підстановки символів. Об'єднання цих речень призводить до створення історії. Перевагою цього підходу є те, що комп'ютер має максимальний контроль над текстом, який потрібно згенерувати. Недолік також очевидний: створення шаблонів і складних правил займає багато часу. Крім того, імітована креативність може призвести до креативності, яка відрізняється від того, чого очікує людина.

```

1 {
2   "name": [ "Arjun", "Yuuma", "Darcy" ],
3   "animal": [ "unicorn", "raven", "sparrow" ],
4   "mood": [ "vexed", "indignant", "impassioned" ],
5   "story": [ "#hero# traveled with her pet #pet#. #hero# was never #
              mood#, for the #pet# was always too #mood#." ],
6   "origin": [ "#[hero:#name#] [pet:#animal#] story#" ]
7 }
```

Лістинг 2.1: Приклад *ажурної* граматики, яка генерує речення типу «Дарсі подорожувала зі своїм домашнім єдинорогом. Дарсі ніколи не була пристрасною, бо єдиноріг завжди був надто обурений».

Алгоритми планування

Граматики можуть (але не обов'язково) йти пліч-о-пліч з алгоритмами планування. Вони генерують історії, припускаючи, що історії складаються з послідовності дій, які спрямовані на досягнення заздалегідь визначеної мети. Перший підхід полягає у використанні *мови планування*. Такі мови відомі зі

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

штучного інтелекту, наприклад, для опису рухів роботів. Прикладом є *ADL*, мова опису дій. Вона базується на діях, які мають попередню та постумову. Попередня умова має бути виконана для виконання дії, а постумова відображає її наслідки. Також дозволені кількісні змінні, тобто оператори 3 та V. Умови можуть бути сформульовані як булеві формули з операторами $\wedge$, $\vee$ та $\neg$. Підтримуються типізовані змінні. ADL припускає *відкритий світ*, тобто те, що не міститься в специфікації, є невизначеним. *Початковий стан* визначається як основа, у нашому випадку початок історії, а *цільовий стан* - як кінцевий стан історії.

2.4 Висновок по розділу

У другому розділі було досліджено методи та інструменти генерації процедурного контенту в іграх. Розглянуто методи генерації ігрових середовищ, зокрема створення ландшафтів і рівнів із використанням алгоритмічних та інтелектуальних підходів. Проаналізовано підходи до генерації архітектурних об'єктів і структур, які забезпечують різноманітність та адаптивність ігрового світу. Також досліджено застосування нейромереж і методів машинного навчання для автоматизованої генерації контенту та покращення його якості. У результаті визначено ефективні технології та інструменти створення процедурного контенту для сучасних ігрових систем

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 . РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ ТА ОЦІНКА ЕФЕКТИВНОСТІ

3.1 Реалізація генерації процедурних об'єктів (на прикладі будівель)

Багато сучасних AAA-ігор відбуваються у відкритому світі, який гравець може вільно досліджувати лише з певними обмеженнями. У таких серіях, як *Grand Theft Auto*, *The Witcher*, *Fallout*, *Assassin's Creed* або *Watch Dogs*, одне або кілька величезних міст слугують центральним місцем сюжету. Свобода, пов'язана з іграми з відкритим світом, сприймається інтенсивніше, коли гравець стикається з меншою кількістю бар'єрів, які заважають йому досліджувати навколишнє середовище. Одним з найпоширеніших типів бар'єрів у міських умовах, безумовно, є будівлі, в які гравець не може потрапити і які представлені лише фасадами. Обмежувальним фактором для того, щоб не моделювати інтер'єр кожної будівлі, часто є час і гроші. Наведемо деякі цифри: Ubisoft наймає 400-600 людей для розробки AAA-гри з відкритим світом, тоді як розробка *Grand Theft Auto IV* зайняла понад 1000 людей і три роки .

Оскільки багато громад не задоволені відсутністю доступності, потреба в більш доступному інтер'єрі обговорюється суперечливо, і деякі фанати навіть створюють модифікації, щоб відкривати замкнені двері, створювати відсутні інтер'єри або навіть додавати цілі нові райони до існуючих ігрових середовищ.

Отже, виникає питання, чого могли б досягти розробники ігор, якби будівництво будівель з правдоподібним інтер'єром було простішим, швидшим і дешевшим. Двома можливими ефектами можуть бути вищий рівень занурення завдяки більшій реалістичності та меншій кількості бар'єрів або збільшення реіграбельності, оскільки нові області можна було б створювати знову і знову - навіть під час гри.

У цьому розділі ми представляємо робочий процес для процедурної генерації 3D-моделей будівель з акцентом на зручність використання та реалізм. Нашим основним внеском є створення доступних багатоповерхових будівель з

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

різними типами сходів та випадковим, але розумним текстуруванням поверхонь будівель. Для оздоблення будівель після процесу генерації ми впроваджуємо систему відстеження вільних просторів у кімнатах та вздовж стін кімнат. Процедурна складова цієї роботи виражається алгоритмом, який створює будівлю на основі заданого плану, який може бути полігоном з отворами або без них.

Головною перевагою можливості створювати будівлі для заданого полігону є те, що алгоритм можна застосовувати до даних ГІС, що є поширеним способом зберігання планів поверхів будівлі (див. рис. 3.1). Це дозволяє розробникам ігор миттєво створювати віртуальні карти з реальних географічних місць, що останнім часом стає дедалі популярнішим в іграх.



Рисунок 3.1 - Інститут комп'ютерних наук та математики у форматі Open Street Мар та відповідне представлення XML OSM.

Модель даних та топологія

Тепер ми визначаємо модель даних для наших алгоритмів (див. рисунок 3.2). Будівля містить кілька кімнат, кілька сходів і принаймні один дах. Текстури визначаються набором *текстур кімнати*. Для формування кімнати потрібно щонайменше три стіни. Кімната може містити кілька *сходових зон* і *зайнятих зон*. Обидва визначають простір, який блокує розміщення інших об'єктів. Стіна кімнати має від нуля до n дверей і вікон.

Модель даних на рисунку 3.2 не представляє вичерпної геопросторової

					БР.ІП - 72.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

топології; наприклад, вона не містить інформації про те, чи є дві кімнати суміжними одна з одною, чи вони перекривають одна одну у двох накладених поверхах. Ця топологія неявно виводиться під час процесу генерації для перевірки просторових відносин, коли це необхідно. Рішення не відстежувати топологію будівлі в моделі даних ґрунтується на простоті визначення залежностей з моделі даних (наприклад, якщо дві кімнати розташовані поруч або з'єднані дверима). Це зменшує складність та схильність до помилок під час моніторингу дійсності, стану та обмежень топології під час кожної зміни архітектури будівлі.

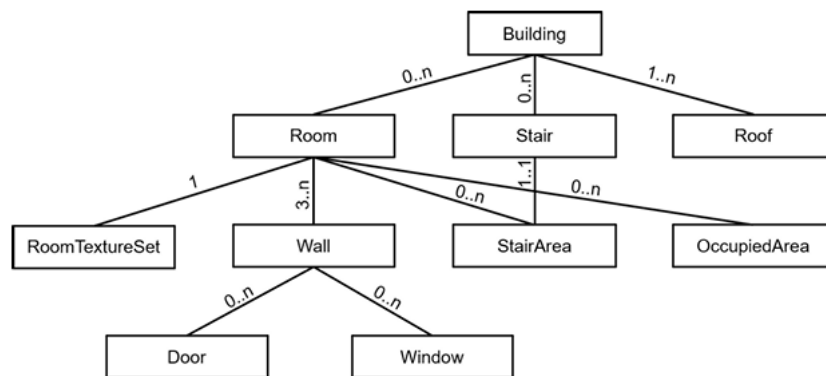


Рисунок 3.2: Модель даних будівлі.

Генерація процедурних будівель

Процес генерації будівлі можна розділити на *створення екземплярів об'єктної моделі* та *генерацію 3D-моделі*. Метою розділення обох є виключення всіх випадкових операцій з генерації 3D-моделі, а отже, гарантування відтворюваності для кожної процедурно згенерованої будівлі. Крім того, процедурну частину можна замінити іншою, а генератор сітки можна використовувати без будь-яких змін.

1. *Створення екземпляра об'єктної моделі* : Модель даних, розраховується з використанням усіх розмірів, планування та обмежень будівлі. Користувач може впливати на модель шляхом параметризації. Оскільки детермінований екземпляр містить випадкові фактори, він не є відтворюваним і

може бути успішним після того, як раніше зазнав невдачі з ідентичними параметрами, визначеними користувачем.

2. *Генерація 3D-моделі* : Модель об'єкта перетворюється на 3D-модель за умови, що вона пройшла перевірку. Цей процес повністю автоматичний і на нього можна вплинути лише шляхом зміни базової *моделі об'єкта* .

3. Основна увага наших досліджень в галузі PCG зосереджена на створенні екземплярів об'єктної моделі та відповідній валідації та обчисленню. Хоча ми вважаємо перетворення перевірених координат у 3D-модель тривіальним, ми обговорюємо деякі подібні аспекти генерації моделі, такі як рівень деталізації, текстуровання та маніпулювання полігонами, щоб вказати на переваги та недоліки.

Створення екземплярів об'єктної моделі

Створення екземпляра об'єктної моделі – це послідовний процес із дванадцяти кроків (див. рис. 3.3). Наш алгоритм має п'ять обов'язкових вхідних параметрів :

- ширина будівлі (з плаваючою поверхнею),
- довжина будівлі (з плаваючою точкою),
- форма будівлі (замкнутий полігон),
- кількість кімнат (ціле число),
- кількість поверхів (ціле число).

Параметри мають бути зрозумілими без тлумачення. У застосунку, де генерується багато будівель, можна розрахувати кількість кімнат і поверхів на основі розмірів будівель, щоб мінімізувати необхідну взаємодію з користувачем.

Вибір форми будівлі

Коли йдеться про житлові будинки, то зустрічаються три поширені форми будівель: прямокутна, Т-подібна та L-подібна. Використання параметризованої ширини та довжини дозволяє нам генерувати варіації цих трьох форм для невеликої різноманітності. Тим не менш, не випереджаючи оцінку, ми виявили, що хоча така параметризація основних форм була можливою, частково випадкова форма будівлі призведе до більш цікавої, привабливої та

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

дослідницької будівлі. На додаток до цих найпоширеніших форм плану поверху, представлений тут метод підтримує будь-які можливі замкнуті форми.

