Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук,
кандидат технічних наук, доцент
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп’ютерні науки
Освітня програма 122.00.01 Інформатика

Тема: Генерації ігрових рівнів в реальному часі з використанням нейромережі

Виконав
студент групи ІНб-2-21-4.0д
Хуторян Нікіта Романович

_____
(підпис)

Науковий керівник
к.т.н. доцент кафедри комп'ютерних наук
Рзаєва Світлана Леонідівна

_____
(підпис)

Київ – 2025

Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач
кафедри комп'ютерних наук,
кандидат технічних наук, доцент
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА
«ГЕНЕРАЦІЇ ІГРОВИХ РІВНІВ В РЕАЛЬНОМУ ЧАСІ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖІ »

Виконавець – студент групи ІНб-2-21-4.0д,
спеціальності 122 Комп'ютерні науки,
освітньої програми 122.00.01 Інформатика
Хуторян Нікіта Романович

1.Вхідні дані: тематична література на тему штучного інтелекту, власний досвід у програмуванні, інтернет-джерела про генерацію ігрових рівнів та нейромережі

2. **Основні завдання:**

- Дослідити тематичну літературу
- Розробити метод інтеграції нейромережі в ігрові рушії
- Розробити прототип нейромережі для генерації ігрових рівнів

3. **Пояснювальна записка:** Обсяг – до 70 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. **Графічні матеріали:** зображення з вихідними даними програми.

5. Строк подання роботи на кафедру: *«___» _____2025 р.*

Науковий керівник : Виконавець:

Доцент кафедри комп'ютерних наук Хуторян Н.Р _____ Дата:_____

Рзаєва С.Л. _____ Дата:_____

КАЛЕНДАРНИЙ ПЛАН КВАЛІФІКАЦІЙНОЇ РОБОТИ

	Етапи виконання	**Термін виконання**	**Відмітка**
	Формування теми	15.04.2025	Виконано
	Затвердження теми Бакалаврської роботи	11.05.2024	Виконано
	Робота над теоретичною складовою	12.05.2024	Виконано
	Написання тез	14.05.2025	Виконано
	Аналіз програмних засобів для створення проєкту	15.05.2025	Виконано
	Створення нейромережі за темою кваліфікаційної роботи	18.05.2025	Виконано
	Передзахист кваліфікаційної роботи	22.05.2025	Виконано
	Захист кваліфікаційної роботи	18.06.2025	

Здобувач: *Хуторян Н.Р*

Керівник роботи: *Рзаєва С.Л.*

РЕФЕРАТ

Кваліфікаційна робота: до 70**с., 18 рис., 31 посилання.**

Актуальність: обумовлена потребою ігрової індустрії в рішеннях, що мають більшу ефективність та змогу масштабувати створення ігрових рівнів

Об’єкт дослідження: процес створення ігрових рівнів

Предмет дослідження: методи та алгоритми процедурної генерації ігрових рівнів які містять в собі нейронні мережі
Мета роботи: дослідити методи генерації ігрових світів та розробити підхід з використанням нейронної мережі який враховує баланс між високою варіативністю контенту та його функціоналом, використовуючи при цьому переваги варіаційних автокодувальників для керованої генерації та швидкого навчання нейронної мережі на базі існуючих прикладів

Завдання:

- Розробити метод інтеграції нейромережі в ігрові рушії

- Розробити прототип нейромережі для генерації ігрових рівнів

Основні результати: у процессі дослідження було створено прототип нейромережі який здатен генерувати ігрові рівні в форматі масивів, та з можливістю інтеграції у ігрові рушії. Також був проведений аналіз існуючих методів генерації рівнів, та різних типів нейромереж.

Практичне значення дослідження: полягатиму у інтеграції нейромережі у ігрові рушії заради автоматичної генерації ігрових рівнів

Ключові слова: Генерація ігрових рівнів, нейромережі, автокодер, варіаційний автокоден

ЗМІСТ

ВСТУП

Сучасна індустрія розробки відеоігор переживає період стрімкого зростання та технологічних викликів. Вимоги гравців постійно зростають, та з часом вони приділяють більше уваги до реіграбельності, унікальності контента та величезні ігрових світів ставляючи перед розробниками нові та складні завдання. Традиційний та самий старий підхід ручного створення ігрових рівнів, хоча й забезпечує високий рівень художнього контролю, надаючи змогу дизайнерам власноруч малювати кожен елемент на карті - є надзвичайно трудомістким, витратним і не завжди такий підхід легко масштабувати для проектів з динамічно-змінюваним або нескінченним ігровим процесом. Саме через це з'явився метод процедурної генерації контенту (PCG) який пропонує ефективні рішення для оптимізації розробки та покращення ігрового досвіду.

З розвитком штучного інтелекту та, зокрема, глибинних нейронних мереж, можливості процедурної генерації рівнів значно розширилися. На відміну від класичних підходів, що використовують алгоритми та оперують заздалегідь визначеними правилами, нейронні мережі здатні навчатися складним та не завжди ординарним закономірностям за допомогою існуючих даних, а потім генерувати новий, унікальний контент, який імітує бажаний стиль та характеристики. Це відкриває шлях до створення не просто випадкових ігрових світів , що можна зробити за допомогою рандомізації, а осмислених та іноді дійсно вражаючих своєю красою ігрових світів, а саме головне - функціонуючих та справних ігрових рівнів, що мінімізує людське втручання та прискорює процес розробки.

Попри значний прогрес у цьому направленні, існують і невирішені проблеми, які вимагають подальших досліджень та пошуку нових методів реалізації. Йдеться саме про досягнення балансу між автономністю процесу генерації контенту та потребою дизайнера у контролі над вихідними даними, бо генерація рівнів за допомогою штучного інтелекту потребує бездоганної якості згенерованих рівнів без артефактів.. Вибір оптимальної архітектури та типу

нейронної мережі, що може ефективно справлятися з такими завданнями, є самим головним процесом при створенні такої моделі.

Актуальність теми дослідження обумовлена потребою ігрової індустрії в рішеннях, що мають більшу ефективність та змогу масштабувати створення ігрових рівнів, а також зростаючим потенціалом глибинного навчання у вирішенні складних задач генерації. Розробка підходів для автоматизованої генерації ігрових рівнів потребує високої якості виконання та мати можливість керувати процесом навчання нейронної мережі- це критично важлива частина для подальшого розвитку індустрії відеоігор, бо дозволяє створювати більш динамічні та реіграбельні світи.

Мета дослідження – дослідити методи генерації ігрових світів та розробити підхід з використанням нейронної мережі який враховує баланс між високою варіативністю контенту та його функціоналом, використовуючи при цьому переваги варіаційних автокодувальників для керованої генерації та швидкого навчання нейронної мережі на базі існуючих прикладів.

Об'єктом дослідження є процес створення ігрових рівнів.

Предметом дослідження є методи та алгоритми процедурної генерації ігрових рівнів які містять в собі нейронні мережі

Методи дослідження: У роботі будуть застосовані методи системного аналізу для вивчення наявних підходів до PCG та їхньої ефективності. Також будуть використовуватися методи глибинного навчання, зокрема варіаційного висновку та нейронних мереж для розробки моделі яка здатна генерувати ігрові рівні. Експериментальні методи для оцінки якості та ефективності згенерованого контенту, зокрема звичайний автокодер. Та використання програмних засобів для моделювання та симуляції, а також візуалізації результатів дослідження, буде ключовим.

РОЗДІЛ 1. ПРОЦЕДУРНА ГЕНЕРАЦІЯ ІГРОВИХ РІВНІВ: ТЕОРЕТИЧНІ АСПЕКТИ ТА ЗАСТОСУВАННЯ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Через швидкий розвиток індустрії відеоігор у 21му столітті - очікування гравців щодо нових продуктів виросли пропорційно. Їхні вимоги до різноманітності та якості ігрового контенту поставили перед розробниками нові та складні виклики. Ручне створення ігрових рівнів, що завжди було трудомістким та довгим процесом, дедалі більше стає непрактичним, особливо для проєктів з великим відкритим світом чи ігор, що пропонують нескінченну реіграбельність. Саме тут з’являється концепція процедурної генерації контенту (PCG), яка передбачає використання чітко прописаних алгоритмів для автоматичного створення ігрових елементів. Це не просто технічний інструмент, а по суті, нова реальність в ігровому дизайні, що дає змогу масштабувати виробництво контенту, роблячи його унікальним при кожному новому запуску гри та навіть адаптує ігровий світ під індивідуальні уподобання гравця.

Попри очевидні переваги, PCG - не є вирішенням усіх проблем , а навіть додає нові, які пов'язані із забезпеченням якості та цілісності згенерованого контенту. Питання “Чи може алгоритм створити рівень, який буде не тільки функціональним, а й захопливим?” довгий час залишалося відкритим. Традиційні підходи до PCG, засновані на жорстких правилах та обмеження алгоритмів, часто стикалися з проблемою генерації одноманітного або, навпаки, хаотичного контенту, що не відповідав критеріям ігрового дизайну.

З появою та стрімким розвитком методів машинного навчання, зокрема глибинних нейронних мереж, у сфері процедурної генерації з’явилося ще більше перспективне майбутнє. Ці потужні інструменти відкрили шлях до створення складніших, адаптивніших та, що найважливіше, "навчених" на реальних прикладах систем генерації. Нейронні мережі здатні не просто

генерувати результат відповідно до правил, а й вивчати приховані закономірності та структури з існуючих даних, а потім генерувати новий контент, який імітує ці властивості. Замість того, щоб жорстко програмувати кожне правило створення рівня, ми можемо "навчити" систему розуміти, що робить рівень "хорошим" або "цікавим" з точки зору дизайну, і дозволити їй самостійно створювати варіації не обмежуючи ніякими правилами.

1.1 Еволюція процедурної генерації контенту: від детермінованих алгоритмів до адаптивних систем

Процедурна генерація контенту – це не нове явище в розробці відеоігор. Її витоки сягають ще епохи перших комп'ютерних ігор, коли суворі обмеження обчислювальних ресурсів змушували розробників шукати нетипові рішення для створення ігрового світу. Спочатку PCG була більш детермінованою, що має на увазі те, що алгоритми були жорстко запрограмовані, та мали надто обмежені правила. Яскравим прикладом є ігри жанру roguelike, де кожен ігровий рівень був унікальним, але створювався за допомогою простих алгоритмів які комбінували попередньо визначені "блоки" або готові "кімнати" за певними правилами з'єднання. Цей підхід був ефективним для створення великих, але часто структурно схожих світів. Приблизно це можна порівняти з операцією додавання в математиці, наприклад: 2+4=6 та 4+2=6 - візуально виглядає по-іншому, але результат один й тей самий.

З розвитком комп’ютерних систем та ускладненням ігор, детерміновані методи почали демонструвати свої обмеження та стали застарілими. Вони часто створювали контент по шаблонам, який, попри свою унікальність, швидко втрачав новизну, або ж, навпаки, генерували незбалансовані, нецікаві або навіть непрохідні рівні, оскільки алгоритмам бракувало "розуміння" ігрового дизайну. Це спонукало до пошуку гнучкіших та інноваційніших підходів до PCG.

