Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

Освітній ступень – бакалавр

На тему: "**Генерація ігрових рівнів з використанням алгоритму колапсу хвильової функції в Unity**"

Виконав: студент 4-го року навчання

Спеціальності: 122 «Комп’ютерні науки»

Освітньої програми: «Комп’ютерні науки»

Стасюк Ілля Вікторович

Науковий керівник: Бучко О. А.,

Кандидат технічних наук

Рецензент: Ім’я, титул

Кваліфікаційна робота

захищена з оцінкою «_____»

Секретар ЕК_____

«____» _____2024 р.

Київ – 2024

Зміст

Вступ

Актуальність

Індустрія ігор добре знайома зі створенням контенту вручну - це кропітка робота, що вимагає великої кількості зусиль та часу.

Процедурний метод натомість полягає в створенні правил, що автоматизують процес створення контенту. Це допомагає значно заощадити робочі зусилля та створювати ігри, в які можна грати неодноразово і нескінченно довго. Проте часто це вимагає глибоких технічних знань та розуміння роботи відповідного алгоритму задля правильного задання таких правил і отримання перебачуваних результатів.

Наступним кроком було б піти ще далі й перетворити творчий намір безпосередньо на контент, інтерпретуючи цей намір та формуючи відповідні правила для генерації вмісту [35]. Це б дало змогу ігровим дизайнерам інтуїтивно створювати контент без необхідності освоєння складних технічних навичок. Використання такого методу значно оптимізує процес розробки ігрових рівнів, що є особливо важливим для невеликих інді ігор, які стають все дедалі популярнішими.

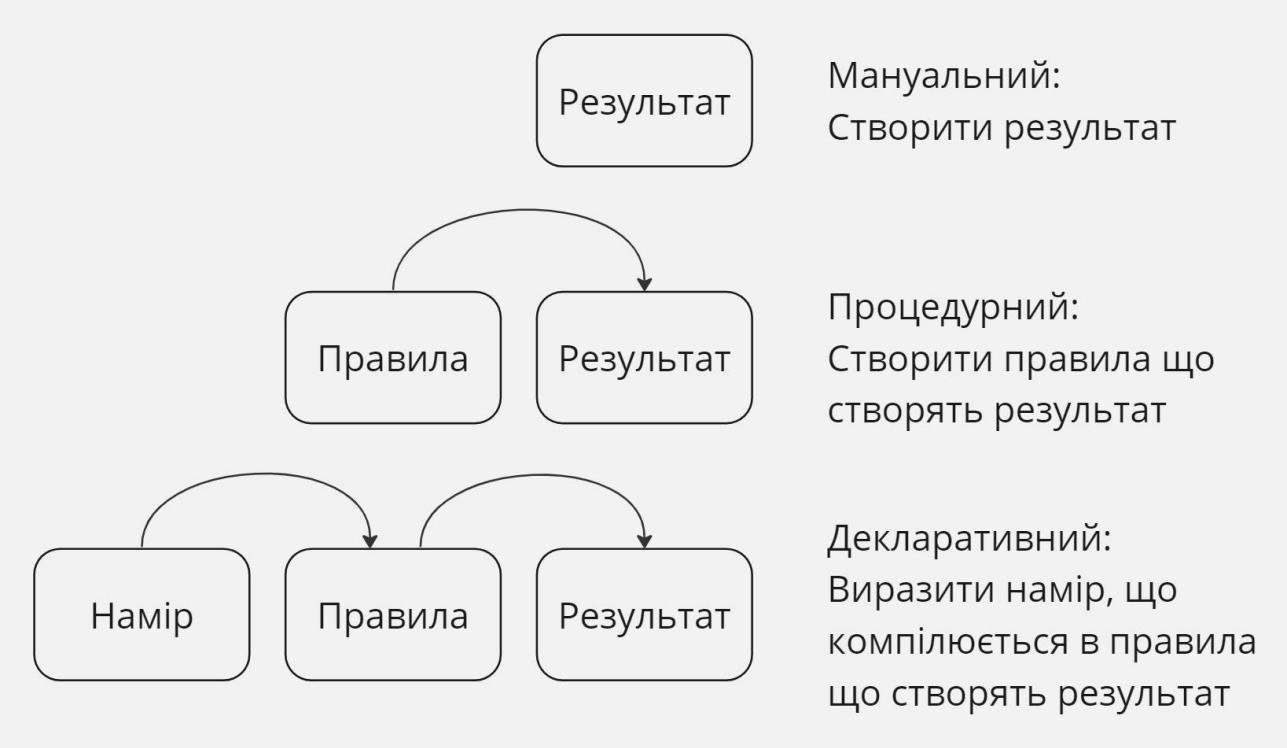

Рисунок 1

Wave Function Collapse, або алгоритм колапсу хвильової функції – це новий ефективний метод, що зміг наблизитись до декларативної генерації контенту, а зокрема ігрових рівнів. Він використовує інформацію, отриману із наданого користувачем зразка, щоб створити подібний, але новий результат, при цьому зберігаючи всі важливі риси.