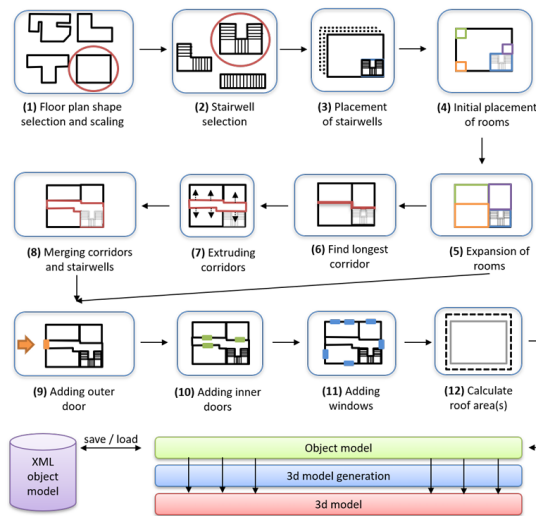


Рисунок 3.3 - Створення екземпляра об'єктної моделі створює перевірений опис будівлі, який пізніше перетворюється на 3D-модель.

Вибір та розміщення сходової клітки

Враховуючи, що будівля має кілька поверхів, було обрано тип сходів. Для нашого прототипу було реалізовано три типи: L-подібні, U-подібні та прямі сходи. Розташування сходових кліток у багатоповерховому будинку є складним завданням. Воно передбачає оптимальне розміщення одночасно на всіх поверхах, а також створення окремої сітки сходових кліток. Сортування поверхів за поверхами допомагає перевірити, чи дійсно вони висхідні, і чи немає пропущених рівнів між ними. Перетин полігону верхнього поверху з полігоном нижнього визначає спільні області, які розглядаються як можлива область розміщення сходів. Цей перетин послідовно обчислюється від верхнього поверху до нижнього поверху, доки на кожному поверсі не залишиться n областей (див. червону область на рисунку 3.4 як приклад).

На наступному кроці розраховується необхідна площа для сходової клітки. Її розмір залежить від типу сходів, висоти кімнати та кількості сходинок. Щоб уникнути блокування стіною на вході та виході зі сходів, додається верхній

та нижній сходові майданчики.

Розміщення полігону сходової клітки в полігоні поверху досягається шляхом накладання контуру сходової клітки та поверху. Полігон сходової клітки обертається доти, доки

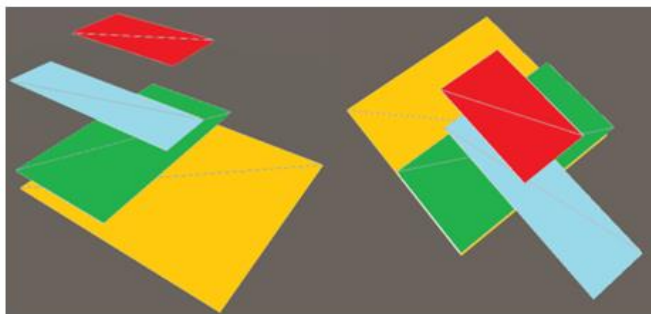


Рисунок 3.4: Розрахунок площі перетину (червоний полігон зверху) трьох полігонів (жовтого, зеленого, синього).

Щоб виміряти якість знайденого положення, довжини ребер, спільних для полігону сходової клітки та кімнати, підсумовуються. Гарне розміщення призводить до високого значення перекриття. Ми називаємо цей метод *найкращим наближенням*.

На рисунку 3.5 описано приклад розміщення прямокутних сходів у багатокутному приміщенні. Алгоритм оцінює багато інших поворотів та положень цих сходових кліток, але для спрощення ми включимо лише сім, показаних тут.

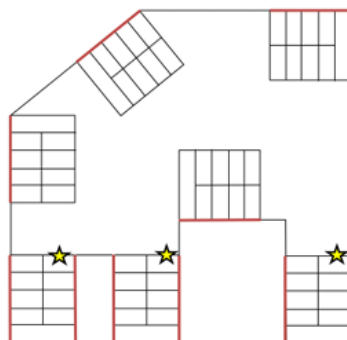


Рисунок 3.5: Оптимальне розташування U-подібних сходів у полігоні

кімнати. Три позиції сходів у нижній частині кімнати будуть оцінені як високі, оскільки значення перекриття як кімнати, так і країв сходової клітки є найвищими (див. червоні лінії). Після розрахунку форми сходів, країв поручнів, сегментів входу та виходу, а також оптимального положення, сходи реєструються для кожної накладеної кімнати.

Найкраще оцінені позиції (найвище перекриття країв) позначені зірочкою.

Початкове розміщення кімнат

Після розміщення сходових кліток ми використовуємо метод групування, щоб розподілити всі кімнати по всіх поверхах, щоб мати максимально рівну кількість кімнат на поверсі. Наш алгоритм базується на зростанні регіонів. Створення зразкової будівлі з трьома поверхами а чотирнадцять кімнат призводять до розподілу 5, 5 та 4 кімнат відповідно. Ці кімнати масштабуються до невеликого початкового розміру 1x1 одиниця та розподіляються порівну на кожному поверсі за допомогою методу найкращого наближення. Тут ми стикаємося з двома проблемами відповідного алгоритму:

- 1) Як можна розмістити n багатокутників форми A в іншому багатокутнику B , якщо B має менше вершин, ніж n ?
- 2) Як ми можемо гарантувати, що не всі n багатокутників форми A розміщені в одній щільній області багатокутника B ?

Пункт 1) вирішується шляхом поділу ребер зовнішнього полігону на кілька підребер, так що вершин завжди буде більше або стільки ж, скільки й кімнат. У прикладі на рисунку 3.6а у нас є вісім кімнат, які потрібно розмістити в полігоні з чотирма вершинами. Отже, ми поділяємо полігон, додаючи одне додаткове ребро на кожне існуюче ребро, так що загальна кількість вершин для цього полігону дорівнює восьми (див. червоні точки на рисунку 3.6b). Поділ - це проста, покрокова інтерполяція між початковою та кінцевою точками ребер.

Пункт 2) вирішується в одному єдиному рішенні, яке ми називаємо Найкраще пристосування за найбільшою відстанню. Як відомо з базового методу найкращого пристосування, всі можливі розташування кімнат зберігаються у списку. Це дозволяє нам побачити всі можливі області розміщення кімнат у

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

формі будівлі. На наступному кроці ми беремо середню точку кімнати та визначаємо n положень кімнат з усіх можливих положень з найбільшою відстанню одна до одної.

Знаходження найбільшої відстані в дослідженнях розглядається як *задача найдальших сусідів*. Через її складність ми обрали достатній алгоритм апроксимації, який є швидким і добре працює.

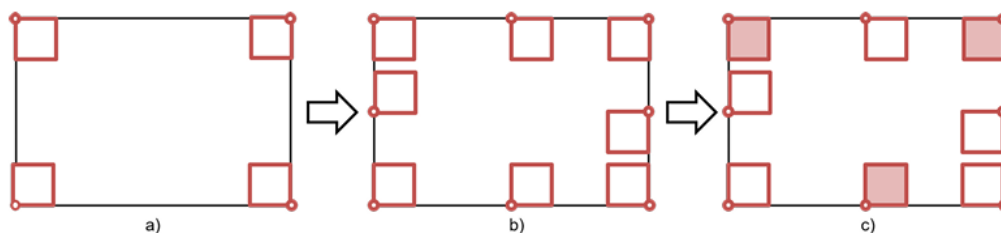


Рисунок 3.6: Додавання більшої кількості вершин до полігону без зміни його форми

Додавання більшої кількості вершин до полігону без зміни його форми дозволяє розмістити більше внутрішніх полігонів за допомогою методу найкращого наближення. Тут чотири вершини (а) перетворюються на вісім шляхом одноразового розділення кожного ребра (b). З усіх дійсних позицій полігону, що повертаються алгоритмами найкращого наближення, вибираються три, які знаходяться найдалше одна від одної (c).

Розширення кімнат

Коли кожна з кімнат містить початкову частину плану поверху, ми розширюємо їх за допомогою кругового підходу вздовж послідовності $A^{\wedge}B^{\wedge}C^{\wedge}A^{\wedge}...$. Алгоритм зростання розширює вершини полігону в кожному напрямку, доки вони не перетнуться із зовнішньою межею (формою будівлі) або з іншим полігоном (іншою кімнатою). Коли вершина досягає зовнішньої межі, полігон все ще можна розширити в усіх інших напрямках. Навпаки, перетин з іншим полігоном, який одночасно розширюється в межах тієї ж форми, завершує алгоритм, щоб уникнути поганих співвідношень сторін та примусово створити

більше неправильних форм шляхом об'єднання решти артефактів (див. наступний абзац). Приклад розширення показано на рисунку 3.7а, b та с.

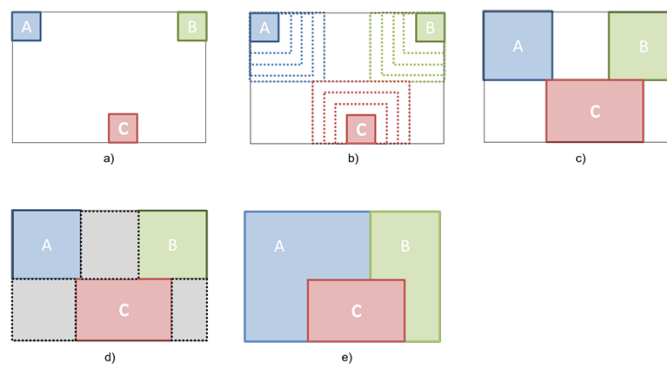


Рисунок 3.7: Полігони А, В та С поетапно розширюються (пунктирні лінії), доки вони не перетинаються з іншими полігонами. Решта областей об'єднуються з кімнатами А, В та С.

Якщо жодну з кімнат більше не можна розширити, алгоритм зростання досяг свого кінцевого стану. Оптимально, ширина та висота будівлі кратні розміру кроку, щоб алгоритм міг використовувати весь простір у зовнішньому полігоні. В іншому випадку, у зовнішньому полігоні можуть бути невеликі, невикористані області, які призводять до небажаних артефактів у згенерованій сітці. Після розширення невикористані області залишаються на поверсі та можуть бути зображені як сірі прямокутники на рисунку 3.7d. Ці області можна витягти, віднявши всі форми кімнат з плану поверху. Отримані форми потім з'єднуються з першою сусідньою кімнатою у списку кімнат на поверсі. Цей підхід подібний до підходу, який використовує сітку та розширює кімнати клітинка за клітинкою. Використовуючи вершини як відповідні точки для зростання кожної кімнати, ми урізноманітнюємо процедуру та результат. Крім того, оскільки наш підхід базується на полігонах та булевих операціях, ми можемо обробляти непрямокутні форми будівель.