Наступним етапом стало впровадження стохастичних елементів та граматичних систем. Додавання випадковості дозволило збільшити варіативність, а граматичні системи (наприклад, контекстно-вільні граматики) надавали можливість описувати складніші структури та їхні взаємозв'язки. Але навіть це не вирішило основну проблему - вона залишалася той самой: як змусити алгоритм генерувати якісний контент, який відповідав би високим стандартам ігрового дизайну? Це вимагало переходу від суто генеративних підходів до тих, що включають елементи самооцінки та оптимізації.

Так з'явилися еволюційні алгоритми, які використовують принципи природного відбору для "еволюції" параметрів генерації або навіть самих алгоритмів. Популяція потенційних рішень (наприклад, наборів параметрів для рівня) оцінюється за певними критеріями (функція придатності), а потім найкращі "індивідууми" відбираються для "схрещування" та "мутації", створюючи нові покоління. Цей процес дозволяє знаходити оптимальні рішення, що відповідають бажаним характеристикам, таким як складність, баланс чи естетика. Приклади їх застосування включають генерацію ландшафтів, рівнів та навіть правил гри.

Зрештою, на порозі сучасних технологій де обчислювальні потужністю доступні кожному, як і доступ до великих обсягів даних, відбувся прорив у застосуванні машинного навчання та, зокрема, глибинних нейронних мере. Це дозволило перейти від явно запрограмованих правил до систем, які можуть навчатися цим правилам та навіть створювати свої, або прихованим закономірностям безпосередньо з існуючих даних. Цей парадигмальний зсув відкрив можливості для створення значно складніших та адаптивніших генеративних систем, здатних імітувати стиль ігрових дизайнерів і навіть створювати по-справжньому нові та несподівані, але при цьому логічні та збалансовані ігрові світи.

1.2 Нейронні мережі як інструмент процедурної генерації контенту: архітектури та принципи функціонування

Здатність нейронних мереж навчатися складним, нелінійним закономірностям із великих обсягів даних зробила їх надзвичайно привабливим інструментом для задач процедурної генерації контенту. На відміну від алгоритмів, що оперують заздалегідь визначеними правилами, нейронні мережі можуть виявляти приховані структури в існуючому контенті (наприклад, у наборі вже спроєктованих ігрових рівнів) і згодом генерувати нові, статистично схожі, але унікальні зразки. Цей підхід “відчиняє” двері для створення контенту, який відповідає певному стилю, але при цьому є унікальним.

Серед різних архітектур нейронних мереж, що використовуються в різних задачах, окреме місце посідають ті, що здатні навчатися складним розподілам даних та синтезувати нові зразки. До них належать, зокрема, генеративно-змагальні мережі (GANs) та, що особливо актуально для цього дослідження - варіаційні автокодувальники (VAEs).

1.2.1 Генеративно-змагальні мережі (GANs) у контексті PCG

Генеративно-змагальні мережі, запропоновані Гудфеллоу у 2014 році , є однією з найпотужніших архітектур для генерації даних які схожі на вхідні. Їхня структура складається з двох основних компонентів: генератора (Generator) та дискримінатора (Discriminator), що працюють у змагальному режимі. Генератор намагається створювати зразки, які якомога більше нагадують реальні дані, тоді як дискримінатор навчається відрізняти справжні дані від згенерованих. Цей "змагальний" процес дозволяє обом мережам взаємно вдосконалюватися: генератор стає все кращим у "підробці" даних, а дискримінатор – у їхньому виявленні.

У контексті процедурної генерації ігрових рівнів, генератор може навчатися створювати зображення рівнів, послідовності об'єктів або навіть складні тривимірні сцени. Дискримінатор, у свою чергу, оцінює, наскільки згенерований рівень відповідає естетичним або функціональним критеріям,

закладеним у тренувальних даних. З перевага GAN можна відмітити здатність генерувати високоякісні, візуально переконливі зразки. Однак, вони мають і певні недоліки:

A) Режимний колапс: Генератор може навчитися генерувати лише обмежений набір зразків, ігноруючи решту різноманіття тренувальних даних. Це призводить до одноманітності згенерованого контенту.
B) Складність навчання: Навчання GANs є досить нестабільним процесом, що вимагає ретельного налаштування гіперпараметрів та архітектури, та також повинні бути структуровані вхідні дані.
C) Відсутність прямого контролю над латентним простором: Хоча існує умовний GAN (Conditional GAN), що дозволяє контролювати деякі атрибути згенерованих даних, прямий контроль над властивостями виходу все одно може бути обмеженний.
D) Незважаючи на ці виклики, GAN знайшов застосування у генерації текстур , персонажів та, звісно, двовимірних ігрових рівнів, демонструючи гарні результати у створенні різних рівнів, але зі схожою ідеєю.

1.2.2 Варіаційні автокодувальники (VAEs) як генеративні моделі

На противагу змагальній парадигмі GANs, варіаційні автокодувальники (VAEs), вперше представлені Кінгмою та Веллінг у 2013 році, є генеративними моделями, що базуються на принципах автокодування та варіаційного висновку. Основна ідея VAE полягає в навчанні ймовірного розподілу даних у латентному просторі, що дозволяє потім відбирати з цього розподілу нові точки та декодувати їх у нові, унікальні зразки даних.

Архітектура VAE складається з двох основних частин:

A)Кодувальник (Encoder) - це частина нейромережі яка приймає вхідні дані і стискає їх у латентне представлення, в особливому вигляді. Проте, на відміну від класичного автокодувальника, кодувальник VAE виводить не

єдину точку в латентному просторі, а параметри (середнє значення μ та логарифм дисперсії logσ2) ймовірного розподілу для цієї точки.

Б) Декодувальник (Decoder): Ця мережа приймає випадкову вибірку з латентного простору (з розподілу, визначеного кодувальником) і перетворює її назад у простір оригінальних даних, намагаючись відновити вхідні дані.

Навчання VAE стаєтьсяза допомогою функції втрат, яка складається з двох основних компонентів:

А) Втрати реконструкції (Reconstruction Loss): Вимірює, наскільки добре декодувальник відновив вхідні дані. Це міра, яка відображає схожість створених даних та оригінала.

В) Втрати Кульбака-Лейблера (KL Divergence Loss): Ця компонента штрафує модель за відхилення латентного розподілу від стандартного нормального розподілу. Вона заохочує латентний простір бути схожим на вхідні дані, що важливо для генерації нових, осмислених зразків шляхом інтерполяції або випадкової вибірки. Саме завдяки цій компоненті VAE може генерувати нові дані, а не просто їх повторювати.

Ключова перевага використання VAE для PCG полягає в його здатності створювати комбіновані варіанти рівнів, які не були присутні в тренувальних даних. Завдяки тому, що латентному простор має систему штрафів, ми можемо переміщатися між точками, що відповідають різним рівням, і генерувати плавну послідовність проміжних станів. Це дає дизайнерам змогу контролювату процедуру генерації набагато більше, та направлятя її у коректну сторону.

На відміну від GAN, VAEs зазвичай не генерують настільки "фотореалістичні" зображення, але їхня стабільність навчанняроблять їх ідеальним вибором для задач, де важлива не лише генерація, а й можливість контролювати та модифікувати властивості згенерованого контенту для того щоб отримувати нові результати. Для задач генерації ігрових рівнів, де

варіативність та контрольованість часто є пріоритетнішими за абсолютний фотореалізм, VAEs виявляються дуже ефективним інструментом.

1.2.3 Варіаційні автокодувальники у світлі генерації ігрових рівнів: особливості та нюанси втілення

Уявимо ігровий рівень не просто як сукупність елементів, а як візуальний відбиток або ж матрицю, де кожна клітинка (піксель) кодує конкретний тип об'єкта: стіну, підлогу, ворога або ж цінного артефакту. Саме такі "зображення рівнів" VAE здатен спочатку стиснути до вкрай компактного, але інформативного латентного простору, а потім – розшифрувати з нього абсолютно нові, унікальніі варіації.

Ключові аспекти використання VAE для генерації рівнів розкриваються через кілька вимірів:

A) Формалізація даних: Для ефективного навчання VAE ігрові рівні мусять бути трансформовані у числовий формат. Це може бути проста бінарна матриця або розширена матриця з категоріальними значеннями для різнотипних об'єктів або навіть кольорове зображення, де кожен ігровий елемент має свій унікальний колірний код. Вибір способу представлення критично впливає на внутрішню будову кодувальника та декодувальника – наприклад, для обробки зображень найкраще пасують згорткові шари.

B) Процес навчання: VAE "набирається досвіду", аналізуючи великий масив вже існуючих ігрових рівнів. Ці рівні стають для моделі зразками, з яких вона видобуває приховані закономірності дизайну. Основна мета тренування – забезпечити "гладкість" і репрезентативність латентного простору. Це означає, що схожі за структурою рівні повинні розташовуватися в латентному просторі поруч, а перехід між ними має бути настільки плавним, щоб дозволяти генерувати логічні, осмислені

проміжні варіанти.

C) Створення нового контенту: Після завершення навчання, генерація свіжого рівня відбувається доволі елегантно: ми просто випадковим чином обираємо точку в латентному просторі (як правило, зі стандартного нормального розподілу, до якого VAE прагне звести свій простір) і дозволяємо декодувальнику перетворити її на повноцінний ігровий рівень. Завдяки згаданій гладкості латентного простору, отримуємо безліч унікальних, але стилістично та структурно цілісних рівнів.

D) Керованість генерацією: Можливість контрольованої генерації є однією з найцінніших переваг VAE. Шляхом маніпуляцій з латентним вектором – змінюючи його окремі компоненти або плавно переходячи між векторами різних зразків рівнів – ми можемо безпосередньо впливати на характеристики кінцевого результату. Уявити, що певні виміри латентного простору корелюють зі "складністю" чи "відкритістю" рівня, дозволяє дизайнеру програмно коригувати ці параметри для досягнення бажаного ефекту. Це надає розробникам потужний інструмент для гнучкого налаштування генеративного процесу.

Практична реалізація VAE для ігрових рівнів нерідко потребує адаптації стандартних архітектур. Зокрема, для ефективної роботи з просторовими даними кодувальник і декодувальник часто будуються на базі згорткових нейронних мереж (Convolutional Neural Networks, CNNs), які вже чудово довели свою ефективність в задачах обробки зображень. Згорткові шари вміло витягують локальні просторові ознаки, що є критично важливим для глибокого розуміння структури ігрових світів.

Звісно, попри всі переваги, існують і певні виклики. Згенеровані VAE рівні іноді можуть містити небажані артефакти або логічні суперечності, особливо

коли модель навчається на обмежених або не ідеально підготовлених даних. Крім того, підбір оптимальних гіперпараметрів та архітектури VAE для конкретного завдання генерації рівнів залишається непростим, але захопливим процесом, що вимагає ретельних експериментів та глибокого розуміння як механізмів машинного навчання, так і специфіки ігрового дизайну.

1.3 Як оцінити якість автоматично створеного ігрового контенту?

Коли система процедурної генерації створює новий рівень чи локацію, одразу постає питання — а чи добре вона це зробила? Звучить просто, але насправді це доволі складна задача. Якщо звичайні рівні перевіряють дизайнери , то тут потрібні чіткі критерії оцінки, які можна автоматизувати.