Об’єкт дослідження

Об'єктом дослідження є процес генерації ігрових рівнів з використанням алгоритмів.

Предмет дослідження

Предметом дослідження є алгоритм колапсу хвильової функції та його застосування для створення ігрових рівнів.

Мета дослідження

Мета дослідження полягає в визначенні переспектив алгоритму колапсу хвильової функції для генерації ігрових рівнів та розробці прототипу системи генерації рівнів у Unity з використанням цього алгоритму.

Завдання дослідження

1. Аналіз існуючих підходів до генерації ігрових рівнів та їх порівняння.
2. Вивчення теоретичних основ алгоритму колапсу хвильової функції.
3. Реалізація прототипу системи генерації рівнів.
4. Тестування та аналіз ефективності розробленої системи.

Розділ 1. Процедурна генерація ігрових рівнів. Теоретичні основи

1.1. Процедурна генерація в іграх

Процедурна генерація ігрових рівнів є однією з найстаріших задач процедурної генерації контенту (англ. procedural content generation - PCG) та першою, яка знайшла практичне застосування в іграх, наприклад, у Rogue (1980), де рівні генерувались у реальному часі. Початково такі методи застосовувались розробниками щоби обійти технічні обмеження і забезпечити більшу кількість контенту на обмеженому обсягу пам'яті.

Сьогодні, процедурна генерація рівнів набула ще більшої важливості у галузі ігрової індустрії та надає широкі можливості для ігрових дизайнерів та розробників. Вона не тільки спрощує процес розробки, але й значно розширює креативні горизонти, дозволяючи створювати унікальні, нескінченні та динамічні світи, а також підтримувати їхню варіативність та свіжість, що є критично важливим для залучення та утримання інтересу гравців. Така технологія забезпечує необмежені можливості для експлорації та інтерактивності, що робить кожен ігровий досвід унікальним та неповторним. Це є особливо важливим, з огляду на різноманіття вимог гравців і зростаючі очікування до якості та кількості контенту. Також процедурна генерація рівнів стала важливим естетичним елементом в деяких типах ігор та є особливо цінною для невеликих розробників ігор.

1.2. Класифікація методів процедурної генерації

Алгоритми процедурної генерації можна розділити на такі чотири категорії [22]:

1. Конструктивні методи (constructive methods)
2. Методи на основі пошуку (search-based methods)
3. Методи машинного навчання (machine-learning methods)
4. Методи на основі обмежень (constraint-based methods)

Конструктивні методи використовують фіксовані алгоритми для генерації ігрового вмісту. Хоча вони можуть ефективно генерувати великі ігрові світи, інколи можуть призводити до генерації неякісного контенту. Такий підхід вимагає численних ітерацій розробки, корекцій та оптимізацій [10].

Методи на основі пошуку застосовують алгоритми оптимізації, такі як генетичні алгоритми, для відшукання рішення, яке відповідає певним критеріям. Ці методи ефективні для налаштування параметрів ігрових світрів під конкретні цілі, але часто вимагають значного часу та обчислювальних ресурсів для генерації.

Методи машинного навчання використовують навчені моделі, такі як нейронні мережі, для створення контенту на основі великих наборів даних. Цей підхід може значно поліпшити якість ігрового контенту, адаптуючи його до поведінки та переваг гравців. Проте, основною перешкодою є потреба у великих і якісних даних, які часто складно зібрати без попереднього ручного створення контенту [10].

Методи на основі обмежень працюють з певними правилами або обмеженнями, що визначають, яким має бути ігровий рівень. Вони дозволяють доволі швидко знаходити рішення, яке точно відповідає встановленим критеріям. Такі алгоритми здатні забезпечити високий рівень контролю та забезпечують стабільність та передбачуваність в генерації.