Отримані полігони кімнат забезпечуються індексом поверху та зберігаються як полігони у списку кімнат. Щоб зробити форму кімнати цікавішою, її кути можна вигнути. Інтерполяція між середніми точками обох

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

стін, що утворюють кут, дозволяє визначити параметризовану кількість стін, що представляють арку. Щоб створити таку криву, ми створюємо список ребер, намальованих між середньою точкою двох сусідніх ребер. Деталізація та плавність арки зростають зі збільшенням кількості ребер. Зрештою, алгоритм проходить через усі середні точки ребра та створює нові ребра стіни з поточної та останньої пари середніх точок. На рисунку 3.8 показано, як кут двох ребер можна заокруглити та замінити списком нових ребер. Сегменти арки знову можна розглядати як прості стіни та обладнати вікнами та дверима.

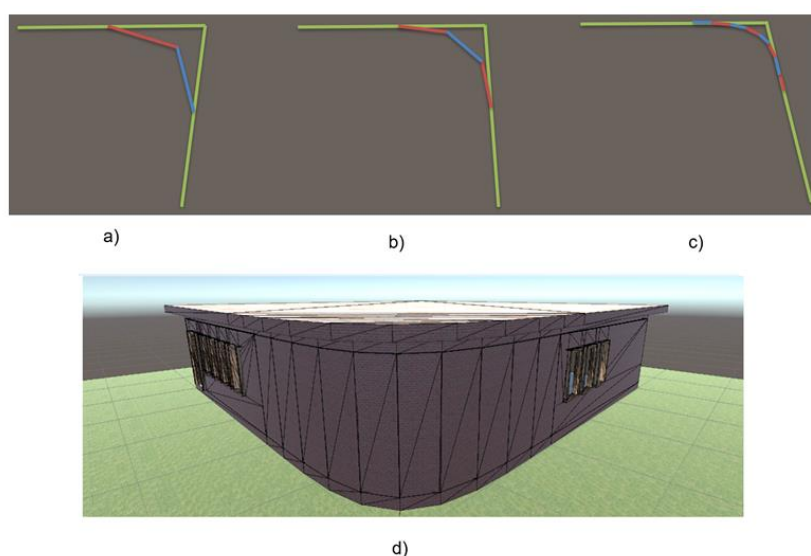


Рисунок 3.8: Три форми арок з (а) двома ітераціями, (б) трьома ітераціями та (с) десятима ітераціями. Зелені лінії – це початкові ребра, заокруглені алгоритмом. (d) показує приклад будівлі з арочним кутом.

Текстурування

Щоб застосувати текстури до поверхонь кімнати, ми вводимо *набір текстур кімнати* (див. рисунок 3.8). Ця структура даних містить вісім різних текстур, по одній для кожної поверхні кімнати, як показано на рисунку 3.8.

Область *вирізання сходів* та область *вирізання вікна* посилаються на текстуру для тих областей, які створюються, коли блок вирізання віднімає частини сітки кімнати для дверей, вікна або сходів. Кожну текстуру в *наборі текстур кімнати* можна явно визначити, але для досягнення різноманітного

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

вигляду згенерованої будівлі ми вводимо інший метод текстуровання. Цей метод підтримує три рівні гранулярності, що дозволяє додавати коефіцієнт рандомізації до вибору текстур. Кожна текстура індексується шестизначним числом (див. приклад на рисунку 3.9) та назвою, що стосується фактичного файлу текстури. Перші дві цифри її номера стосуються типу поверхні, наприклад, *стелі*. Другі дві цифри ідентифікують другий рівень деталізації, такий як *внутрішня стеля* та *зовнішнє вигнання*. Останні дві цифри визначають точну текстуру. Якщо цифри не вказані, алгоритм вибирає випадкову текстуру в заданій області видимості (наприклад, *стеля* або *внутрішня стеля*). Тому ініціалізація *набору текстур кімнати* чотиризначними ідентифікаторами (див. другий рівень на рисунку 3.8) призведе до кімнат з різними текстурами на ітерацію.

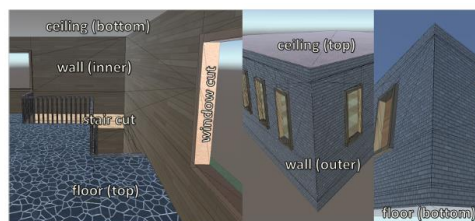


Figure 4.9: Surfaces of a room texture set mapped to the faces of a mesh.

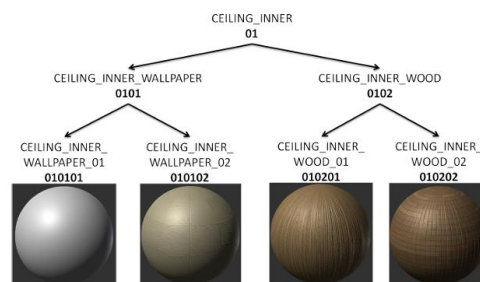


Рисунок 3.9 - Зернистість типу поверхні додає випадковий фактор або вибирає текстуру бетону.

Додавання дверей та вікон

Додавання вікон та дверей відбувається у суворому порядку. Пріоритет надається зовнішнім дверям, далі йдуть внутрішні двері та вікна.

Орієнтуючись на європейський архітектурний стандарт, алгоритм прагне розмістити головні входні двері в коридорі. Додавши просте правило, що

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволяє головний вхід у вітальні, ми можемо легко змінити планування на американський стиль будівлі.

У реальних будівлях розташування зовнішніх дверей залежить від багатьох факторів, наприклад, рельєфу місцевості навколо будівлі, району або типу кімнат. Оскільки це не входить до наших можливостей, ми створюємо лише одні двері на першому поверсі, де коридор торкається зовнішньої стіни. Якщо такої стіни немає, вхід створюється на сходовій клітці або, як другий варіант, у найбільшій кімнаті будівлі, якою зазвичай є вітальня.

Розташування внутрішніх дверей досягається шляхом рекурсивного відстеження того, чи кімната прямо чи опосередковано (через n кімнат) з'єднана з коридором або з кімнатою із зовнішніми дверима. Якщо це не так, алгоритм з'єднує цю кімнату із сусідньою кімнатою, яка ще не має з'єднання з цією кімнатою. Потім алгоритм продовжує роботу з наступними кімнатами, доки всі кімнати не стануть доступними.

Кількість вікон розраховується на основі розміру кімнати, використовуючи фіксовану кількість на квадратний метр. Розташування дверей та вікон у стінах залежить від топології будівлі, відповідно від розташування, розміру та суміжності її кімнат. Загалом, слід розрізняти зовнішні двері та зовнішні вікна, відповідно внутрішні двері та внутрішні вікна (наприклад, службовий люк між кухнею та їдальнею). Кожне з останніх з'єднує дві кімнати, і, отже, необхідно перевірити, чи обидві кімнати мають спільну стіну, і що стіна не перекрита перешкодою (наприклад, сходами). Зовнішні кімнати та вікна потребують перевірки лише за наявності існуючого вікна або дверей на цільовій стіні.

Щоб запропонувати динамічне позиціонування дверей та вікон, ми вводимо правила розміщення: «Перший», «Останній», «Найкращий» та «Випадковий». Ці правила застосовуються, коли алгоритм розміщення вибирає вільні, спільні області на краях двох суміжних кімнат, які мають спільні двері або вікно.

- *Перша* – перша вільна область, яку визначає алгоритм.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

- *Остання* – остання вільна область, визначена алгоритмом.
- *Найкращий* – область з найкращим розміром (мінімальний надлишок).
- *Випадковий* – випадкова область.

Алгоритм проходить по всіх сфокусованих стінах та збирає вільні сегменти довжиною, більшою за ширину вікна та/або дверей. Використання стратегій, що передбачають порядок (*Перша* та *Остання*), вимагає суворого порядку стін для проходження. Цей порядок задається послідовним індексом стіни (0, 1, 2...) та вимогою, щоб стіни завжди визначалися за годинниковою стрілкою. Таким чином, правило «*Перша*» поверне першу вільну область для першої спільної стіни, спрямовану від початкової точки стіни до кінцевої точки стіни, наприклад, стіна 0 кімнати 1 та стіна 1 кімнати 2 на рисунку 3.10. Відповідно, правило «*Остання*» поверне останню вільну область на останній спільній стіні (наприклад, стіна 3 кімнати 1 та стіна 2 кімнати 2).

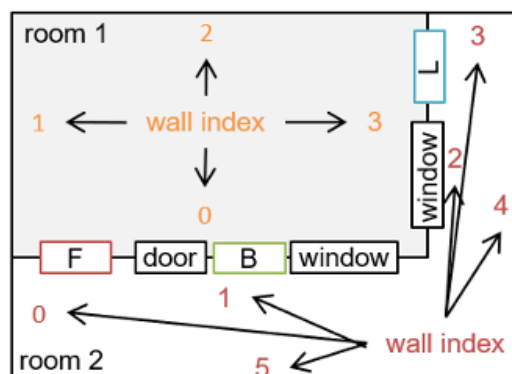


Рисунок 3.10 - Розміщення нових дверей або вікна за правилом Перший (F), Найкращий (B) та Останній (L). Випадковий вибір може повернути будь-який із трьох зразків розміщених об'єктів.

Оскільки ширина сегмента може бути більшою за об'єкт, який потрібно розмістити, ми дозволяємо вирівнювати об'єкт у межах сегмента ліворуч, посередині або праворуч.

Механізм, який використовується для перевірки наявності спільних стін

у двох кімнатах, полягає у порівнянні кожного краю стіни кожної кімнати один з одним та обчисленні їхнього взаємного перекриття. Результатом функції спільної стіни є список країв, що визначає спільні та вільні області кімнат 1 та 2.

Оскільки вікна та двері зберігаються у прямокутній формі, параметризація ширини, висоти та вертикальних зсувів є простою справою.

Пам'ятаючи, що розташування вікон і дверей впливає не лише на конструкцію будівлі, а й на інтер'єр, ми відстежуємо простір перед і позаду кожного об'єкта за допомогою так званої *зайнятої площі*, яка запам'ятовується на рівні кімнати (див. виділені об'єкти на рисунку 3.11). Ці площі активно використовуються, коли йдеться про розміщення об'єктів сходів у кімнатах або коли будівля облаштована за наступним процедурним алгоритмом. *Займана площа* представлена двовимірним прямокутником, який дозволяє їй адаптуватися до ширини вікна або дверей.

Якщо *зайнята область* новоствореного об'єкта перетинається з існуючою *зайнятою областю* в цій або сусідній кімнаті, це призведе до анулювання розміщення. Наприклад, *зайняті області* уникають дверей, розміщених у стінах з гострими кутами, або дверей, прикритих сходовими клітками.

Тепер ми обговоримо розташування вікон та дверей на корпусі будівлі. Як зазначають, існує суттєва різниця між сприйняттям вікон, вирівняних вздовж усіх поверхів, та випадковим розподілом. На додаток до вільного розташування вікон, ми вводимо підхід *клонування фасаду*. Після створення на першому поверсі вікна можна клонувати до всіх кімнат на верхніх поверхах, уніфікуючи зовнішній вигляд та структуру зовнішній вигляд будівлі. На рисунку 3.11 показано порівняння випадкового розташування вікон та *клонування фасаду*.