Проблема в тому, що "хороший" ігровий рівень — це не просто те, що технічно працює. Він має бути цікавим, не занадто легким і не неможливо складним, плюс ще й приємним для ока. Тобто доводиться поєднувати числові показники з більш суб'єктивними речами на кшталт "а чи сподобається це гравцеві?".

1.3.1 Оцінювання за допомогою числових показників

Використовувати числові показники є найефективнішим методом оцінювання. Через те, що за допомогою чисел нейромережа може сама оцінювати себе, без втручання людини

A) Чи можна пройти рівень взагалі? Базова, але критично важлива річ. Якщо від точки старту неможливо дістатися до фінішу — рівень бракований. Тут допомагають алгоритми пошуку шляху.
B) Наскільки він складний? Можна підрахувати кількість ворогів, перешкод, необхідних кроків для проходження. Деякі дослідники навіть запускають ботів, які проходять рівень, і дивляться, скільки спроб їм потрібно.
C) Чи достатньо різноманітні згенеровані рівні? Якщо система видає один і той же рівень з мінімальними змінами — це погано для реіграбельності.

Тут використовують різні метрики подібності (наприклад, відстань Хеммінга для бінарних карт).

D) Збалансованість елементів. Чи не занадто багато ворогів у одній частині рівня? Чи достатньо ресурсів для гравця? Це комбінація декількох показників разом.

E) Відповідність технічному завданню. Якщо ми хочемо рівень із 5 кімнатами і обов'язково з боссом — чи дотримується система цих вимог?

1.3.2 Суб’єктивне оцінювання за допомогою людської думки

Але числові показники не є єдиним і завжди підходящим методом оцінювання. Є речі, які може оцінити лише людина:

A) Експертна оцінка. Досвідчені дизайнери чи геймери дивляться на згенеровані рівні і ставлять бали. Найпряміший метод, але відверто кажучи, дорогий і не завжди об'єктивний.

B) Тестування з реальними гравцями. Даємо людям пограти і дивимося — скільки часу витрачають, де застрягають, чи подобається їм процес. Дуже інформативно, але знову ж таки, потребує багато часу.

C) Опитування та анкети. Після гри просимо гравців розповісти про враження. Що сподобалось, що ні, чи хотіли б зіграти ще раз.

D) Естетична привабливість. Показуємо рівні групі людей і просимо оцінити, наскільки вони виглядають "круто" чи "гармонійно".

1.4 Як вписати нейронні мережі в реальну розробку ігор

Генерувати красиві картинки рівнів за допомогою VAE — це ефективно. Але справжня цінність таких систем з'являється тоді, коли їх можна реально використати в процесі створення гри. І тут починаються цікаві виклики:

згенерований контент має не просто виглядати гарно, а ще й працювати як справжній ігровий рівень.

Щоб це зробити, потрібно розуміти не тільки машинне навчання, але й те, як взагалі створюються ігри і що потрібно дизайнерам.

1.4.1 Гібридні підходи та взаємодія з традиційними методами PCG

Важливо розуміти, що нейронні мережі не обов'язково мають повністю замінювати традиційні методи генерації. Часто більш ефективним виявляється їх поєднання. Наприклад, VAE може генерувати базову структуру рівня, після чого традиційні алгоритми забезпечують її функціональність — перевіряють прохідність від старту до фінішу, розміщують ігрові елементи відповідно до встановлених правил.

Такий підхід дозволяє поєднати креативні можливості нейронної мережі з надійністю та контрольованістю детерміністичних алгоритмів.

Основні гібридні моделі:

-VAE з постобробкою на основі правил. Нейронна мережа генерує базову карту рівня, після чого традиційні алгоритми здійснюють її коригування — перевіряють зв'язність, розміщують об'єкти згідно з логікою гри. Це забезпечує поєднання стилістичної варіативності та функціональної коректності.

-Умовні VAE з експертними параметрами. Можна навчити мережу генерувати рівні відповідно до заданих характеристик: рівень складності, жанрові особливості, акцент на певних ігрових механіках. Дизайнер визначає параметри, мережа їх реалізує в конкретному рівні.

-Генерація модульних компонентів. Замість створення цілісного рівня VAE може генерувати окремі структурні елементи або "кімнати", які потім

об'єднуються традиційними алгоритмами компонування. Особливо ефективно це працює для тайлових ігор.

Ключовим принципом залишається не заміщення людського дизайнера, а надання йому потужного інструменту для оптимізації творчого процесу.

1.4.2 Підготовка даних для навчання генеративних моделей

Якість та обсяг тренувальних даних визначають ефективність будь-якої системи глибинного навчання, і генерація ігрових рівнів не є винятком. Навіть досконала архітектура мережі не зможе функціонувати належним чином при використанні неякісних або недостатніх даних.

Ключові аспекти роботи з даними:

A) Джерела тренувальних даних. Для навчання можуть використовуватися рівні з існуючих ігор (отримані через парсинг), рівні, створені дизайнерами вручну, або навіть вихідні дані простіших алгоритмів генерації як базова основа.
B) Формат представлення даних. Рівні потребують адаптації до формату, який може обробляти нейронна мережа. Це можуть бути растрові зображення (де кольори кодують типи елементів), числові матриці, або векторні представлення з описом об'єктів та їх властивостей. Вибір формату впливає на архітектуру моделі.
C) Нормалізація даних. Часто необхідна нормалізація значень (приведення до діапазону 0-1) для стабілізації процесу навчання мережі.
D) Методи збагачення даних. При обмеженому обсязі доступних даних можна використовувати техніки аугментації — горизонтальне/вертикальне відображення, повороти (якщо це не порушує логіку рівня), додавання контрольованого шуму. Важливо дотримуватися балансу.

E) Контроль якості даних. Принцип "garbage in, garbage out" є фундаментальним у машинному навчанні. Наявність некоректних, непрохідних або логічно суперечливих рівнів у тренувальному наборі призведе до відтворення цих недоліків моделлю. Тому ретельна фільтрація та валідація даних є критично важливою.

1.4.3 Виклики практичної інтеграції та перспективи розвитку

Створення прототипу, який генерує візуальні представлення рівнів, суттєво відрізняється від розробки системи, інтегрованої в реальний ігровий рушій і здатної функціонувати під час розробки або безпосередньо в процесі гри.

Основні проблеми:

A) Формат виходу. Мережа видає зображення, а рушію потрібні структуровані дані — масиви тайлів, списки об'єктів з координатами, можливо навіть 3D-моделі. Треба писати парсери, які це все перетворюють.
B) Швидкість. Якщо рівні генеруються під час гри, мережа має працювати швидко. Доводиться оптимізувати модель, використовувати GPU-прискорення.
C) Гнучкість. Дизайнери хочуть мати контроль — задавати обмеження, редагувати згенерований контент, вбудовувати свої ідеї. Система має це підтримувати.
D) Валідація в реальному часі. Потрібні автоматичні перевірки — чи рівень прохідний, чи немає заблокованих зон, чи все логічно. І це має працювати швидко, без участі людини.

Результат того вартий. Якщо все зробити правильно, можливо:

- Прискорити прототипування — дизайнери швидко тестують ідеї

- Збільшити кількість контенту — особливо важливо для ігор, де треба багато унікальних рівнів
- Адаптувати під гравця — генерувати рівні залежно від стилю гри та навичок
- Зробити гру "живою" — вона може вчитися на поведінці гравців і ставати кращою

Інтеграція нейронних мереж у PCG — це не просто технічна задача. Це новий підхід до розробки ігор, який може все змінити. Але для успіху потрібна команда, де є і ML-інженери, і геймдизайнери, і звичайні програмісти.

Висновки розділу 1

Були розібрати теоретичні основи процедурної генерації в іграх — від простих алгоритмів до сучасних нейронних мереж. Стало зрозуміло, що традиційні методи (алгоритмічні, граматичні, еволюційні) дають хороший контроль, але часто застрягають на обмеженій варіативності.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ІГРОВИХ РІВНІВ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Незважаючи на суттєвий прогрес у розробці методів процедурної генерації контенту (детально розглянутий у попередньому розділі), індустрія відеоігор продовжує стикатися з рядом нерозв'язаних проблем. Особливо гостро стоїть питання збалансування повного контролю дизайнера над ігровим досвідом із ефективністю автоматизованих систем. Традиційні підходи до PCG, забезпечуючи високу швидкість генерації, нерідко створюють контент, який сприймається як механічний — позбавлений творчої складової чи такий, що просто не відповідає цільовому ігровому стилю. У даному розділі здійснено критичний аналіз наявних рішень, що використовують нейронні мережі для генерації ігрових рівнів, виявлено їхні обмеження та сформульовано дослідницьке завдання, вирішення якого є метою цієї роботи.

2.1 Критичний огляд сучасних підходів до генерації ігрових рівнів за допомогою нейронних мереж

За останні кілька років можна спостерігати справжню експансію методів глибинного навчання у сферу процедурної генерації ігрового контенту. Причина такого інтересу досить очевидна – нейронні мережі здатні виявляти неочевидні закономірності у високовимірних даних, що відкриває перспективи автоматизованого створення складного та привабливого контенту.

GAN-архітектури:

Генеративно-змагальні мережі продемонстрували, мабуть, найбільш вражаючі результати у плані візуальної якості. Умовні архітектури на кшталт Pix2Pix виявилися особливо ефективними для трансформації схематичних зображень рівнів у деталізовані візуальні карти. CycleGAN, своєю чергою,

дозволяє виконувати стилістичні перетворення без потреби у парних навчальних даних – досить корисна властивість на практиці.

Експерименти з безумовними GAN показують здатність до генерації двовимірних рівневих структур з вражаючою візуальною достовірністю. Проте детальніший аналіз виявляє серйозні підводні камені. Нестабільність навчального процесу та злощасна схильність до mode collapse призводять до того, що різноманітність генерованих зразків істотно деградує. Математично це проявляється як зниження ентропії розподілу результатів порівняно з цільовим.

RNN-підходи:

Рекурентні архітектури, зокрема LSTM та GRU варіанти, знайшли своє місце у задачах послідовної генерації рівневих елементів. Їхня здатність кодувати часові залежності дозволяє дотримуватися локальних правил розміщення об'єктів та врахування фізичних обмежень – що дуже важливо для створення "граблених" рівнів.

На практиці RNN показують непогані результати при генерації лінійних послідовностей для платформерів, де контекст обмежується кількома попередніми елементами. Але ось коли справа доходить до нелінійних, двовимірних структур – тут починаються проблеми. Класична проблема vanishing gradient у довгих послідовностях призводить до втрати довгострокових залежностей, внаслідок чого згенеровані структури виходять фрагментованими.

VAE-методології: обіцянки латентного простору та їх реальність

Варіаційні автокодувальники пропонують концептуально інший підхід через створення структурованого латентного простору з визначеними статистичними властивостями. Теоретично це повинно забезпечувати не просто генерацію нових зразків, а й можливість контрольованої інтерполяції між існуючими структурами – досить привабливе обіцяння.

Практичні дослідження підтверджують спроможність VAE успішно моделювати розподіли рівневих структур класичних 2D-ігор. Регуляризація через KL-дивергенцію забезпечує гладкість латентного простору, що начебто має спрощувати інтерпретацію закодованих характеристик.