1.3. Проблема універсальності

При порівнянні різних методів генерації основними критеріями виступають:

- Універсальність
- Передбачуваність
- Контрольованість
- Якість
- Ефективність

- Простота у розробці

Ключовою проблемою у розробці методів процедурної генерації ігрових рівнів є універсальність. Більшість робіт по цій темі (як у наукових колах, так і в індустрії) зосереджена на конкретних іграх. Причинами цього є, по перше, брак розуміння того, як за допомогою сучасних методологій розробити якісні генератори рівнів, які можуть використовуватись у різних іграх. А по-друге, значна кількість роботи, необхідна для створення генератора навіть для одного типу рівня, становить серйозну проблему [6]. Цей останній фактор значною мірою сприяє першому. Розробляти універсальні генератори рівнів виявилося непрактично, оскільки вони неминуче будуть складатися із різних окремих генераторів, кожен із яких потребуватиме значного інвестування ресурсів.

1.4. Алгоритми задоволення обмежень

Алгоритми задоволення обмежень вирішують проблему виконання обмежень (англ. Constraint Satisfaction Problems, CSP). Такі проблеми визначаються як:

- Змінні: Елементи, для яких потрібно визначити значення.
- Домени: Набори можливих значень, які можуть бути призначені змінним.
- Обмеження: Умови, які мають бути виконані між змінними.

Розв’язування такої проблеми полягає у призначення кожній змінній певного значення таким чином, щоб не було порушено жодного з обмежень. Якщо таке порушення відбувається, тобто знайдено конфлікт, алгоритм відступить та спробує інше значення. Таким чином відбувається пошук у просторі часткових рішень. Повне рішення досягається лише тоді, коли всі змінні мають значення, які не порушують обмеження [23].

Для прискорення пошуку, використовуються евристики, які допомагають вибрати наступну змінну або значення, яке має велику ймовірність не вести до порушення обмежень. Це змінює порядок пошуку, але не впливає на здатність знайти рішення, якщо воно існує.

Під час пошуку також застосовується алгоритми поширення або пропагація (англ. propagation) обмежень - тобто виключення значень, які неминуче призведуть до конфліктів у майбутньому. Це ефективно скорочує простір пошуку, дозволяючи уникнути глухі кути без зміни загального порядку пошуку. Добре відомим алгоритмом поширення обмежень є AC3.

В контексті процедурної генерації ігрових рівнів алгоритми задоволення обмежень дозволяють згенерувати рівні, що задовольняють певні ігрові обмеження. До прикладу, баланс між складністю і прохідністю, естетичними вимогами та інтерактивністю. Вони також дозволяють гарантувати, що згенеровані рівні будуть відповідати визначеним критеріям і забезпечувати бажану ігрову динаміку.

1.5. Алгоритми синтезу текстур

Алгоритми синтезу текстур у комп'ютерній графіці займаються створенням великого зображення, що відтворює текстуру меншого зразка. Різні алгоритми досягають цієї мети по-різному. Багато з них фокусуються на перевикористанні маленьких фрагментів з вхідного зображення для формування нового, більшого зображення, роблячи це таким чином, щоби приховати шви між фрагментами. Результатом алгоритмів синтезу текстур є зображення, де кожна ділянка схожа на ділянку вхідного зображення. Таким чином досягається глобальна подібність [7].

Алгоритми синтезу текстур найкраще працюють лише із зображеннями схожими на текстури, оскільки вони базуються на припущеннях, що вхідне зображення задовольняє умови локальності та стаціонарності [15]:

1. Стаціонарність означає що різні фрагменти зображення завжди подібні між собою.
2. Локальність означає що значення кожного пікселя можна визначити за його найближчими сусідами.

Існують різні підходи до синтезу текстур:

- Явна побудова розподілу ймовірностей і семплінг із нього. Як правило, цей підхід є дуже повільним, тому більшість алгоритмів його уникають.
- Вибір пікселя на основі найближчих сусідів. Незважаючи на свою простоту, ці алгоритми працюють надзвичайно добре. Проте у наївній імплементації вони також є повільними, оскільки виконують повний пошук у вхідному семплі для кожного фрагмента згенерованого зображення.
- Генерація за допомогою "патчів" або шматків. Ці алгоритми складають вихідне зображення напряму з різних за розміром частин вхідного зображення. Їхньою основною перевагою є ефективність, яка досягається без компромісу з якістю.