Концепція LOD включає кілька етапів оцінки якості тривимірної моделі з метою оптимізації використання пам'яті під час процесу рендерингу. Існує кілька факторів, що впливають на LOD, такі як відстань об'єкта до глядача, швидкість його руху та його важливість у сцені. Найбільш впливовим фактором, що сприяє різниці в якості, є кількість вершин сітки, але є й інші фактори, такі як роздільна здатність текстур та складність шейдерів. Наголошують на корисності LOD у

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

тривимірному моделюванні міст та надають детальну категоризацію LOD у мові розмітки CityGML. Виходячи з цього визначення, ми вводимо три рівні LOD у нашій реалізації для контролю якості згенерованих будівель, які приблизно відповідають LOD 2-4 CityGML:

- 1) LOD 0 - будівля з інтер'єром, перекриттями та сходами,
- 2) LOD 1 - корпус будівлі із зовнішніми деталями (двері та вікна),
- 3) LOD 2 – корпус будівлі з даховими конструкціями та простими текстурами. Сітки та текстури стін та вікон опущені.

Процедура генерації сітки будівлі відповідає за генерацію кожного окремого рівня та включає реалізацію для обробки кожного рівня деталізації окремо (див. рисунок 3.11).

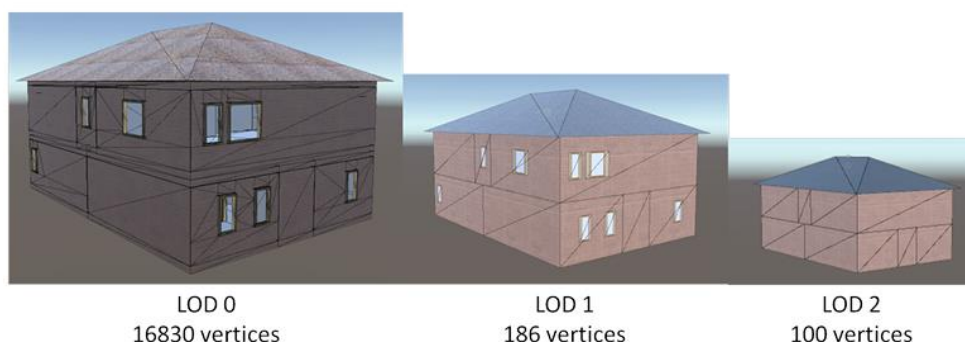


Рисунок 3.11 - Одна й та сама будівля згенерована у трьох різних рівнях деталізації (LOD).

3.2 Генерація інтерактивних історій та сценаріїв із використанням нейромереж

Створення систем генерування історій – це давня галузь у сфері процедурної генерації контенту для ігор (PCG-G). Система генерування історій розроблена для створення зв'язних, достовірних та драматично змістовних наративів. Це завдання, яке навіть автори-люди час від часу не можуть виконати, часто через обмежений бюджет та велике робоче навантаження. Ці дві останні перешкоди є поширеною причиною для дослідження процедурного сторітелінгу,

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

навіть якщо аспект тиску в часі завжди обговорюється. Це корисно не лише для основного сюжету гри, але й для підсюжетів. Таким чином, автори зберігають суверенітет над основною сюжетною лінією, щоб хід гри залишався відомим для всієї команди розробників.

Ще одним вагомим аргументом на користь процедурного сторітеллінгу є креативність, яку генерує алгоритм. Люди часто пишуть про свій особистий досвід або змінюють історії, які вони самі прочитали. На противагу цьому, алгоритм, заснований на величезному наборі доступної літератури та мовних моделей, черпає інформацію з майже нескінченного пулу правил, можливостей, фразувань та зв'язків. Тож замість того, щоб відображати думки, упередження, смаки чи настрої одного автора, алгоритм повертає суміш своїх навчальних даних. Однак навіть вони можуть бути упередженими та порушувати етичні принципи.

Використання історій у сфері ігор, незалежно від того, чи написані вони від руки, чи створені, є різноманітним: деякі ігри мають на меті лише базову історію для передачі настрою та обстановки, щоб гравець міг поставити себе на місце головного героя. Інші живуть з різноманітних, розгалужених історій, які враховують рішення гравців і навіть можуть мати різні кінцівки. Загалом ми визначили п'ять різних класів ігор, що характеризуються характером їхнього сюжету:

- ігри без сюжету (наприклад, *Tetris* , *Rollercoaster Tycoon*),
- ігри з простим сюжетом для передачі настрою та оточення (наприклад, *Need for Speed*, *Street Fighter II*),
- ігри зі складними, але фіксованими сюжетами (наприклад, *Final Fantasy VII*, *Legend of Zelda*),
- ігри з інтерактивними сюжетами, які передають відчуття, що гравець може впливати на хід гри (наприклад, *Borderlands*, *Mass Effect*),
- ігри з інтерактивними історіями, на хід і кінцівку яких гравець може активно впливати (наприклад, *Life is Strange*, *Detroit: Become Human*).

Ця робота зосереджена на останній категорії, а саме на генерації

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерактивних історій, на які гравець може впливати під час гри. Наша мотивація полягає в тому, що текстові пригоди зараз переживають відродження завдяки генерації за допомогою мовних моделей. Подібно до *AI Dungeon 2* та *GPT-2 Adventure*, ми використовуємо статистичні мовні моделі для генерації унікальної історії з кожним новим запуском генератора.

Ми відрізняємо наш підхід від вищезгаданих тим, що пропонуємо розумний набір дій, з яких гравець може вибрати для продовження історії, подібно до відомих *дієслів* у традиційних пригодницьких іграх LucasArts (наприклад, *Day of the Tentacle*). Представляючи три методи генерації дій, ми порівнюємо результати з точки зору креативності, керованості, узгодженості, настрою та різноманітності.

Крім того, ми оцінюємо два процеси генерації історій з точки зору доцільності та продуктивності: один, у якому генератор спонтанно реагує на попередньо виконану дію, та інший, у якому всі дії та їх наслідки попередньо розраховуються. Ми розглядаємо, як контролювати розвиток історій, керуючи станом, прогресом та інвентарем поза генератором, а також розміщуючи попередньо підготовлені текстові модулі у згенерованих абзацах, щоб зберегти об'єкти, місця та людей у сюжетній лінії. Ми не будемо зосередитися на перенавчанні мовної моделі під генератором, що було зроблено Волтоном та Вітмором і виявилось цінним для вільного введення тексту в пригодницьких жанрах, таких як «*взяти меч*» або «*відчинити двері*».

Гібридний підхід мовних моделей та управління станом

У нашому підході ми використовуємо творчі можливості мовних моделей та додаємо елементи керування, щоб дозволити гравцеві взаємодіяти з генератором історій. У наступних розділах ми розглянемо три фази: ініціалізацію, виконання та завершення, як показано на рисунку 3.12. Згодом ми обговорюємо проблеми узгодженості, генерації дій та продуктивності генератора історій. Ініціалізація мовної моделі та шаблонів.

Фаза ініціалізації – це двоетапний процес. Модель завантажується перед запуском генератора (крок 1 на рисунку 3.12).

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

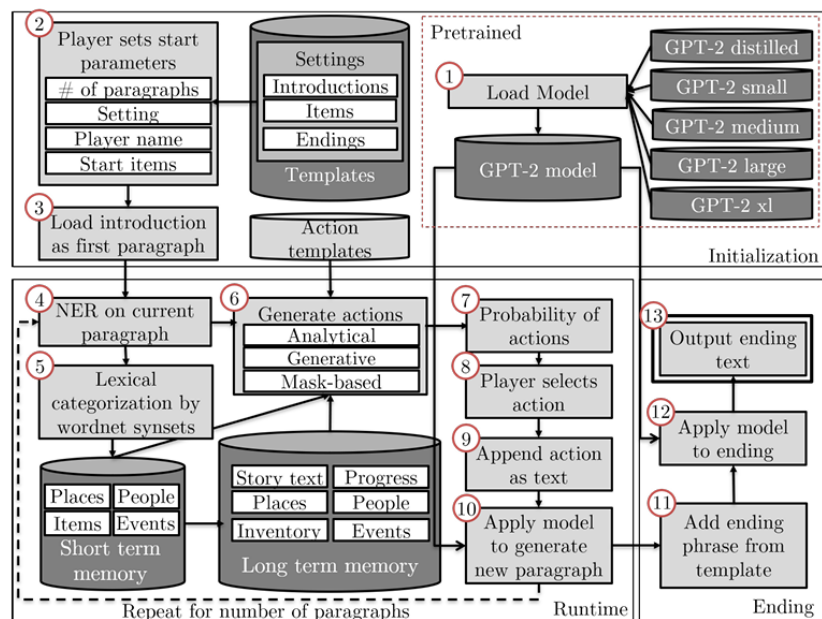


Рисунок 3.12 - Гібридний підхід до генерації історій поділено на три кроки: ініціалізація, виконання та завершення.

Гравець може вибирати між різними варіантами моделі (див. таблицю 3.1), які головним чином визначаються їхнім розміром, що, у свою чергу, залежить від кількості шарів та параметрів нейронної мережі. Моделі будуть розглянуті пізніше ще раз щодо різноманітності та швидкості застосування.

Таблиця 3.1

Попередньо навчені моделі GPT-2.

Модель	Кількість шарів (Layers)	Приховані шари (Hidden layers)	Параметри (млн)
DistilGPT-2	6	768	81,9
GPT-2 small	12	768	117
GPT-2 medium	24	1024	345
GPT-2 large	36	1280	774

На другому кроці фази ініціалізації (крок 2 на рисунку 3.12) гравець вибирає різні параметри, такі як кількість абзаців, які має містити оповідь, ім'я гравця, необов'язкових членів групи та набір предметів для гри. Після цього один

з кількох можливих вступів завантажується з об'єкта налаштувань та представляється гравцеві (крок 3 на рисунку 3.12).

Оскільки вступ є основою історії, він повинен містити якомога більше інформації про персонажів, місця та предмети в історії. Тільки за наявності достатньо великої кількості інформації мовна модель може встановити контекст і згодом достовірно продовжити історію. Тут ми представляємо двох членів групи головної героїні, ім'я яких гравець може вільно обрати і які супроводжуватимуть її під час подорожі через повторювані згадки. Окрім друку вступу як першого абзацу, предмети гри додаються до видимого інвентарю, щоб гравець знав, що він носить із собою.