Однак реальність виявляється складнішою. Reconstruction loss у стандартних VAE часто призводить до розмитості результатів – візуально це помітно поступається GAN-архітектурам. Крім того, припущення про гауссівський характер апостеріорного розподілу може бути занадто обмежувальним для складних рівневих структур. А той самий disentanglement латентних представлень, попри всі теоретичні обіцянки, на практиці виявляється нетривіальною задачею, яка потребує спеціалізованих архітектурних хитрощів.

Висновок:

Порівнюючи різні підходи, стає очевидним, що кожна архітектура має власну "зону комфорту" та специфічні слабкості. GANs дають найкращу візуальну якість, але страждають від нестабільності та слабкого контролю. RNN ефективні для послідовних структур, проте мають фундаментальні обмеження при роботі зі складними топологіями. VAE обіцяють найкращі можливості контролю, але поступаються у якості реконструкції. Такий стан речей вказує на відсутність "срібної кулі" та необхідність або гібридних підходів, або спеціалізованих архітектур під конкретні типи ігрових завдань.

2.2 Існуючі обмеження та невирішені проблеми в автоматизованій генерації рівнів

Проведений огляд підкреслює, що, хоча нейронні мережі відкрили нові горизонти для PCG, ці рішення не є універсальною "чарівною паличкою".

Існують фундаментальні обмеження та невирішені проблеми, які вимагають подальших досліджень та розробок:

A) **Контрольованість та інтерпретованість:** Одна з головних проблем полягає у недостатньому контролі над процесом генерації. Розробники часто прагнуть не просто "створити щось випадкове", а генерувати рівні, що відповідають певним, заздалегідь визначеним критеріям (наприклад, рівень складності, наявність певних елементів, дотримання стилістики). Хоча умовні генеративні моделі (Conditional GANs, Conditional VAEs) пропонують часткове рішення, точний та інтуїтивно зрозумілий контроль над складними атрибутами рівня все ще є викликом. Інтерпретованість латентного простору, особливо для великих та складних мереж, залишається складною задачею, що обмежує здатність дизайнера розуміти, чому мережа генерує саме такий контент.

B) **Якість та різноманітність:** Досягнення оптимального балансу між якістю та різноманітністю є постійною дилемою. Деякі моделі можуть генерувати дуже якісні, але одноманітні рівні, тоді як інші продукують високу різноманітність, але часто з артефактами або нелогічними елементами. Проблема "режимного колапсу" в GANs є яскравим тому підтвердженням.

C) **Необхідність великих обсягів тренувальних даних:** Більшість сучасних моделей глибинного навчання вимагають величезних обсягів розмічених даних для ефективного навчання. У випадку з ігровими рівнями це означає, що для навчання моделі потрібні сотні або тисячі вже спроєктованих рівнів, що може бути значною перешкодою для невеликих студій або інді-розробників.

D) **Інтеграція в ігрові рушії та ігровий цикл:** Створення автономної системи генерації – це лише половина справи. Реальна цінність з'являється тоді, коли цю систему можна безшовно інтегрувати в ігровий рушій і використовувати її для генерації контенту "на льоту" або як

частину робочого процесу дизайнера. Це вимагає не лише генерації візуальних даних, а й структурної інформації, яку може обробити ігровий рушій (наприклад, колізійні мапи, дані про розташування об'єктів, інформацію про прохідність).

Е) **Оцінка та валідація:** Навіть якщо рівень згенерований, як ми перевіримо, що він є "хорошим"? Попередній розділ коротко торкався метрик, але розробка надійних, універсальних метрик, що враховують не лише функціональність, а й ігровий досвід, залишається активною галуззю досліджень.

Ці проблеми свідчать про те, що, незважаючи на обіцянки, сучасні методи PCG на основі нейронних мереж все ще мають значний простір для вдосконалення. Наше дослідження буде зосереджено на подоланні деяких з цих обмежень, зокрема щодо контрольованості та інтеграції згенерованого контенту.

- Деталізації викликів контрольованої генерації в VAEs: Як саме досягається контроль і чому це складно.
- Нюансах представлення ігрових рівнів для VAEs: Різні підходи до кодування інформації про рівень.
- Проблемах оцінки та валідації генерованого контенту: Які методи використовуються і чому вони не завжди достатні.
- Масштабованість та ефективність: Як забезпечити, щоб система працювала не лише в теорії, а й на практиці.

2.3 Детальний аналіз методологічних прогалин та невирішених аспектів у існуючих підходах до генерації рівнів на основі VAE

Незважаючи на значні успіхи, досягнуті завдяки застосуванню варіаційних автокодувальників (VAEs) для процедурної генерації ігрових рівнів, існує низка методологічних прогалин та невирішених аспектів, що стримують їхнє широке впровадження у комерційній розробці. Зокрема, питання керованої генерації, ефективного представлення складних ігрових даних та надійних метрик оцінки залишаються предметом активних наукових дискусій та досліджень.

2.3.1 Виклики контрольованої генерації та інтерпретації латентного простору VAE

Одна з фундаментальних переваг VAE полягає у здатності формувати **гладкий та безперервний латентний простір**, що теоретично дозволяє здійснювати інтерполяцію між існуючими зразками та генерувати нові, логічні варіації. Однак, на практиці, цей латентний простір не завжди є настільки "інтерпретованим", як хотілося б. Тобто, не завжди зрозуміло, які конкретні "виміри" в цьому прихованому просторі відповідають за певні характеристики згенерованого рівня, такі як складність, щільність об'єктів, наявність пасток чи тип ландшафту.

Традиційно, для досягнення певної міри контролю над генерацією VAE, дослідники вдаються до кількох стратегій:

Навчання умовних VAE (Conditional VAEs, CVAE): Цей підхід передбачає подачу додаткової інформації (умови) на вхід як кодувальнику, так і декодувальнику. Цією умовою може бути вектор, що представляє бажані атрибути рівня (наприклад, "складність: висока", "тип локації: ліс"). Хоча CVAE дозволяють генерувати рівні, що відповідають певним умовам, визначення оптимальних умов і забезпечення їхнього точного впливу на вихід все ще є складним завданням. Більше того, не завжди легко точно кількісно описати всі бажані характеристики рівня для подачі на вхід моделі .

Пошук "дизентангованих" латентних просторів: Ідеальний латентний простір – це той, де кожен вимір (або невеликий набір вимірів) відповідає за один, незалежний, семантично зрозумілий атрибут даних. Тобто, зміна одного виміру латентного вектора повинна призводити до зміни лише однієї характеристики рівня, не зачіпаючи інших. Для досягнення цього використовуються модифікації функції втрат VAE (наприклад, β-VAE або FactorVAE), що заохочують модель до "дизентангованого" навчання. Проте, повна дизентангованність є надзвичайно складною, якщо не неможливою, для складних даних, таких як ігрові рівні, де багато атрибутів взаємопов'язані.

Пост-генераційні маніпуляції та оптимізація: Часто після генерації рівня VAE, для досягнення бажаних властивостей застосовуються додаткові алгоритми оптимізації або евристики. Це може включати коригування окремих елементів, додавання або видалення об'єктів, або навіть перевірку та модифікацію прохідності. Хоча це дозволяє "довести" рівень до потрібного стану, це частково нівелює переваги автоматизованої генерації і вимагає додаткових обчислювальних ресурсів.

Проблема інтерпретації латентного простору VAE є критичною, оскільки вона безпосередньо впливає на керованість системи. Якщо дизайнер не може інтуїтивно зрозуміти, як зміна латентного вектора вплине на згенерований рівень, то інструмент втрачає свою цінність як креативний помічник, перетворюючись на "чорний ящик". Це вимагає розробки візуалізаційних інструментів та методів для кращого розуміння та маніпуляції латентним простором.

2.3.2 Представлення ігрових рівнів для нейронних мереж: вибір оптимальної репрезентації

Спосіб, у який ігровий рівень представлений для нейронної мережі, є фундаментальним для успіху генерації. Існує кілька підходів, кожен з яких має

свої переваги та недоліки, і вибір оптимального залежить від типу гри, складності рівня та бажаних результатів:

Плиткові (Tile-based) представлення: Це один з найпоширеніших методів, особливо для 2D-ігор. Кожен рівень кодується як двовимірна матриця, де кожній комірці (плитці) присвоюється числовий ідентифікатор, що відповідає певному типу об'єкта (наприклад, 0 – порожньо, 1 – стіна, 2 – підлога, 3 – ворог). Цей формат легко обробляється згортковими нейронними мережами (CNN), які чудово справляються з просторовими даними. Однак, такий підхід може бути менш ефективним для представлення об'єктів, що займають кілька плиток, або об'єктів зі складними взаємодіями.

Зображення (Image-based) представлення: Рівні можуть бути представлені як звичайні зображення, де різні типи плиток або об'єктів кодуються різними кольорами (наприклад, стіни – чорні, підлога – біла, вороги – червоні). Це дозволяє використовувати потужні архітектури CNN, розроблені для обробки зображень. Проте, така репрезентація може втрачати деяку семантичну інформацію, і для отримання функціональних даних для рушія потрібен додатковий етап постобробки, який перетворює пікселі назад у структуровані ігрові об'єкти.

Графові представлення: Для складних рівнів з нелінійною структурою або великою кількістю взаємозв'язків між об'єктами, рівень може бути представлений як граф. Вузли графа можуть бути кімнатами, зонами або ключовими об'єктами, а ребра – зв'язками між ними. Генерація таких структур вимагає використання Графових нейронних мереж (Graph Neural Networks, GNNs), що є більш складною, але потенційно потужнішою архітектурою для складних, нерегулярних даних. Проте, розробка GNN для генерації є порівняно новою областю і вимагає значно більше обчислювальних ресурсів.

Гібридні представлення: Часто оптимальним є поєднання різних підходів. Наприклад, основна структура рівня може бути згенерована як плиткове

представлення, а потім деталізовані об'єкти або їхні атрибути можуть бути додані за допомогою іншого методу або навіть ручної корекції.

2.3.3 Якість згенерованого контенту та проблеми валідації

Проблема оцінки якості згенерованих нейронними мережами ігрових рівнів є однією з найскладніших. Як ми вже згадували у Розділі 1.3, це включає як об'єктивні, так і суб'єктивні метрики. Проте, глибинний аналіз виявляє, що навіть ці метрики мають свої недоліки:

1) **Об'єктивні метрики:** Хоча прохідність, збалансованість та різноманітність можуть бути виміряні кількісно, вони не завжди корелюють з "веселощами" або "цікавістю" рівня. Рівень може бути прохідним, але нудним; збалансованим, але неінноваційним. Розробка метрик, які могли б точно вимірювати такі суб'єктивні характеристики, залишається відкритим питанням. Більше того, більшість об'єктивних метрик вимагають попередньо визначених правил або симуляцій, що може бути складним для динамічно генерованого контенту.
2) **Суб'єктивні метрики:** Людська оцінка, хоч і є "золотим стандартом" для визначення "хорошого" рівня, є трудомісткою, дорогою та схильною до суб'єктивності та упередженості. Забезпечення статистично значущих результатів вимагає великої кількості тестерів та ретельно розроблених опитувальників. Це робить її непридатною для швидкої, автоматизованої валідації великих обсягів згенерованого контенту.
3) **"Artifacts" та "Glitchy" контент:** Однією з постійних проблем генеративних моделей є поява "артефактів" – нелогічних або нефункціональних елементів на рівні, які виникають через недоліки навчання моделі або через те, що модель бачила недостатньо різноманітних прикладів. Це можуть бути стіни, що не з'єднуються, заблоковані проходи, об'єкти, розміщені в повітрі, або візуальні спотворення. Ефективне автоматичне виявлення та виправлення цих

артефактів – це окреме дослідницьке завдання, що часто вимагає поєднання машинного навчання з традиційними евристиками.