В контексті генерації ігрових рівнів найпривабливішими виглядають саме алгоритми синтезу за допомогою шматків, оскільки вони менше всього спираються на припущення щодо стаціонарності та локальності. Це дозволяє використовувати їх навіть для генерації зображень що мають певну структуру та менш подібні до текстур, як, до прикладу, мапа ігрового світу.

1.6. Особливість алгоритму WFC

Алгоритм Wave Function Collapse виділяється серед інших методів своєю інноваційністю та унікальністю, пропонуючи універсальне та зручне рішення для створення ігрових рівнів.

WFC є методом на основі обмежень. Це забезпечує високу якість та ефективність генерації. Завдяки цьому він ідеально підходить для генерації різноманітних стуктур і може гарантувати точний результат що задовольняє усім вимогам суміжності, чого не здатні досягнути більшість інших алгоритмів [7].

WFC також був натхненний алгоритмами синтезу текстур [1]. Тому іншою його перевагою є декларативність. Користувачі здатні управляти процесом генерації, використовуючи зображення як основу. Такий підхід мінімізує потребу в

ручному роботі для кожного нового рівня, знижуючи технічні бар'єри та спрощуючи процес створення контенту.

Ці особливості пояснюють зокрема широкий академічний інтерес до WFC. Дослідники постійно шукають способи покращення алгоритму та його адаптації для нових застосувань. Не зважаючи на свою новизну, алгоритм WFC уже є важливим інструментом у технологічному та творчому арсеналі сучасних дизайнерів та розробників.

1.7. Висновки

Алгоритми процедурної генерації контенту, зокрема ігрових рівнів, знаходять широке застосування у ігровій індустрії. Однак, основними викликами для їх ефективного використання є обмежена універсальність та складність в управлінні.

Методи, що базуються на обмеженнях, можуть бути ефективними для підвищення універсальності алгоритмів, тоді як алгоритми синтезу текстур забезпечують декларативний підхід до генерації контенту.

Алгоритм Wave Function Collapse (WFC), поєднує обидві ці технології і тому виділяється серед інших методів.

Розділ 2. Алгоритм колапсу хвильової функції

2.1. Термінологія

Зображення (image). У контексті цього дослідження термін “зображення” має ширше значення, ніж просто картинка. Зображення представляє абстрактні вхідні дані з певним просторовим розміщенням, що мають визначену метрику подібності. Кожен елемент вхідних даних є дискретним і належить певній скінченній множині. Це узагальнює вимоги до даних, передбачені алгоритмом WFC. Також це дає змогу візуалізувати ці дані за допомогою певної функції відображення. В оригінальному алгоритмі така функція задає відображення із числового значення в колір.

Зразок (sample) - це зображення, виходячи із якого, алгоритм генерує результат. Головною особливістю алгоритму WFC є саме те, що він не потребує складних налаштувань параметрів і здатен генерувати результат виходячи лише із невеличкого зразка.

Сітка (grid) описує просторове розміщення даних, які задають певним зображенням. Сітка може існувати без зображення, оскільки вона визначає лише просторову структуру даних. Проте кожному зображенню відповідає певна сітка. Найчастіше всього сітка являє собою прямокутний масив. В цьому дослідженні під терміном сітка передбачатиметься саме прямокутний масив.

Клітинка (cell) є складовою одиницею сітки. Кожна клітинка задає місце, що може містити один чи декілька елементів із множини вхідних даних.

Тайл (tile) позначає окремий елемент із множини вхідних даних. Тобто зображення складається із тайлів. В оригінальному алгоритмі кожен тайл відповідає окремому кольору, присутньому у зразку.

Патерн (pattern) - являє собою елементи, що мають задані відношення суміжностей між собою. Суміжність патернів визначає які патерни можуть розташовуватися поруч один з одним. Патерни зазвичай представляють собою невеликі зображення із тайлів розміром N x N. Патерни містять інформацію, яка

дозволяє алгоритму підтримувати зв'язність та логічну структуру під час генерації.

При побудові патернів зазвичай також враховуються різні трансформації із діедричної групи симетрій квадрату. Задля гнучкості, сам алгоритм WFC трактує різні трансформації патернів як окремі патерни.

2.2. Огляд алгоритму

Алгоритм WFC створює зображення, локально подібні на вхідний зразок .

Локальна подібність означає, що [1]

1. Вихідні дані повинні містити лише ті патерни, які присутні у вхідних даних.
2. (Слабка умова) Розподіл патернів на виході повинен бути близьким до щільності таких патернів на вході.