Хід сюжету

Фаза виконання – це повторюваний процес, який триває, доки гравець не завершить певну кількість абзаців на свій вибір. Вона починається з розпізнавання іменованих сутностей (NER) в останньому абзаці – це вступ у першому запуску. Сутності не тільки розпізнаються, але й категоризуються за допомогою *WordNet Synsets*, щоб вибрати групу іменників, з якими гравець може фізично взаємодіяти (кроки 4 та 5). Наприклад, ці категорії можуть бути

- іменник.тварина (іменники, що позначають тварин),
- іменник.артефакт (іменники, що позначають об'єкти, створені людиною),
- іменники, що позначають їжу та напої (іменники, що позначають їжу та напої),
- іменник.рослина (іменники, що позначають рослини), та
- іменник.об'єкт (іменники, що позначають природні об'єкти (не створені людиною)).

Це дозволяє уникнути пізніших дій, таких як «Візьми любов» або «Поговори з вірою», під час генерації дій. Такі дії можуть здаватися креативними в деяких місцях, але частіше вони порушують хід гри, оскільки є заплутаними або недоречними.

Ці об'єкти зберігаються в *короткочасній пам'яті* з життєвим циклом, що

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

охоплює поточний абзац. Вони також переносяться в *довготривалу пам'ять* для подальшого використання. Це дозволяє генератору пов'язувати минулі події або минулі розмови, а отже, додавати контролю та узгодженості історіям. Розрізнення короткочасної та довготривалої пам'яті дозволяє ігровому механізму з часом забувати об'єкти меншої важливості.

Уникнення надмірності сутностей

Сутності можуть зустрічатися в абзаці кілька разів з різними іменами. Процес NER виявить вирази «Єлизавета» та «Королева» з речення «Єлизавета II королева Сполученого Королівства». Для людини зрозуміло, що обидва терміни відносяться до однієї й тієї ж особи, але для компонента NER це не так. Таким чином, генератор історій розглядатиме «Єлизавета» та «Королева» як окремі сутності та може одночасно генерувати дії «Поговорити з королевою» та «Поговорити з Єлизаветою». Це призводить до невідповідності в історії та порушує хід читання, тому вирази, що відносяться до однієї сутності, повинні бути зведені до одного виразу. Ми використовуємо *розв'язання кореференцій* для виявлення таких виразів та зберігаємо перший знайдений вираз, відкидаючи всі інші. У перших оцінках ми спробували *Spacy Knowledgebase* для *розв'язання кореференцій*, яка добре працювала для простих та правдоподібних текстів. Але оскільки наш генератор історій часто створює сценарії, які не завжди мають ці дві характеристики, або тому, що з'являються невідомі імена, або кореференції не завжди вказані, база знань була надійною лише в кількох випадках. Зрештою, ми використали реалізацію *Hugging Face*, вона базується на векторах слів та нейронних мережах і надійніше виявляє кореференції.

Генерація дій

Як згадувалося у вступі, ми реалізували три методи для генерації дій, які гравець може використовувати для продовження історії (див. рисунок 3.13 та крок 6 на рисунку 3.12). Така дія є не лише дієсловом та об'єктом, але й має кілька атрибутів, як показано в Лістингу 3.2. Тип *повідомляє* про іменований тип сутності, *дія* - про тип дії, *речення* - про повне речення на основі шаблону, яке додається до історії, якщо гравець вибирає цю дію, *просте речення* містить спрощену версію

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

шаблонного речення, а ймовірність - це ймовірність речення, обчислена за допомогою базової мовної моделі.

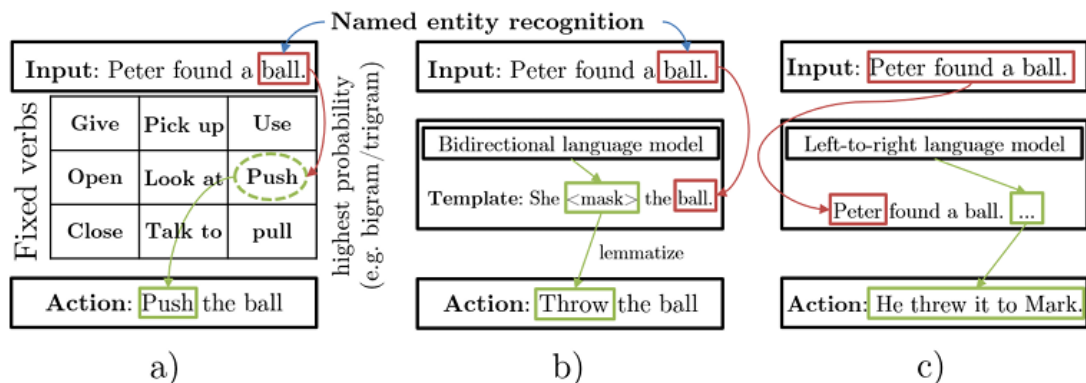


Рисунок 3.13 - Три різні способи запропонувати фіксовані дії для продовження історії: аналітичний (а), масковий (б) та генеративний підхід (с).

Підхід а) на рисунку 3.13 використовує НЛП для аналізу попереднього абзацу історії та пропозиції осмислених дій, таких як *взяти*, *підібрати*, *відвідати*, *використати* або *натиснути* зі статичного списку в поєднанні з предметом, людиною або місцем.

```
1 { "name": "potion", "type": "item", "action": "use", "sentence": "Peter bravely used the potion without hesitation.", "simple": "Peter used the potion.", "probability": 0.76 }
```

Лістинг 3.2 - Об'єкт дії «potion» з його окремими властивостями

Оскільки дії походять від розпізнаних сутностей, абзац з багатьма сутностями може призвести до однаково великої кількості дій. Тому корисно пріоритезувати дії. Ми робимо це на кроці 7, перевіряючи дії на правдоподібність. Ця перевірка обчислює ймовірність виникнення комбінації дії та об'єкта, про який йде мова. Було розглянуто кілька підходів. Спочатку ми оцінили ngrams (бі- та триграми), щоб обчислити ймовірність p дії, наприклад, «*Пітер використав зілля*» буде набагато імовірніше, ніж «*Пітер*

розмовляє з зіллям». Навчена модель ngram базувалася на корпусі Reuters, що містив 10 788 новинних документів з 1,3 мільйонами слів. Однак у багатьох випадках навіть прості комбінації, такі як « бере яблуко», не могли бути знайдені в нашому згенерованому списку триграм. Отже, ми оцінили семантичну подібність у другу чергу, використовуючи вектори слів для обчислення ймовірностей. Тут результат також був незадовільним, оскільки подібність часто не могла дати жодної інформації про зв'язок між дієсловами та іменниками. Наприклад, поєднання «штовхати» та « любити» було оцінено вище, ніж «брати» та «яблуко». Розрахунок ймовірності на основі моделі GPT-2 зрештою приніс переконливі результати. Обчислення втрати як помилки моделі для заданого речення допомагає з'ясувати, яке речення з більшою ймовірністю знайдеться в навчальному корпусі GPT-2. Ми помітили, що використання простого речення, зведеного до підмета, присудка та додатка, і таким чином лінгвістично нормалізованого, забезпечує кращу порівнянну ймовірність, ніж прикрашені речення на основі шаблонів.

Підхід b) також базується на вилучених сутностях попереднього абзацу. Тут просте речення утворюється з комбінації особового займенника та додатка, у якому позиція дієслова маскується. Використання двонаправленої мовної моделі (наприклад, BERT) дозволяє замінити цей заповнювач статистично найбільш обґрунтованим дієсловом. Показано послідовності, які заповнюють замасковане поле у виразах «He <mask>the ball» та «She <mask>the ball». Хоча не можна явно вимагати використання дієслова для маски, лінгвістично рідко існує інший варіант для моделі. Виділяються дві особливості: по-перше, обраний особовий займенник відіграє певну роль у виборі дієслова. Таким чином, ймовірності відрізняються для *he* та *she* як першого слова в замаскованій фразі. По-друге, може статися, що те саме дієслово використовується знову в іншому часі. Тому рекомендується лематизація дієслів. Тут чітко видно гендерне упередження, яке базується на даних навчання мовної моделі. Бордія та Боумен стверджують, що це упередження може навіть посилюватися в моделях, і пропонують методи вимірювання та мінімізації цього упередження.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

Ймовірність, яка використовується для визначення найкращого дієслова, визначається шляхом розв'язання замаскованого слова, щоб його не доводилося обчислювати вручну, як це робиться в підході а). Контроль над предметами під час використання дій «use», «take» та «combine» можна підтримувати за допомогою WordNet Synsets та перевірки вибраного дієслова на наявність синонімів. Якщо дієслово «take» або є синонімом, об'єкт можна додати до інвентарю гравця. Використання предмета обробляється аналогічно. Поєднання предметів є окремим випадком; його також можна досягти, маскуючи цільовий об'єкт у реченні. Поєднанню завжди передують два об'єкти в інвентарі. Маскування використовується для визначення статистично найбільш ймовірного результату поєднання за допомогою BERT.

Впровадження, спостереження та вдосконалення

У цьому розділі основна увага приділяється спостереженням, які ми зробили під час впровадження та тестування нашого підходу. Ми обговорюємо технічну реалізацію, продуктивність, різноманітність персонажів, узгодженість та узгодженість настроїв. Показуються результуючі потенціали для покращення та обговорюється їх реалізація. Також буде представлено приклад результату створення історії після всіх удосконалень.

Технічна реалізація та виконання

Ми виконали дві окремі реалізації для створення історії на основі мовних моделей. З одного боку, було використано Twine, інструмент, який дозволяє писати інтерактивні, нелінійні історії та робити їх доступними для гравців на веб-сайті. Під час використання Twine важливо, щоб усі гілки історії були попередньо обчислені, оскільки гра компілюється і тому не може бути оновлена під час гри. Перевага очевидна: весь розрахунок можна виконати на відповідній системі, а згенеровану історію можна відтворити на будь-якому пристрої. За бажанням гру можна передати далі, і можна уникнути обмеження, яке полягає в тому, що мовні моделі зазвичай не забезпечують відтворюваного результату. Недоліком є те, що всі шляхи історії повинні бути згенеровані, незалежно від того, чи виконує їх гравець чи ні. Якщо гравцеві пропонується лише одна дія,

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

необхідно обчислити загалом $\text{num_paragraphs} = \text{абзаци глибини історії}$. Якщо пропонується більше однієї дії, num_paragraphs дорівнює:

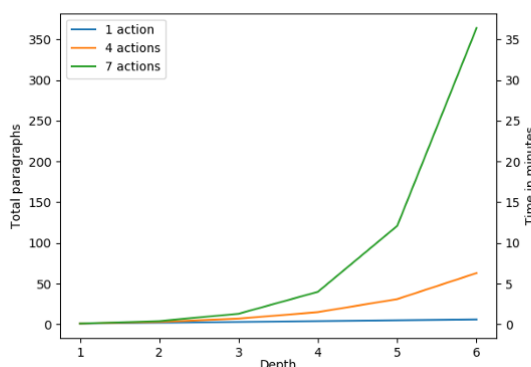


Рисунок 3.14 - Кількість абзаців для сюжетів з однією, чотирма та сімома діями.