Таким чином, для забезпечення високої якості генерованих рівнів, необхідно не лише розробити потужну генеративну модель, але й впровадити надійні механізми валідації, можливо, у формі ієрархічної системи, де базові функціональні вимоги перевіряються автоматично, а більш високорівневі (наприклад, цікавість, естетика) – через людську оцінку або інтеграцію з ігровою симуляцією.

2.3.4 Масштабованість та ефективність генеративних моделей для ігрових рушіїв

Застосування нейронних мереж у реальних ігрових проєктах вимагає розгляду їхньої масштабованості та обчислювальної ефективності. Тренування великих генеративних моделей, таких як VAEs, може займати години або навіть дні на потужних GPU. Хоча це не є проблемою для генерації контенту на етапі розробки, якщо планується динамічна генерація "на льоту" під час гри, продуктивність стає критичним фактором.

Виклики ефективності включають:

- **Розмір моделі:** Великі, складні VAE з багатьма шарами та мільйонами параметрів вимагають значних обчислювальних ресурсів та пам'яті. Це може бути проблемою для мобільних пристроїв або консолей з обмеженими ресурсами. Оптимізація архітектури, використання легких моделей або дистиляція знань (knowledge distillation) є потенційними рішеннями.
- **Час генерації:** Хоча декодування нового зразка з VAE зазвичай є швидким, якщо модель генерує складні рівні або велику кількість рівнів одночасно, сукупний час може бути значним.

- **Інтеграція в ігровий цикл:** Система генерації повинна працювати асинхронно або бути достатньо швидкою, щоб не викликати "фризів" або затримок в ігровому процесі. Це може вимагати розміщення моделі на окремому сервері або використання оптимізованих бібліотек для висновку нейронних мереж (наприклад, ONNX Runtime, TensorRT).

2.4 Вдосконалення контролю та забезпечення якості згенерованих рівнів: роль зворотного зв'язку та гібридних систем

Питання контрольованої генерації та забезпечення стабільно високої якості згенерованих ігрових рівнів залишається одним із найбільш гострих у сфері процедурної генерації контенту за допомогою нейронних мереж. Незважаючи на обіцянки глибинного навчання, повна автономія генеративних моделей часто призводить до непередбачуваних результатів. Це спонукає дослідників до пошуку нових методологій, що поєднують потужність нейронних мереж із можливостями цілеспрямованого впливу та оцінки.

2.4.1 Концепція "людина в циклі" (Human-in-the-Loop) у PCG

Одним із найбільш перспективних напрямків є концепція "людина в циклі" (Human-in-the-Loop, HIL). Це підхід, де людський експерт (ігровий дизайнер, художник) активно взаємодіє з генеративною системою, надаючи зворотний зв'язок, спрямовуючи процес генерації або коригуючи результати. У випадку з VAE, дизайнер може не просто генерувати рівні випадковим чином, а робити такі речі:

А) **Направляти пошук у латентному просторі:** Використовувати інтерфейси, що дозволяють візуалізувати латентний простір і переміщатися по ньому, щоб знайти бажані варіації рівня]. Наприклад,

зміна однієї компоненти латентного вектора може призвести до збільшення кількості ворогів, а іншої – до зміни текстури стін. Завдання полягає в тому, щоб зробити ці "виміри" зрозумілими для дизайнера.

B) **Коригувати згенеровані рівні:** Якщо VAE генерує рівень з дрібними артефактами або небажаними елементами, дизайнер може вручну їх виправити, а потім цей "виправлений" рівень може бути використаний для донавчання (fine-tuning) моделі, що дозволить їй з часом краще розуміти бажані характеристики.

C) **Надавати переваги та оцінки:** Системи PCG можуть генерувати декілька варіантів рівнів, а дизайнер обирає найкращі. Цей вибір може бути використаний для навчання "функції придатності" або "функції переваг", яка потім може бути інтегрована в процес навчання або оптимізації генеративної моделі. Це дозволяє моделі вчитися суб'єктивним уподобанням дизайнера.

D) **Інтерактивна еволюція:** Поєднання VAE з еволюційними алгоритмами, де людина виступає в ролі "селектора", обираючи найбільш перспективні згенеровані варіанти, які потім "схрещуються" та "мутують" у латентному просторі VAE для створення нових поколінь рівнів. Це дозволяє досліджувати величезний простір можливих рівнів, керованих людською інтуїцією.

Підхід HIL визнається ключовим для подолання "проблеми контролю" у генеративних моделях, оскільки він дозволяє поєднати ефективність автоматизації з креативним інтелектом людини, який часто не може бути повністю формалізований в алгоритмах.

2.4.2 Розширені методи контролю латентного простору VAE

Поза базовими CVAE, дослідники активно працюють над вдосконаленням інтерпретації та контролю латентного простору VAE. Це не просто академічний інтерес, а пряма відповідь на потреби ігрових дизайнерів.

Дискретні VAE (Discrete VAEs): Для ігрових рівнів, що складаються з дискретних елементів (плитки, об'єкти), стандартні VAE, які працюють з безперервним латентним простором, можуть мати проблеми з генерацією чітких, дискретних структур. Модифікації VAE, що використовують дискретні латентні змінні, дозволяють моделі краще працювати з такими типами даних, що може призвести до більш чітких та менш "артефактних" рівнів.

Ієрархічні VAE (Hierarchical VAEs): Складні ігрові рівні мають ієрархічну структуру (наприклад, рівень складається з кімнат, кімнати – з плиток та об'єктів). Ієрархічні VAE (HVAE) намагаються моделювати цю структуру, маючи кілька рівнів латентних змінних, де вищі рівні кодують глобальні властивості рівня, а нижчі – локальні деталі. Це дозволяє генерувати рівні з більшою внутрішньою послідовністю та логікою, а також, потенційно, надає більш гранульований контроль над різними рівнями абстракції дизайну.

Навчання причинно-наслідкових зв'язків у латентному просторі: Деякі передові дослідження зосереджені на тому, щоб не просто "розплутати" латентний простір, а й навчити модель причинно-наслідковим зв'язкам між атрибутами рівня та ігровим досвідом. Наприклад, як зміна кількості монстрів впливає на складність, або як розміщення певних об'єктів впливає на прохідність. Це дозволяє VAE генерувати рівні, які не лише відповідають заданим умовам, а й "розуміють", чому вони є такими.

Ці розширені архітектури та підходи є надзвичайно перспективними для створення VAE-систем, які можуть генерувати складні, високоякісні ігрові рівні з високим ступенем контролю для дизайнера.

2.4.3 Оцінка згенерованого контенту: перехід від статичних метрик до динамічної валідації

Проблема оцінки якості згенерованих рівнів виходить за рамки простих об'єктивних та суб'єктивних метрик. Для повноцінної валідації необхідно інтегрувати оцінку безпосередньо в ігровий процес.

Агенти, що грають: Замість того, щоб покладатися лише на людських тестерів, можна використовувати інтелектуальних агентів (наприклад, на основі посиленого навчання), які грають у згенеровані рівні. Ці агенти можуть надавати кількісні дані про прохідність, час проходження, кількість смертей, збір предметів тощо. Це дозволяє автоматизувати значну частину процесу тестування та швидко відсіювати нефункціональні рівні .

Симуляція ігрового процесу: Розширені симуляції ігрового процесу, що імітують поведінку гравця, можуть бути використані для оцінки балансу, складності та навіть "веселощів" рівня. Симуляція дозволяє тестувати рівні у різних умовах та з різними стилями гри, що дає більш повну картину їхньої якості.

Метрики, засновані на сприйнятті: На додаток до чистих ігрових метрик, можна розробляти метрики, що базуються на сприйнятті людини, використовуючи, наприклад, моделі когнітивної психології або навіть біометричні дані (якщо це можливо) для оцінки емоційної реакції гравця на рівень. Хоча це складна галузь, вона є ключовою для розуміння "цікавості" контенту.

Загалом, майбутнє оцінки якості згенерованих рівнів полягає у розробці комплексних систем, що поєднують автоматизоване тестування за допомогою інтелектуальних агентів, детальну симуляцію ігрового процесу та інтерактивний зворотній зв'язок від людських дизайнерів. Такий багаторівневий підхід дозволить досягти нового рівня якості та надійності в процедурній генерації контенту.

Висновки до розділу 2

Було проведено детальний аналіз існуючих рішень та методологічних прогалин у сфері автоматизованої генерації ігрових рівнів на основі нейронних мереж. Критичний огляд сучасних підходів, включаючи застосування GANs, RNNs та VAEs, виявив їхні переваги та, що не менш важливо, значні обмеження. Зокрема, було підкреслено проблеми, пов'язані з нестабільністю навчання, режимним колапсом, недостатньою контрольованістю генеративного процесу та труднощами в інтерпретації складних латентних просторів.

Глибокий аналіз викликів контрольованої генерації виявив, що традиційні підходи, такі як CVAE або використання дизентангованих латентних просторів, не завжди надають достатній рівень контролю, що вимагає розробки більш досконалих методологій. Було також детально розглянуто різні підходи до представлення ігрових рівнів для нейронних мереж, виявивши, що конвертація згенерованих даних у формат, придатний для ігрового рушія, є окремим, але критично важливим завданням.

РОЗДІЛ 3. ПРОЦЕС РОЗРОБКИ НЕЙРОМЕРЕЖІ ДЛЯ ГЕНЕРАЦІЇ ІГРОВИХ РІВНІВ

3.1 Постановка задачі і загальні вимоги до нейромережі для генерації ігрових рівнів

Задача розробки полягає в тому, щоб створити нейромережу, яка має змогу навчатися на вже готових ігрових рівнях а потім генерувати нові рівні самостійно. Вхідні дані нейромережі для навчання - масиви, в кінцевому результаті вони будуть візуалізовані. В ході роботи я буду використовувати дві моделі нейромережі - класичний автокодер та варіаційний автокодер (VAE). Також мені знадобиться генератор ігрових рівнів без нейромережі, але з мінімальними умовами генерації, щоб отримати велику кількість матеріалу для навчання нейромережі.

Нейромережа буде генерувати ігровий рівень для 2D-гри з виглядом зверху. Але схожим методом також можна генерувати рівні для 3D-ігор. Для початку стиль рівнів буде в піксельній графіці, для того, щоб для початку створити мінімально життєздатний продукт, але в розділі (ТУТ ПОСИЛАННЯ) буде розглянуто, як таким чином генерувати набагато складніші рівні. Також, рівні будуть доволі прості. Задача полягає в тому, щоб навчити нейромережу генерувати рівні який містить в собі ландшафт, на якому будуть такі елементи , як річки, озера, гори, населені пункти, та мости.
Структура проекта виглядає наступним чином:
1) generator.py - генератор ігрових рівнів з мінімальними правилами генерації, щоб отримати велику кількість даних. Це дані будуть використовуватися як референс для майбутніх ігрових рівнів.
2) autoencoder.py - класичний автокодер , який буде займатися реконструкцією конкретного прикладу. Він потрібен щоб порівняти роботу різних моделей нейромереж.