Рисунок 2 Візуалізація умови 1 [1].

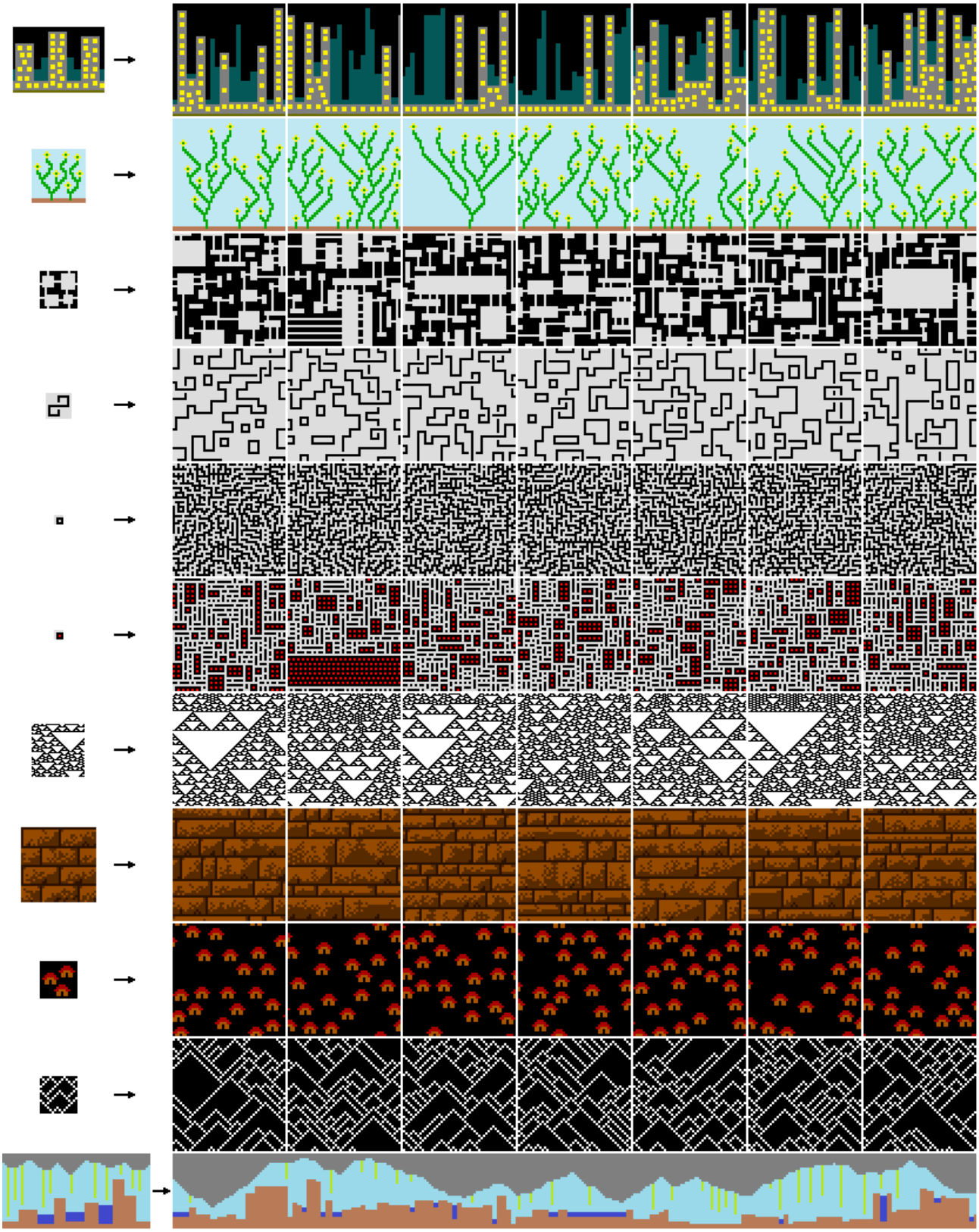

Рисунок 3 Результати роботи алгоритму на різних зразках [1].