Глибина зростає експоненціально, тоді як обсяг лінійного сюжету (з однією дією) залишається незмінним. Змінна story_depth відповідає кількості абзаців, які гравець бачить, щоб дійти до кінця історії. Показує експоненціальний характер розрахунку при збільшенні можливих дій на абзац. При вимірюванні на процесорі Intel i7 та відеокарті GeForce GTX 1080 TI з 11 ГБ оперативної пам'яті генерація одного абзацу займала приблизно 10 секунд (включаючи попередню обробку NLP, застосування шаблонів та генерацію нових текстів; на основі малої моделі GPT-2). На рисунку 3.15 показано зростання кількості абзаців та відповідний час генерації (див. вісь у праворуч).

Друга реалізація реагує на введення користувача та обчислює наступні абзаци та дії на льоту, під час гри. Оскільки в цьому підході немає попереднього обчислення можливих наступних абзаців, глибина історії відповідає кількості абзаців, які потрібно обчислити. Це усуває експоненціальне зростання попереднього обчислення. Недоліком є те, що пристрій, на якому відтворюється історія, повинен мати можливість генерувати текст у режимі реального часу за допомогою мовної моделі на основі нейронної мережі. Навіть за наявності відповідного обладнання генерація нового абзацу займає деякий час, що може порушити хід ігрового процесу. Крім того, ця вимога запобігає можливості гри на мобільних пристроях, якщо немає постійного підключення до відповідного

хмарного сервісу. З цих двох причин ми відносимо наші подальші розгляди до попередньо обчисленого варіанту.

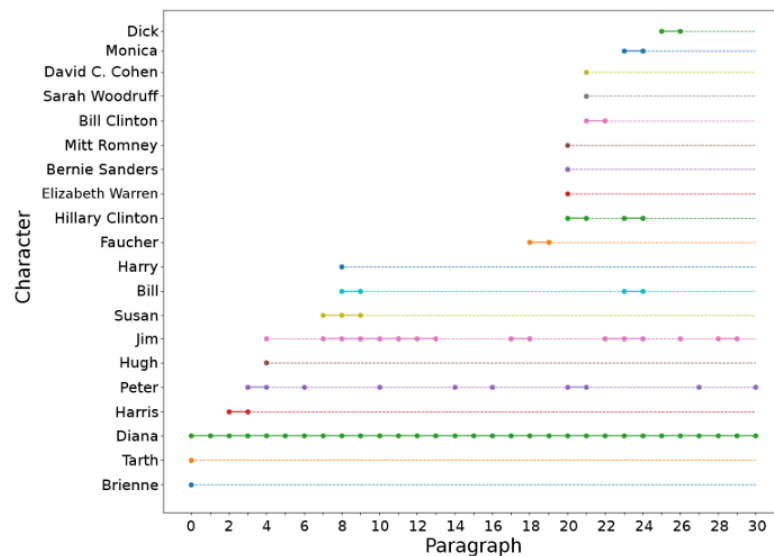


Рисунок 3.15 Майже в кожному новому абзаці з'являється новий персонаж, окрім головного героя та двох членів групи.

Різноманітність персонажів

Ми спостерігали, що кількість персонажів збільшується з кількістю програних абзаців. Персонажі, що повторюються, трапляються надзвичайно рідко, що негативно впливає на контекст історії. Таким чином, між головним героєм та антагоністом немає довгострокових стосунків, тривалих любовних романів чи дружби. У нашому прикладі, за винятком головного героя, названого в кожному абзаці, та двох необов'язково визначених членів групи, неможливо побудувати жодних стосунків з іншим персонажем, оскільки вони зазвичай мають тривалість життя лише кілька абзаців. Той факт, що люди забуваються під час гри, пов'язаний з тим, що мовна модель ніколи не приймає всю історію як вхідні дані, а лише останні n речень. Звичайно, цей обмежений вхідний дані не включає всіх осіб, які з'явилися досі.

Швидко стало очевидним, що кількість задіяних персонажів потрібно контролювати, щоб зберегти хід історії в межах. Для цього було зроблено просту заміну нових імен у тексті вже відомими, виключаючи імена головного героя та

його двох супутників. Було подбано про те, щоб замінити їх на ту саму статтю, щоб особові займенники продовжували відповідати статі людини. Визначення статі здійснюється бібліотекою вгадування статі. Також було помічено, що коли історія починається з уривку з відомої літератури, персонажі з книги з'являються. З одного боку, це призводить до цікавої історії, з іншого боку, це не продовжує час, протягом якого персонаж з'являється в історії.

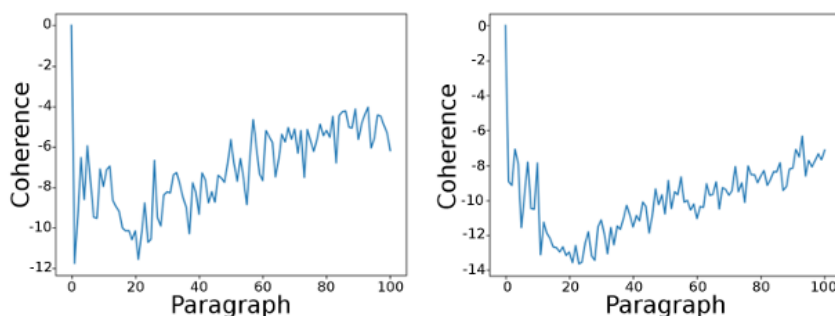


Рисунок 3.16 - Історія зі 100 абзаців з (ліворуч) та без (праворуч) елементів, що підтримують зв'язність.

Узгодженість

Одна з найбільших проблем автоматичного створення текстових пригод полягає в тому, щоб зробити контекст історії чітко видимим і зберегти його в усіх розділах. Хоча результати створення текстів за допомогою мовних моделей на основі трансформаторів самі по собі є дуже переконливими, при створенні довгих текстів бракує «червоної нитки». Завдяки заходам, таким як регулярне найменування супутників або предметів, які гравець носить у своєму інвентарі, спостерігався позитивний вплив на зв'язність [32]. У наступному списку наведено деякі приклади наших самостійно написаних контрольних речень:

- [людина] закотила [свої] очі.
- [людина] прокашлявся.
- [людина] глибоко вдихнула.

Ми виміряли когерентність, застосувавши чотириетапний конвеєр когерентності тем, представлений Родером, Ботом та Хіннебургом. Вони

розділили набір слів на пари слів. На основі довідкового корпусу обчислюються ймовірності слів. Потім обчислюється повна узгодженість усіх пар за допомогою нормалізованої точкової взаємної інформації (NPMI). Всі ці бали агрегуються (середнє арифметичне) до кінцевого значення когерентності u_{Mass} з діапазоном $-14 < x < 14$ [31]. Ми використовуємо набір слів з десяти тем, витягнутих з відповідних текстів на основі латентного розподілу Діріхле (LDA). Щоб визначити прийнятну кількість тем, ми розраховували когерентність підготовленого тексту для різної кількості тем. Найвища когерентність та найбільш значущі теми були виявлені для кількості з десяти тем. На рисунку 3.17 показано порівняння історії з (ліворуч) та без (праворуч) супутників імен та елементів інвентарю. Це показує, що когерентність можна дещо покращити, перемішуючи існуючу інформацію про осіб та інвентар. Це можна пояснити тим, що LDA виділяє осіб та об'єкти як теми та розпізнає їх пізніше в наступних абзацах.

На перцептивному рівні також було помічено, що зв'язність не завжди є найважливішим фактором, коли йдеться про розважальну цінність гри. Часто саме ігрова механіка або гумористичний елемент викликають позитивну реакцію гравців. Тим не менш, заходи, що підвищують зв'язність історії, важливі для підвищення загального сприйняття процедурно згенерованої історії.

Узгодження почуттів

Ми також спостерігали за взаємозв'язком між позитивним і негативним настроями в окремих абзацах за допомогою аналізу настроїв і помітили, що настроїв не залишається однаковим протягом певної кількості абзаців, а змінюється без розбору². Таким чином, ми ввели контрольні слова, такі як «на щастя» або «на жаль», як додаткові вхідні дані для генератора тексту.

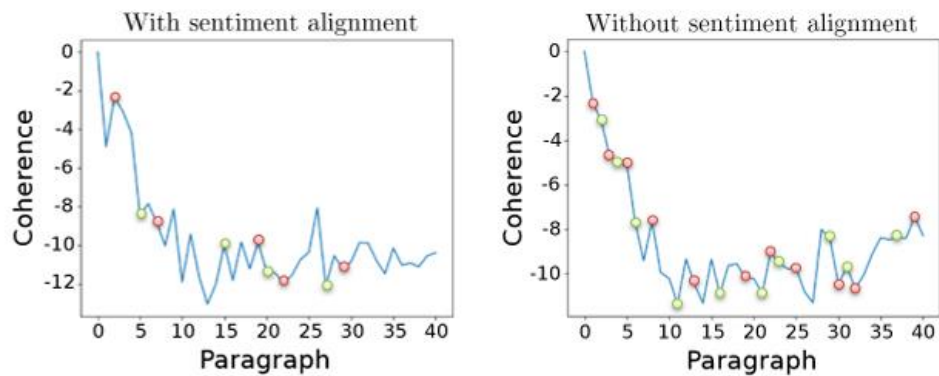


Рисунок 3.17 Історія з 40 абзаців з вирівнюванням за настроєм та без нього. Червоні крапки означають негативну зміну настрою, зелені крапки – позитивну.

На рисунку 3.17 показано, що нам вдалося вплинути на настрій деяких абзаців за допомогою контрольних слів. У нашому прикладі настрої змінився не 31 раз (ліворуч), а 19 разів (праворуч). З іншого боку, це також показує, що цей інструмент, здається, не має жодного впливу на когерентність, оскільки два верхні графіки схожі.

Ми оцінили три різні підходи до побудови дій, які гравець може обрати для продовження історії. *Аналітичний* та *масковий* підходи спрямовані на генерування конкретних дій. *Генеративний* підхід створює найбільш доречне речення, яке продовжує історію, не обов'язково маючи на увазі дію. У таблиці 3.2 наведено огляд спостережуваних характеристик кожного методу. Аналітичний підхід, безумовно, є тим, який можна і потрібно контролювати найбільше. Він дозволяє більш цілеспрямовану взаємодію з людьми та об'єктами [33]. Частково дії стають неправдоподібними, оскільки можуть відбуватися взаємодії з нематеріальними об'єктами, такими як *прийняття кохання*. Також дії часто повторюються, тому в реалізації було збережено список уже згенерованих дій, що дозволяє приховати вже запропоновані дії. Повторювані дії не були негативно помічені при використанні підходів на основі масок та генеративних підходів.