3) autoencodervae.py - варіаційний автокодер, який буде мати змогу створювати нові рівні та постійно навчатися зберігаючи свій прогресс.

4) converter.py - потрібен для того, щоб конвертувати файли формуту .json в .npy, для того, щоб варіаційний автокодер міг “прочитати” ці дані та навчатися на них.

5) Каталог Dataset1 - зберігає згенеровані генератором ігрові рівні в форматі масивів.

6) Каталог Datasetvae - зберігає файли формату .npy які були створені за допомогою конвертера

7) Каталог Outputvae - зберігає всі покоління процесу навчання варіативного автокодера. Потрібен для того, щоб відстежувати прогрес

8) Каталог Outputnew - зберігає нові згенеровані рівні.

3.2 Аргументація вибору методу нейромережі для розв’язання задачі генерації рівнів

Щоб генерувати нові карти, я вибрав варіаційний автокодер. Тому що ця модель вчиться не просто відновлювати дані, а створювати прихований простір, де кожна точка відповідає певній карті. Завдяки особливому методу навчання, що має свій прихований простір в вигляді даних, модель вчиться генерувати нові, раніше невідомі карти, беручи випадкові точки з цього простору. VAE досить стабільний у навчанні і працює навіть якщо даних не дуже багато. Також він добре підходить для генерації зображень, які потім можна представити в вигляді ігрового рівня, якщо до цих зображень мати відповідні масиви.

Ігровий рівень завжди має координати, а кожен об’єкт на рівні - свій ідентифікатор, наприклад ID або класс. Якщо представити рівень абстрактно, то

кожен ”крок” координат це 1 піксель, що є найдрібнішою одиницею цифрового зображення. Тому ми можемо представити координати в вигляді масиву, де кожен елемент - це конкретна координата, а розмір масиву залежить від роздільної здатності зображення гри, такі як: 8-bit, 16-bit, 32-bit і тд.

Тому, кожен елемент масива може містити якусь інформацію, і в цьому випадку ця інформація є ідентифікатором предмета, який розташований по тим координатам, які відповідають його елементу в масиві.
Наприклад, ми маємо гру, де ігрова карта має розмір 32 на 32 пікселя, на ігровому рівні два елемента - дерево та кущ. Кущ має ідентифікатор “1”, дерево “2”, їх розмір - однаковий, 4 на 4 пікселя, в масиві це буде виглядати таким чином:

[1,1,1,1,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

[1,1,1,1,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

[1,1,1,1,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

[1,1,1,1,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

(для прикладу показано тільки перші 4 пікселя по ординаті)
Для роботи такого прикладу потрібно щоб нейромережа мала уяву про те, що елементи мають розмір 4 на 4, та розуміла як їх правильно розташувати, в іншому випадку - вони будуть накладатися один на одного.

Тому для дослідження обрано саме такий метод, але так як генеруватися будуть піксельні рівні, елементи в масивах повинні виглядати як вектор кольору RGB, що містить інформацію про те, який колір потрібно намалювати на ігровому рівні Наприклад, [50, 120, 80] - елемент масиву який містить в собі темно-зелений відтінок.

3.3 Етапи створення моделі генерації ігрових рівнів

1)Для навчання нейромережі потрібно багато прикладів, і замість того, щоб створити їх власноруч, що займе багато часу, був написаний простий алгоритм.

Він генерує карти за допомогою шума Перліна, та для кожної висоти він має свій колір:

```
def get_terrain_color(value):  2 usages
    if value < -0.15:
        return (0, 40, 100)
    elif value < -0.05:
        return (0, 70, 150)
    elif value < 0.05:
        return (50, 120, 80)
    elif value < 0.2:
        return (80, 160, 80)
    elif value < 0.35:
        return (110, 130, 80)
    elif value < 0.5:
        return (100, 100, 100)
    else:
        gray = int(160 + (value - 0.5) * 180)
        return (gray, gray, gray)
```

Рис. №1 Функція надання кольору пікселям

Потім він генерує ігровий рівень, та зберігає весь рівень як масив в форматі JSON, в якому кожен елемент - це вектор кольору RGB, також кожен елемент дорівнює одному пікселю.

```
for y in range(HEIGHT):
    for x in range(WIDTH):
        nx, ny = x / WIDTH, y / HEIGHT
        terrain = terrain_noise([nx, ny])
        color = get_terrain_color(terrain)
        pygame.draw.rect(screen, color,  rect: (x * PIXEL_SIZE, y * PIXEL_SIZE, PIXEL_SIZE, PIXEL_SIZE))
        terrain_map[y][x] = terrain
        if terrain < -0.1:
            water_map[y][x] = True
```

Рис.№2 Алгоритм генерації озер,річок та ланшафту

В алгоритмі можна завдати кількість генерацій, тому потрібну кількість прикладів для нейромережі можна згенерувати за один раз.

Також він візуалізує створені рівні. Вони мають такий вигляд:

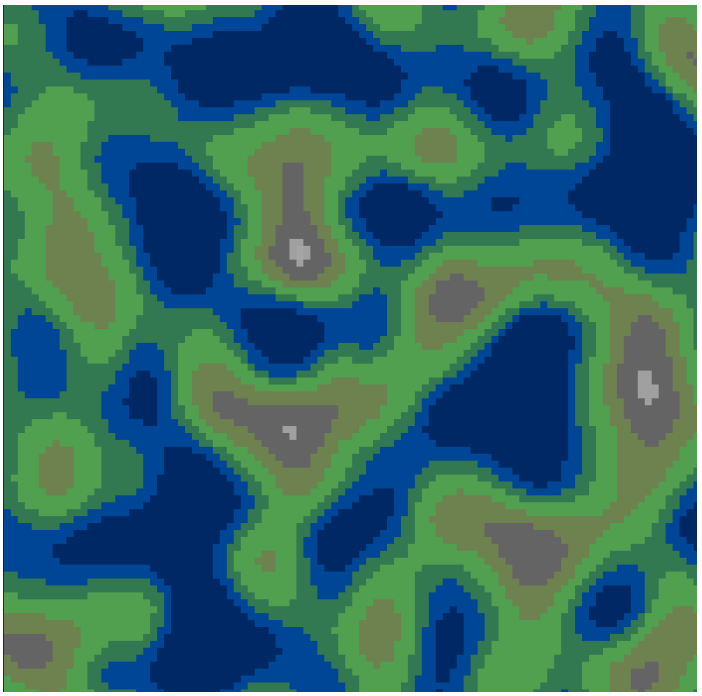

Рис. №3 Приклад згенерованого ландшафту

Міста з синім відтінком - це річки та озера. Зелений відтінок - це ландшафт, все інше - гори.

2) Створення класичного автокодера. Було створено автокодер, який складається з двох частин: енкодера і декодера. Енкодер стискає вхідне зображення, яке представлено у вигляді масиву, у компактний код — короткий опис основних рис. Потім цей код трохи змінюється за допомогою випадкового шуму, щоб зробити модель гнучкішою. Декодер розпаковує цей код назад у зображення. Модель навчається так, щоб відтворювати вхідні картинки якнайкраще, використовуючи дві втрати: одна відповідає за якість відновлення, а друга за те, щоб коди мали певний зрозумілий розподіл. Але класичний автокодер не може генерувати нові зображення, він просто стискати й відновлювати вже відомі дані. Його основна задача — навчитись стискати й відновлювати вже відомі дані. Він просто копіює те, що бачив, і не створює нічого нового. А основна задача в ході дослідження - порівняти його з іншими методами нейромереж.

3) Створення конвертера файлів .json в .npy. Він був створений для того, щоб перетворити дані з формату JSON у формат, зручний для нейромережі — тобто у формат .npy (NumPy-матриці). NPY — це спеціальний формат для збереження масивів NumPy, який набагато швидше завантажується, займає менше пам’яті та не потребує парсингу тексту. Конвертер також нормалізує дані — ділить значення пікселів на 255, щоб перевести їх у діапазон [0, 1]. Це необхідно для стабільного навчання нейромережі.

4) Побудова варіаційного автокодера(VAE). Варіаційний автокодер було створено з метою автоматичної генерації нових ігрових карт на основі вже існуючих прикладів. Головна перевага цього підходу в тому, що після навчання модель може не просто відновлювати побачені приклади, а й створювати нові, схожі на них, комбінуючи знання про вже існуючі карти. У рамках реалізації моделі було використано TensorFlow - одну з найпопулярніших бібліотек для машинного навчання. Також, важливо зазначити, що нейромережа має пам’ять, так як створює файл в форматі .h5, в який записує та з якого читає інформацію.

Рис. №4 Покроковий процес роботи VAE

3.4 Тестування роботи програмного забезпечення та нейромереж

Для початку було створено багато даних за допомогою алгоритму для генерації рівнів. Вони зберігаються в .json форматі, а сам процесс генерації

візуалізується, та створює рівні як на Рис.№ . Згенеровано 500 файлів формату .json, цієї кількості повинно бути достатньо для навчання нейромережі. Вони автоматично були завантажені в теку “Dataset1” та пронумеровано він 0 до 499.

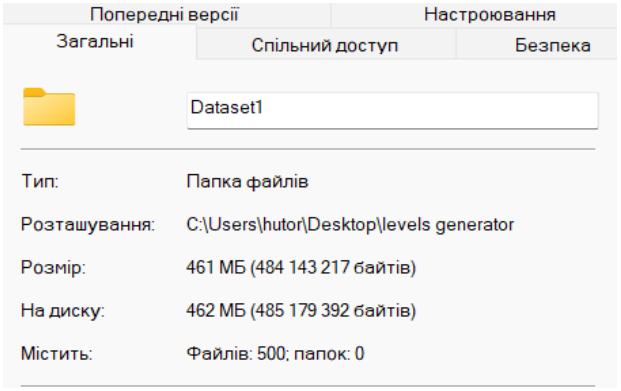

Рис. № 5 Папка з згенерованими масивами

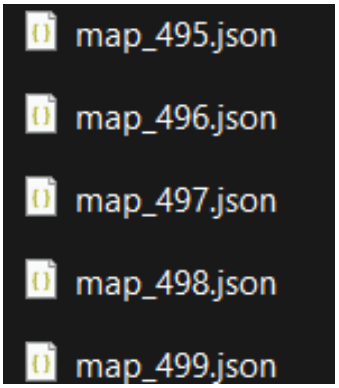

Рис. №6 Масиви

Загальна вага бази даних склала 461 мегабайт.

Потім було запущено класичний автокодер, який завантажив в свою пам’ять всі 500 файлів, та почав обробляти інформацію. Після чого він почав навчатися. Для дослідження було вказано 300 ітерацій, що означає, що в кінцевому результаті буде отримано 300 поколінь нейромережі, кожна з яких буде краща за попередню, також буде отримано результат кожного покоління.

В консолі почала з’являтися інформація про кожне покоління, а в теці “Output” результат кожного покоління.