У наших тестах підхід *на основі масок* пропонував хороший баланс між керованістю та різноманітністю. Генеративний підхід, як правило, створює

нарратив з текстової пригоди, який більше схожий на роман, ніж на гру.

Дії, в яких можна було підняти предмети, були дуже рідкісними в наших тестах. Це призводить до ще меншої ймовірності поєднання двох об'єктів. Як згадувалося, синоніми до дієслова *take* використовувалися для збільшення частоти такої дії. Використання корпусу wordnet виявилось складним, оскільки синоніми до *take* можуть мати інші значення, ніж взяття предметів. З цієї причини було створено та використано керований статичний список відповідних синонімів.

Таблиця 3.2

Три підходи до динамічного формування дій.

Характеристика	Аналітичний (Analytical)	На основі масок (Mask-based)	Генеративний (Generative)
Керованість (Controllability)	Висока	Середня	Низька
Різноманітність (Diversity)	Низька	Середня	Висока
Інтерактивність (Interactivity)	Висока	Висока	Низька
Складність (Complexity)	Висока	Низька	Низька
Автентичність (Authenticity)	Середня	Висока	Висока

Різні мовні моделі

Ми проаналізували різні мовні моделі з Таблиці 3.1 на предмет їхньої узгодженості, використовуючи описану вище Узгодженість Тематичної Моделі. У кожному випадку розглядалися історії з однією можливою дією та довжиною сто абзаців. На рисунку 3.10, подібно до попередніх рисунків, описано контекст тем, витягнутих з абзаців за допомогою LDA. Розгляд здійснюється цілісно, а не послідовно з двох послідовних абзаців. Це показує, що GPT-2 small досягає найвищої середньої узгодженості. Моделі GPT-2 medium, large та xl досить близькі одна до одної, а модель DistilGPT-2 досягає помітно нижчої узгодженості у згенерованих прикладах історій. Це можна пояснити тим, що GPT-2 small здатна встановити логічний зв'язок між абзацами (що, очевидно, складно для DistilGPT-2), але через свою обмежену область охоплення вона частіше має

справу з одними й тими ж темами. Різноманітність змісту medium, large та xl, здається, призводить до того, що моделі схильні часто змінювати теми [34].

3.3 Оцінка якості згенерованого контенту та перспективи розвитку

Ми представили різні нові методи в галузі генерації процедурного контенту для полегшення створення віртуальних світів. Генерація процедурного контенту була представлена в розділі наступним чином:

Процедурна генерація контенту - це автоматичне створення цифрових ресурсів для ігор, симуляцій або фільмів на основі заздалегідь визначених алгоритмів і шаблонів, що потребують мінімального втручання користувача.

Зокрема, ми зробили свій внесок у розробку дорожньої мережі, створення доступних 3D-будівель та створення інтерактивних історій для ігор. Цей внесок підтримує процес розробки ігор двома способами: з одного боку, вони заощаджують час розробки, автоматично генеруючи велику кількість артефактів. З іншого боку, вони розширюють творчий спектр розробників, створюючи неупереджений контент, який виходить за межі людської уяви.

Наш підхід до генерації довільних дорожніх мереж, показує, що генерація сітки без ручного втручання можлива та може бути використана на практиці. Результат виглядає приємно та може бути розширений багаторазовими способами, наприклад, шляхом інтеграції генератора для тротуарів, ділянок, включаючи будинки або промислові будівлі, або мостів. Інтерфейс запропонованого генератора може бути використаний для генерації чисто випадкових дорожніх мереж, дорожніх мереж одним із різних методів, тобто растровим, радіальним, манхеттенським, розгалуженим, або навіть з реальних даних ГІС, експортованих з *openstreetmaps.org* або *Google Maps*. Експериментальна генерація деяких вулиць показує, що генератор не робить кількісну оцінку згенерованого вмісту, якщо він явно не запрограмований на це. Час виконання для 236 заданих доріг та 205 похідних перехресть становив близько 7 хвилин на процесорі i7 з частотою 3,4 ГГц. Тому ми рекомендуємо

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

зменшити обсяг вхідних даних, наприклад, об'єднавши сусідні перехрестя в одне або видаливши менші сегменти дороги [35]. Альтернативою генерації дорожніх мереж під час проектування є потокова або виконавча генерація, в якій генеруються лише ті частини дорожньої мережі, що знаходяться в полі зору гравця. Ми також змогли визначити текстурівання доріг як додаткову сферу досліджень. Дорога, наприклад, має не лише текстуру з двома смугами руху, але теоретично може мати будь-яку кількість смуг, бічну смугу або різну розмітку. Щоб не зберігати текстуру для кожного типу, очевидною є генерація за допомогою шейдера.

У розділі показано наш підхід до створення тривимірних текстурованих моделей будівель. Представлений метод гарантує, що до кожної кімнати можна дістатися через коридор, а до кожного поверху можна дістатися унікальними сходами. Порівняно з генерацією дорожньої мережі, він розширює логіку простої генерації сітки за допомогою різних обмежень, які забезпечують достовірність згенерованої будівлі. Ці обмеження походять з області планування поверхів та розміщення об'єктів у заданому просторі. Булеві оператори як для сіток, так і для полігонів виявилися особливо корисними, оскільки правила створення будівлі можна формалізувати подібно до математичного рівняння. Таким чином, секція дверей відповідає простому відніманню двох сіток замість ручного обчислення набору координат. Генерація UV-карт є серйозною проблемою, яку ми вирішили, ввівши кортежі групи індексів/сіток [36]. Ці значення допомогли відстежувати текстури в процедурному процесі генерації.

У розділі ми представили підхід, який генерує інтерактивні історії за допомогою методів класичного та нейронного мережевого NLP. Аналізується початковий вступ, наданий користувачем, а такі сутності, як об'єкти, місця та персонажі, витягуються та класифікуються за допомогою *WordNet Synsets*. Потім ми пропонуємо три підходи для генерації вибраних дій для гравця, щоб написати речення, яке продовжує історію. Включаючи витягнуті елементи, місця та персонажів з поточного уривку, дії тісно пов'язані з історією. Було зазначено відмінності між трьома підходами до генерації цих дій, а саме: достовірність,

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

різноманітність та керованість. Найкращих результатів було досягнуто шляхом побудови дій з конструкції суб'єкт-предикат-об'єкт, в якій предикат маскується та передбачається мовною моделлю. Ми реалізували міру когерентності, використовуючи LDA та когерентність тематичної моделі, щоб поточну когерентність усієї історії можна було спостерігати в кожному новому уривку. Нам вдалося підвищити когерентність історії, використовуючи довгострокову та короткострокову пам'ять, яка утримує персонажів та предмети в наративному циклі, щоб гравець міг створити довгостроковий зв'язок з об'єктами, місцями та персонажами.

Щоб підтвердити функціональність наших підходів, кожен метод був повністю реалізований та оцінений. Було показано, що сучасні інструменти розробки ігор спрощують експериментування з генеративними ідеями з різних областей [37]. Сучасні ігрові двигуни, наприклад, підтримують вбудовування генераторів у робочий процес дизайнерів-людей та дозволяють швидко оцінювати, допомагаючи вирішити, чи використовувати генератор для процесу розробки чи ні.

Крім того, було показано, що додавання механізмів керування та обробка всіх особливих випадків є дорогим з точки зору розробки генератора, а також тестування його функціональності. Взявши за приклад генератор будівель, було проведено тестові запуски, в яких будівля з десятима кімнатами мала поміститися на площі 10 м². Реалізація перевірок правдоподібності для генератора займає багато часу та може обмежити креативність (хто знає, чи знайде генератор спосіб відобразити стільки кімнат на такій малій площі?).

І навіть якщо генератор не дасть бажаного результату, цілком можливо, що він принаймні послужить стимулом для людської творчості. Наприклад, поверхні будівлі можуть не ідеально вписуватися в стиль гри, але текстури можна адаптувати пізніше, щоб створити гармонійну загальну картину. Або ж згенерована історія дещо абсурдна, але може бути використана як натхнення для професійного письменника.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

Майбутня робота

Під час дослідження літератури та впровадження окремих внесків стало зрозуміло, що ПХГ є настільки великою галуззю, що її можна поширити на будь-яку область на будь-якій глибині.

Наприклад, у контексті процедурної генерації будівель ми розглядаємо автоматизоване розміщення меблів, семантичне розташування кімнат та більш точне освітлення скляними елементами або лампами як цікаві напрямки для досягнення привабливого вигляду повністю обладнаної будівлі. Хоча будівлі, згенеровані нашим підходом, різноманітні, додаткові декорації, орнаменти або деталі можуть бути використані в майбутньому для створення ще більш індивідуального вигляду.

Завдання тут полягає не лише в реалізації всіх систем, але й у їх гармонізації та поєднанні таким чином, щоб вони створювали ціле середовище процедурно. Прикладом може бути 2D Jump 'n Run зі згенерованими рівнями, згенерованими моделями персонажів, включаючи анімацію, та процедурною музикою, яка підтримує гру контекстно-залежним слуховим способом.

У контексті тривимірного віртуального світу, як це обговорювалося в цій роботі, будівельними блоками були б рельєф місцевості, дорожня мережа, детектор ділянок та генератор будівель; доповнені розміщенням мешканців та адекватною симуляцією їхнього повсякденного життя. Щоб забезпечити це для якомога більшої кількості жанрів та ігрових механік, формальна мова для генераторів є незамінною не лише для тривіальних вимог, таких як налаштування пропорцій 3D-моделей, але й для гармонізації інтелектуального твердження, сеттингу чи художнього стилю. Розробка такої формальної мови була б складною з кількох причин:

- визначення інтерфейсів для підключення окремих генераторів,
- однозначна класифікація генератора та визначення його чіткого завдання,
- гарантувати розширюваність мови та
- та забезпечення правильної послідовності виконання окремих

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

генераторів під час створення віртуального світу.

Таке формальне визначення процесу генерації мало б позитивний ефект, що багато розробників могли б одночасно працювати над генератором процедурного світу, оцінювати свої ідеї, використовуючи вже перевірені методи інших, та розширювати загальну систему, коли вони могли б перевірити свій новий внесок. Вплив такої технології міг би бути величезним, якщо вона практично доповнить існуючий процес проектування в розробці ігор.

Будь-яка робота, яку виконують машини, повинна полегшувати роботу людини, а не замінювати її. Однак, результати впровадження нової технології важко передбачити, і це може створити страх у людей бути заміненими. На щастя, попередні розробки процедурних генераторів не показали жодних негативних наслідків. Навпаки, генератори були сприйняті позитивно, як показує зростаюче використання, наприклад, Houdini. Тим не менш, багато інструментів все ще потребують участі дизайнерів для досягнення бажаного результату.

Генерація процедурного контенту за допомогою машинного навчання (PCGML) – це ще одна галузь, яка зараз переживає великий ентузіазм у дослідженнях і, безумовно, продовжуватиме це робити в майбутньому.

Навчання на основі існуючих даних вже потрапило в ігрову індустрію, як продемонстрував стартап *modl.ai*, який, окрім використання машинного навчання для створення цифрового контенту, також використовує методи штучного інтелекту для тестування ігор або виявлення використання незаконного програмного забезпечення для читів [38]. Перевага традиційних генераторів на основі правил полягає в тому, що вони можуть генерувати велику кількість ресурсів, які, у свою чергу, формують основу для навчання моделі машинного навчання. Під час реалізації генератора будівель ми згенерували близько 30 гігабайт 3D-моделей, які можна було б використовувати для такого навчання. Цікаве питання тут полягає в тому, що саме слід генерувати: плани поверхів, які генератор потім перетворює на 3D-моделі? Чи безпосередню структуру FBX-файлу? Facebook нещодавно опублікував *PyTorchSD*, фреймворк, який здатний застосовувати глибоке навчання до 3D-сіток,

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

наприклад, для завдань деформації або рендерингу.

Під час застосування будь-якої форми машинного навчання до процедурної генерації фундаментальне питання полягає в тому, чи можна параметризувати генератор PCGML так само, як і генератор на основі правил, наскільки різноманітним буде вихідний сигнал і наскільки високою буде достовірність результатів. У розділі ми зазначали, що застосування машинного навчання може пов'язувати з високими витратами ресурсів. Ще одне питання полягає в тому, чи буде PCGML в майбутньому відбуватися на локальній платформі, чи децентралізовано в хмарі.

Щодо тем, обговорюваних у цій роботі, наступним кроком, очевидно, є об'єднання дорожньої мережі, будівництва та генерації історій в один єдиний робочий процес та усунення прогалин між ними, таких як генерація ландшафту, виявлення ділянок та генерація персонажів. Ми сподіваємося, що зможемо розвивати PCG для віртуальних світів у майбутньому та продовжувати разом зі зростаючою спільнотою PCG впроваджувати більшість інструментів PCG у виробництво.

3.4 Висновок по розділу

У третьому розділі було розглянуто реалізацію системи генерації процедурного контенту та оцінку її ефективності. Проведено реалізацію генерації процедурних об'єктів на прикладі створення будівель, що дозволило оцінити можливості використання нейромереж у задачах автоматизованого проектування. Досліджено генерацію інтерактивних історій і сценаріїв із використанням методів машинного навчання та нейромережових моделей. Також виконано оцінку якості згенерованого контенту та визначено перспективи подальшого розвитку процедурної генерації в ігровій індустрії. Отримані результати підтверджують ефективність застосування нейромереж для створення адаптивного, різноманітного та інтерактивного ігрового контенту.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено та реалізовано підходи до застосування нейромереж для генерації процедурного контенту в іграх, що є комплексним процесом, спрямованим на створення адаптивного, різноманітного та реалістичного ігрового середовища. У роботі проаналізовано існуючі методи процедурної генерації контенту, досліджено можливості використання сучасних нейромережових моделей, а також реалізовано приклади генерації ігрових об'єктів і сценаріїв. Проведене дослідження охопило як теоретичні аспекти, так і практичну реалізацію, що дозволило отримати цілісне уявлення про ефективність застосування інтелектуальних методів у геймдеві.

На початковому етапі було проведено аналіз предметної області, визначено основні підходи до процедурної генерації контенту та їх обмеження. Встановлено, що традиційні алгоритмічні методи, засновані на правилах, забезпечують обмежену варіативність і потребують значних зусиль для масштабування. У свою чергу, використання нейромереж, зокрема генеративних моделей, дозволяє створювати більш складний і різноманітний контент, який краще відповідає вимогам сучасних ігор.

У процесі моделювання було структуровано основні типи контенту, що підлягають генерації, зокрема ландшафти, архітектурні об'єкти та інтерактивні сценарії. Це дало змогу сформулювати підхід до побудови системи генерації, орієнтованої на модульність та розширюваність. Особливу увагу було приділено інтеграції нейромереж у процес генерації, що дозволило автоматизувати створення складних структур та підвищити якість результатів.

Практична реалізація включала розробку прототипу системи генерації процедурного контенту, в якій застосовано нейромережові моделі для створення будівель та генерації текстових сценаріїв. Було забезпечено взаємодію між компонентами системи, що відповідають за обробку даних, генерацію контенту та оцінку результатів. Це дозволило перевірити працездатність обраних підходів і оцінити їх ефективність у реальних умовах.

					БР.ІІІ - 72.00.00.000 ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

У ході тестування було проведено оцінку якості згенерованого контенту за такими критеріями, як різноманітність, узгодженість, реалістичність та продуктивність. Результати показали, що використання нейромереж значно підвищує якість ігрового контенту порівняно з традиційними методами, хоча водночас вимагає більших обчислювальних ресурсів та належного налаштування моделей. Виявлено також потенційні проблеми, пов'язані з контрольованістю генерації та необхідністю додаткового навчання моделей для специфічних задач.

Загалом, розроблена система відповідає поставленим цілям: вона забезпечує автоматизовану генерацію різних типів ігрового контенту, демонструє гнучкість і можливість масштабування. Перевагами запропонованого підходу є підвищення різноманітності контенту, зменшення витрат часу на розробку та можливість адаптації до поведінки користувача. Водночас існують обмеження, пов'язані з ресурсомісткістю нейромереж та складністю їх інтеграції у виробничі процеси.

У перспективі подальших досліджень доцільним є вдосконалення методів контролю генерації контенту, оптимізація продуктивності моделей, а також розширення функціональності системи за рахунок інтеграції більш складних нейромережових архітектур (наприклад, diffusion-моделей) та використання їх для генерації повноцінних ігрових світів. Це дозволить ще більше підвищити рівень автоматизації та якість створюваного контенту в сучасних відеоіграх.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. OpenAI. (2025). GPT Models Documentation. <https://platform.openai.com/docs>
2. Unity Technologies. (2025). Unity ML-Agents Toolkit. <https://unity.com/products/machine-learning-agents>
3. Unreal Engine. (2025). Procedural Content Generation Framework. <https://docs.unrealengine.com>
4. TensorFlow. (2025). Documentation. <https://www.tensorflow.org>
5. PyTorch. (2025). Documentation. <https://pytorch.org/docs>
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. <https://www.deeplearningbook.org>
7. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
8. Yannakakis, G. N., & Togelius, J. (2018). Artificial Intelligence and Games. Springer. <https://gameaibook.org>
9. Shaker, N., Togelius, J., & Nelson, M. (2016). Procedural Content Generation in Games. Springer.
10. Togelius, J., Yannakakis, G., Stanley, K., & Browne, C. (2011). Search-Based Procedural Content Generation. IEEE Transactions on Computational Intelligence and AI in Games. <https://doi.org/10.1109/TCIAIG.2011.2148116>
11. Summerville, A., et al. (2018). Procedural Content Generation via Machine Learning (PCGML). IEEE Transactions on Games. <https://doi.org/10.1109/TG.2018.2846639>
12. Radford, A., et al. (2021). Learning Transferable Visual Models From Natural Language Supervision. <https://arxiv.org/abs/2103.00020>
13. Ramesh, A., et al. (2022). Hierarchical Text-Conditional Image Generation with CLIP Latents. <https://arxiv.org/abs/2204.06125>
14. Vaswani, A., et al. (2017). Attention Is All You Need. <https://arxiv.org/abs/1706.03762>

					БР.ІІІ - 72.00.00.000 ІІЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Brown, T., et al. (2020). Language Models are Few-Shot Learners.
<https://arxiv.org/abs/2005.14165>
16. NVIDIA. (2024). GAN-based Image Generation.
<https://developer.nvidia.com>
17. Karras, T., et al. (2019). StyleGAN. <https://arxiv.org/abs/1812.04948>
18. Ronneberger, O., et al. (2015). U-Net: Convolutional Networks for Image Segmentation. <https://arxiv.org/abs/1505.04597>
19. Microsoft. (2025). ML.NET Documentation.
<https://learn.microsoft.com/en-us/dotnet/machine-learning/>
20. Hugging Face. (2025). Transformers Documentation.
<https://huggingface.co/docs>
21. OpenCV. (2025). Documentation. <https://docs.opencv.org>
22. Blender Foundation. (2025). Blender Python API.
<https://docs.blender.org/api/current/>
23. Epic Games. (2025). MetaHuman & AI Tools.
<https://www.unrealengine.com>
24. Google DeepMind. (2024). Generative Models Research.
<https://deepmind.com/research>
25. Chen, M., et al. (2021). Generative Pretrained Models for Code.
<https://arxiv.org/abs/2107.03374>
26. Karl Sims. (1991). Artificial Evolution for Computer Graphics. ACM SIGGRAPH.
27. O'Reilly. (2023). Generative Deep Learning (D. Foster).
<https://www.oreilly.com>
28. Ammanabrolu, P., et al. (2020). Story Generation with Neural Models.
<https://arxiv.org/abs/2002.02782>
29. Li, B., et al. (2021). Procedural Level Generation via GANs. IEEE.
<https://doi.org/10.1109/TG.2021.3050843>
30. Pathak, D., et al. (2017). Curiosity-driven Exploration by Self-supervised Prediction. <https://arxiv.org/abs/1705.05363>

					БР.ІІІ - 72.00.00.000 ІІЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

31. Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks.
32. IEEE. (2024). AI in Game Development Survey. <https://ieeexplore.ieee.org>
33. ACM. (2024). Advances in Procedural Content Generation. <https://dl.acm.org>
34. GameDev.net. (2025). Procedural Generation Techniques. <https://www.gamedev.net>
35. Stack Overflow. (2025). Machine Learning & Game Dev Discussions. <https://stackoverflow.com>
36. Unity Asset Store. (2025). Procedural Generation Tools. <https://assetstore.unity.com>
37. Unreal Marketplace. (2025). AI & Procedural Plugins. <https://www.unrealengine.com/marketplace>
38. ISO/IEC. (2023). Software Engineering Standards. <https://www.iso.org>
39. OWASP. (2024). Secure AI Development Practices. <https://owasp.org>
40. Gartner. (2024). AI Trends in Game Development. <https://www.gartner.com>

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Застосування нейромереж для генерації процедурного контенту в іграх

Обсяг пояснювальної записки: 83 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____

					БР.ІП - 72.00.00.000 ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		