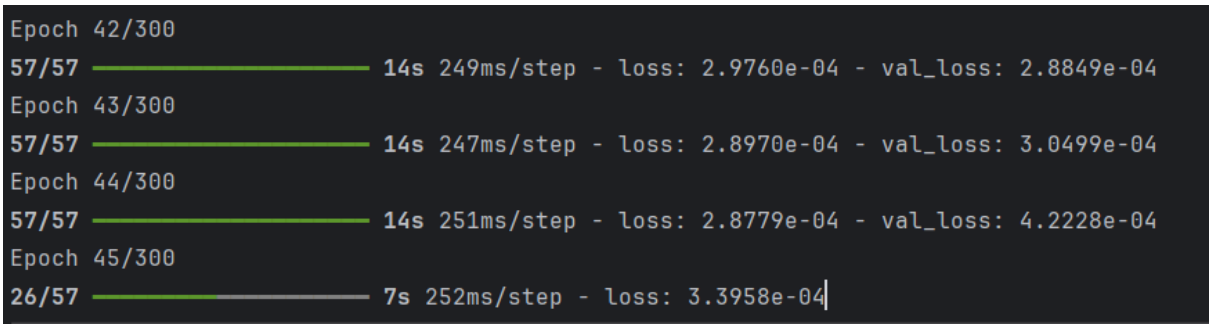

Рис.№7 Вивід в консоль при роботі автокодера

Loss — функція втрат, найважливіший параметр. Це числове значення помилки, яке показує, наскільки погано (або добре) модель відновлює вхідні дані. У класичному автокодері loss показує різницю між оригінальним зображенням і відновленим. Чим менше значення loss, тим краще модель виконує своє завдання. Чим менше значення - тим краще. На Рис.№ можна побачити що значення зменшується з кожним поколінням. Але іноді воно більше чим в попередньому - це теж нормально, так як це дає змогу нейромережі робити роботу над помилками. На 44 поколінні значення втрат залишається величезним, але головне, що воно стабільно зменшується.

Step - швидкість обробки даних.

val loss — це перевірочна помилка: наскільки добре модель справляється з даними, які вона не бачила раніше.

Ось результат першого покоління. Видно схожість з Рис.№3 та подібних, на даних яких нейромережа навчається, але якість важко назвати хорошою. Рівень розмитий та немає жодних деталей, а також гір.

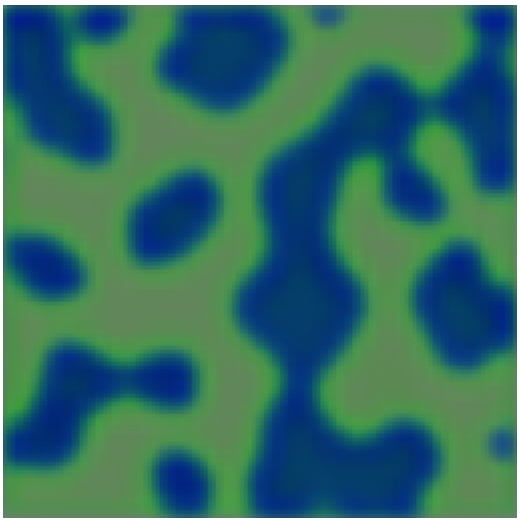

Рис. №8 Перше покоління автокодера

А ось результат покоління №50 та №80. Loss: 2.6076e-04 1.9192e-04 відповідно.

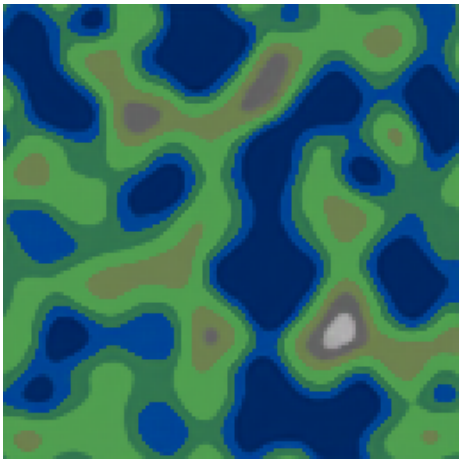

Рис.№ 9 50 покоління автокодера

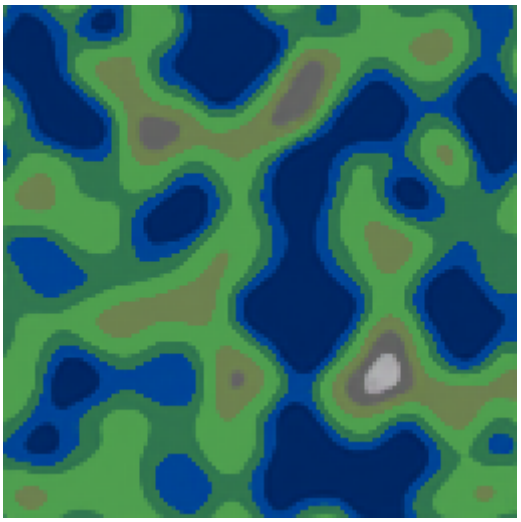

Рис.№ 10 80 покоління автокодера

Вже видно деталі, та деяку схожість з оригіналом, але між собою вони майже не відрізняються. В цьому і є суть класичного автокодера. Він не створює нового результату, він бере один приклад, і намагається довести його до ідеалу. Всі 300 поколінь мають схожу структуру та відрізняються лише деталями. Але все ж таки іноді класичний автокодер може створювати щось нове, при умові, що вхідні дані будуть відрізнятися від тих, що він бачив до цього. Але навіть в такому випадку, буде створено дуже схожий рівень, так як до цього вже бачив велику кількість інформації.

Після покоління №300 було виведено таке повідомлення, яке показує те, що пам’ять нейромережі збережена, та при наступному включенні програми ця пам’ять буде завантажена:

```
Epoch 300/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 18s 322ms/step - loss: 4.8538e-05 - val_loss: 6.1240e-05
Модель збережена: map_generator_model.keras
```

Рис.№11 Закінчення роботи автокодера

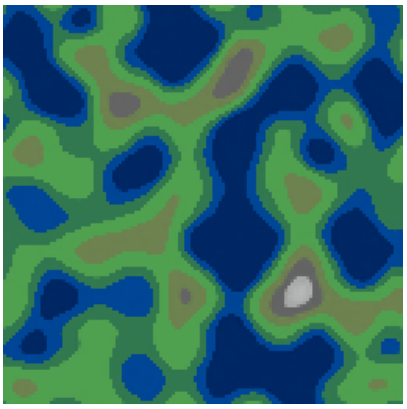

Рис. № 12 Останній(300) згенерований рівень автокодером

Дуже схоже на рівні, які використовувались як вхідні дані (Рис.№).
Також це покоління майже не відрізняється від покоління №80, тому скоріш за все, 100 ітерацій було б достатньо.

Перейдемо до тестування варіаційного автокодера. Але перед цим необхідно використати конвертер json в npy, для коректної роботи нейромережі. Буде використано те ж самий набір вхідних даних, який складає 500 файлів у форматі json.

```
Збережено Datasetvae\map_494.npy, формат: (256, 256, 3)
Збережено Datasetvae\map_495.npy, формат: (256, 256, 3)
Збережено Datasetvae\map_496.npy, формат: (256, 256, 3)
Збережено Datasetvae\map_497.npy, формат: (256, 256, 3)
Збережено Datasetvae\map_498.npy, формат: (256, 256, 3)
Збережено Datasetvae\map_499.npy, формат: (256, 256, 3)
```

Рис. №13 Збереження масивив в форматі .npy, вивід в консоль

Тут можна побачити слово “формат”, перше та друге значення якого, є розміром масива, який в цьому випадку є 256 на 256, тому ігрові рівні мають по 256

пікселів в довжину та ширину. Третє значення вказує кількість каналів кольору, так як в ході наукової роботи в значеннях елементів масиву використовується кольорова модель RGB, то кольорів - 3, це основні кольора: червоний, зелений, синій. Цей формат потрібен для того, щоб нейромережа мала змогу правильно обробити інформації. Тепер в теці “Datasetvae” знаходиться 500 файлів формату npy, тому був запущений варіаційний автокодер. Він має два вихідних значення:

А) Порівняння вхідних і вихідних даних
В) В кінці ітерацій всіх поколінь нейромережа генерує задану кількість абсолютно нових ігрових рівнів

Нейромережі завдано навчатися 200 поколінь, після чого згенерувати 30 нових рівнів.

Все схоже з класичним автокодером, але з’явилося два нових значення:

1) kl_loss(Kullback-Leibler Divergence) - це можна назвати “штрафом” для нейромережі, таким чином вона розуміє, як добре вона впоралась зі своєю роботою.
2) reconstruction_loss - це те ж саме що і loss, але для реконструкції, чим менше - тим краще вийшла реконструкція вхідних даних.

Ця нейромережа працює трохи інакше чим класична, але алгоритм в них має деяку схожість, тому що використовувалась одна й та сама бібліотека машинного навчання - TensorFlow.

Але відрізняються вони тим, що класичний автокодер відразу бере значення всіх вхідних даних, та на їх основі намагається покращити генерацію одного рівня, а варіативний - по черзі , він ітерує вхідні дані по черзі, та намагається повторити рівень. Кожне покоління робить 4 ітерації.

Таким чином VAE дійсно навчається, тому що обробляє різні варіанти даних, саме через це, в кінцевому результаті він може створити щось унікальне.

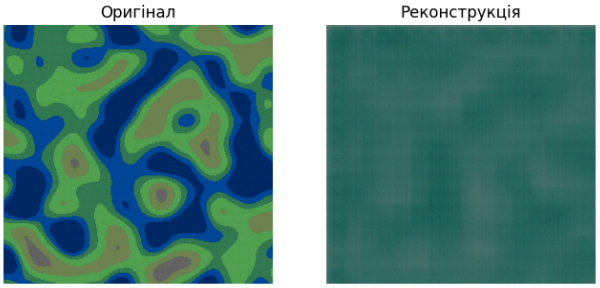

Рис.№14 Покоління 1 VAE

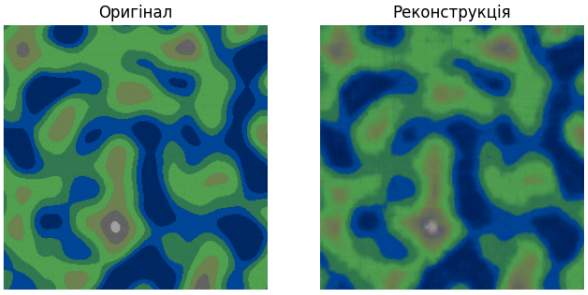

Рис.№15 Покоління 100 VAE

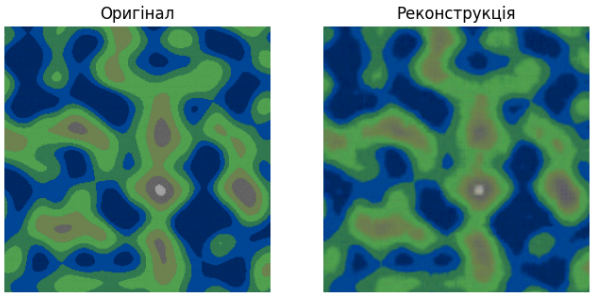

Рис.№16 Покоління 150 VAE

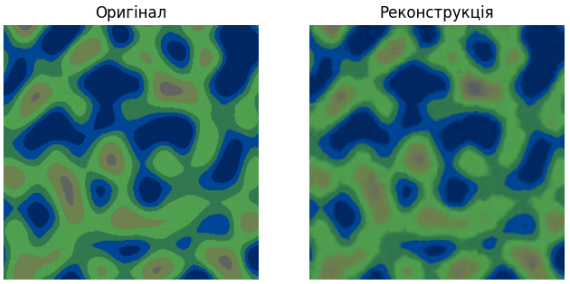

Рис.№17 Покоління 200 VAE

Як можна побачити, за 200 ітерацій нейромережа навчилась доволі точно створювати схожі зображення, трохи не вистачає якості, але вона може навчитися за більшу кількість поколінь.

Також, як було зазначено, варіативний автокодер не просто намагається навчитися повторювати дані, а й створювати схожі. Було згенеровано декілька зображень:

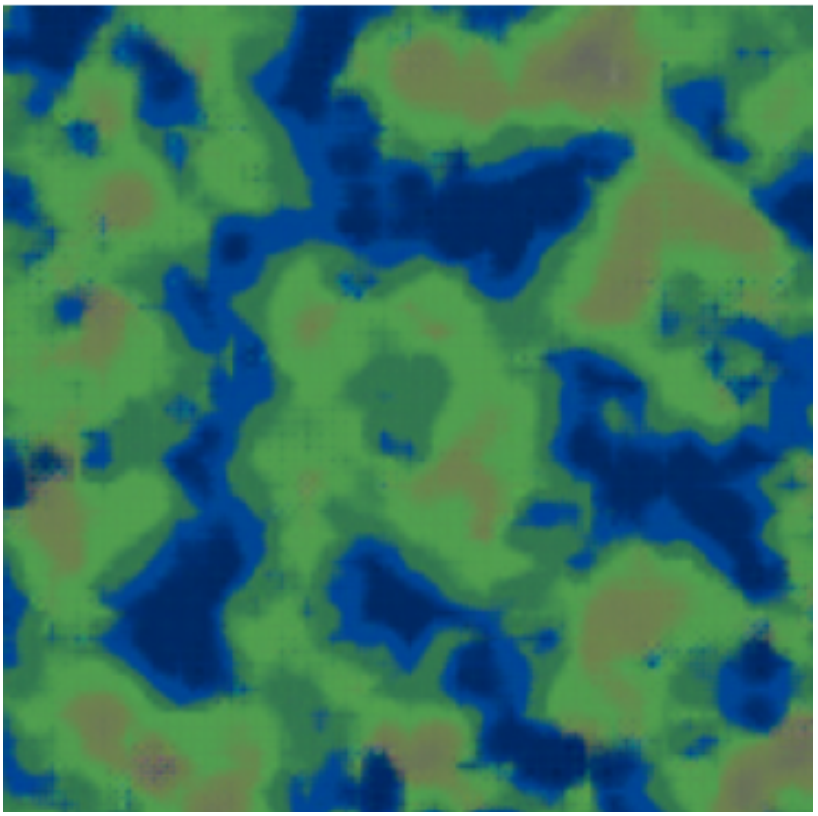

Рис.№18 Згенерований рівень за допомогою VAE

На цих вихідних даних є артефакти, але це нормально, 200 поколінь - недостатньо для ідеального результату. Але можна помітити, що стиль даних залишається таким самим, але це вже повністю згенеровані зображення.

3.5 Засоби інтеграція схожої нейромережі в реальну гру

На цей момент ця нейромережа схожа на прототип через те, що не виглядає як ігровий рівень. Але справа в тому, що окрім візуалізації зображень, нейромережа також створює масив, який можно використовувати в різних ігрових двигунах, таких, як: Unity, Godot, Unreal Engine. І вся інформація може

зберігатися в масивах, тому окрім візуальних даних елементи масиву можуть містити в собі інформацію про сам об’єкт та його взаємодію з гравцем. В цьому прикладі формат який використовує нейромережа має вигляд:(256, 256, 3), що містить в собі інформацію тільки про розмір та кольорову палітру, але елементи можуть також містити такі дані, як: чи може персонаж пройти через об’єкт, чи може персонаж взаємодіяти з об’єктом, якщо так, то як?

Тому цей прототип може бути використаний для реальної генерації ігрових рівнів, і все що потрібно змінити - це формат елементів масиву в вхідних даних та обробку цих даних в програмному коді нейромережі.

Висновки до розділу 3

Було реалізовано та досліджено прототип системи автоматизованої генерації ігрових рівнів на основі варіаційного автокодувальника (VAE), враховуючи висновки попередніх теоретичних та аналітичних розділів. Детально описано етапи підготовки даних та архітектуру використаного VAE. Проведені експерименти підтвердили здатність розробленої моделі генерувати нові, унікальні, але стилістично та структурно відповідні тренувальним даним ігрові рівні. Було відзначено наявність візуальних артефактів на початкових етапах навчання, що вказує на потенціал для покращення з більшою кількістю ітерацій.

ВИСНОВКИ

У цій кваліфікаційній роботі було успішно вирішено актуальну задачу автоматизованої генерації ігрових рівнів із застосуванням сучасних методів глибинного навчання, що має значне теоретичне та практичне значення для ігрової індустрії.

На першому етапі було проведено ґрунтовний аналіз теоретичних засад процедурної генерації контенту, висвітлено її еволюцію та ключові переваги для сучасних відеоігор. Детальне вивчення архітектур генеративних нейронних мереж, зокрема GANs та VAEs, дозволило обґрунтувати вибір варіаційного автокодувальника як найбільш перспективного інструменту для досягнення цілей роботи, враховуючи його здатність формувати керований та інтерпретований латентний простір, що є критично важливим для ігрового дизайну.

Аналітичний розділ роботи зосередився на критичному огляді існуючих рішень та виявленні методологічних прогалин у поточних підходах до генерації ігрових рівнів. Були ідентифіковані та проаналізовані ключові виклики, такі як нестабільність навчання, проблема контрольованої генерації, необхідність у великих обсягах якісних даних, а також труднощі з ефективним представленням та валідацією згенерованого контенту. Особливу увагу було приділено перспективі інтеграції концепції "людина в циклі" та застосуванню гібридних систем для подолання обмежень повністю автономних генеративних моделей.

На основі отриманих теоретичних та аналітичних висновків, у практичній частині роботи було розроблено та реалізовано прототип системи генерації ігрових рівнів. Демонстрація згенерованих рівнів підтвердила спроможність варіаційного автокодувальника до створення нових, унікальних, але при цьому стилістично та структурно коректних ігрових мап. Було також доведено можливість ефективної інтеграції згенерованого контенту в ігрові рушії шляхом перетворення вихідних даних моделі у структурований масив, що містить як

візуальну, так і семантичну інформацію про ігрові об'єкти. Це відкриває шлях до практичного застосування подібних систем у реальних проєктах розробки ігор, дозволяючи значно прискорити етап прототипування та збагатити ігровий досвід за рахунок більшої варіативності контенту.

Загалом досягнуто поставленої мети – розроблено та досліджено підхід до автоматизованої генерації ігрових рівнів, що забезпечує баланс між різноманітністю та функціональністю, використовуючи потенціал варіаційних автокодувальників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. І.В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
2. Інновації в ігровому світі: як штучний інтелект змінює гру. *Інтернет-видання "Полтавщина"*. URL: https://poltava.to/news/81338/ (дата звернення: 22.05.2025).
3. Variational Autoencoder. URL: https://paperswithcode.com/method/vae (дата звернення: 20.05.2025).
4. Variational Autoencoder. URL: https://www.sciencedirect.com/topics/materials-science/variational-autoencoder (дата звернення: 24.05.2025).
5. Variational autoencoders learn transferrable representations of metabolomics data. URL: https://www.nature.com/articles/s42003-022-03579-3 (дата звернення: 18.05.2025).
6. Improving VAE based molecular representations for compound property prediction. *Journal of Cheminformatics*. URL: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-022-00648-x (дата звернення: 25.05.2025).
7. Cloud-VAE: Variational autoencoder with concepts embedded. URL: https://www.sciencedirect.com/science/article/abs/pii/S0031320323002303 (дата звернення: 26.05.2025).
8. Python 3.13.4 documentation. URL: https://docs.python.org/3/ (дата звернення: 26.05.2025).
9. Konvey D. Machine Learning for Hackers. New York: O'Reilly Media, 2012. 300 с. (дата звернення: 28.05.2025).
10. What Is Machine Learning? Definition, Types, Applications, and Trends. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/ (дата звернення: 28.05.2025).

11. What is machine learning?. URL: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-machine-learning (дата звернення: 29.05.2025).
12. What is a neural network?. URL: https://www.ibm.com/think/topics/neural-networks (дата звернення: 21.05.2025).
13. What is a Neural Network?. URL: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/ (дата звернення: 26.05.2025).
14. What is a Neural Network?. URL: https://www.cloudflare.com/ru-ru/learning/ai/what-is-neural-network/ (дата звернення: 23.05.2025).
15. Explained: Neural networks. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (дата звернення: 26.05.2025).
16. What Is a Neural Network? Definition, Working, Types, and Applications in 2022. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/ (дата звернення: 29.05.2025).
17. 22 Great Articles About Neural Networks. URL: https://www.datasciencecentral.com/22-great-articles-about-neural-networks/ (дата звернення: 20.05.2025).
18. Dynamic neural networks: advantages and challenges. URL: https://academic.oup.com/nsr/article/11/8/nwae088/7624214 (дата звернення: 22.05.2025).
19. An Overview of Neural Network. URL: https://www.sciencepublishinggroup.com/article/10.11648/j.ajnna.20190501.12 (дата звернення: 19.05.2025).
20. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (дата звернення: 19.05.2025).

21. Network properties determine neural network performance. URL: https://www.nature.com/articles/s41467-024-48069-8 (дата звернення: 22.05.2025).
22. Procedural Level Generation. URL: https://www.stevetech.org/blog/procedural-level-generation (дата звернення: 23.05.2025).
23. Level Generation Through Large Language Models. URL: https://dl.acm.org/doi/10.1145/3582437.3587211 (дата звернення: 23.05.2025).
24. Documenting Python Code: A Complete Guide. URL: https://realpython.com/documenting-python-code/ (дата звернення: 26.05.2025).
25. Neural network. URL: https://www.britannica.com/technology/neural-network (дата звернення: 29.05.2025).
26. Procedural Content Generation for General Video Game Level Generation. URL: https://journal.iberamia.org/index.php/intartif/article/view/681 (дата звернення: 25.05.2025)
27. What Is a Neural Network?. URL: https://www.investopedia.com/terms/n/neuralnetwork.asp (дата звернення: 24.05.2025).
28. Neural networks. URL: https://www.jstor.org/stable/24096496 (дата звернення: 30.05.2025).
29. What are Neural Networks?. URL: https://www.datacamp.com/blog/what-are-neural-networks (дата звернення: 29.05.2025).
30. What is deep learning?. URL: https://www.ibm.com/think/topics/deep-learning (дата звернення: 27.05.2025).
31. Current Advances in Neural Networks. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-statistics-040220-112019 (дата звернення: 19.05.2025).
32. Neural Networks (NN). URL: https://www.interaction-design.org/literature/topics/neural-networks?srsltid=Af

mBOopT6rtHHgrSAH-wpZGErUeGY7pbLZdxjuqjnjiLXDfddm_jhoAi (дата звернення: 19.05.2025).