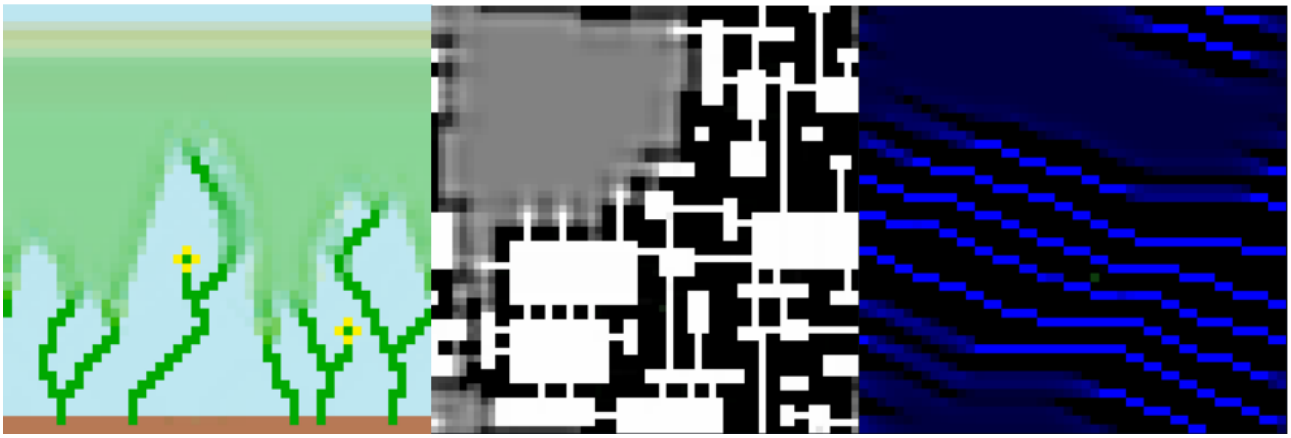

Рисунок 4 Алгоритм в процесі генерації [1].

2.2.1. Види алгоритму

Існують два основні варіанти алгоритму WFC: модель з перекриттями (overlapping model) та проста експліцитна модель (simple tiled model) [1].

2.2.1.1. Модель з перекриттями

Модель з перекриттями генерує зображення на основі вхідного зразка. Патерни автоматично виводяться зі зразка і являють собою невеликі унікальні зображення розміром n × n (зазвичай n=3). Обмеження суміжностей визначаються шляхом порівняння граней цих патернів, що усуває необхідність додаткових налаштувань. Основною перевагою цієї моделі є простота і декларативність.

Хоча ця модель найбільше підходить для синтезу текстур, вона також може застосовуватися для генерації рівнів. Саме вона забезпечує декларативність алгоритму WFC, оскільки позбавляє необхідності в складній конфігурації для кожного окремого рівня.

2.2.1.2. Проста експліцитна модель

Проста експліцитна модель базується на обмеженнях суміжностей, заданих користувачем вручну, що дозволяє створювати складніші правила генерації. Патерни та відношення між ними задаються явно, причому кожен патерн відповідає певному тайлу.

Однак, ручне встановлення правил генерації є трудомістким і складним завданням, яке потрібно виконувати для кожного нового набору патернів. Навіть для досягнення найпростіших результатів може знадобитися кілька сотень обмежень, що робить цю модель непрактичною в чистому вигляді.

Для спрощення процесу встановлення обмежень зазвичай використовують систему симетрії [10] та систему маркування суміжностей. Система симетрії значно зменшує обсяг роботи, дозволяючи користувачеві не задавати окремі обмеження для кожної трансформації патерну. Система маркування суміжностей дозволяє призначати кожному ребру патерну певний маркер і налаштовувати матрицю суміжності маркерів. Це значно скорочує кількість необхідних дій для конфігурації.

Проста експліцитна модель може бути більш придатною для генерації рівнів.

2.3. Опис алгоритму

2.3.1. Визначення проблеми

Нехай K буде набором можливих тайлів. Набір K складається з усіх цілих чисел від 0 до $|K|$-1, кожен тайл представлений цілим числом.

Нехай I буде зображенням розміру $N \times M$. Зображення є відображенням:

$$I: Z^2 \rightarrow K, Z^2 = \{(x,y) \mid x \geq 0, x < N, y \geq 0, y < M\}.$$

де Z^2 є сіткою, а точки (x,y) - клітинками.

Нехай P - множина всіх патернів.

Нехай i та j будуть одиничними векторами в напрямках x та y відповідно.Тоді $D = \{i, j, -i, -j\}$, $d \in D$..

Матриця суміжностей A розмірів $|D| \times |P| \times |P|$ визначається як:

$$A[d, p1, p2] =$$

1, якщо $p1$ може розміщуватись поруч із $p2$ в напрямку $d, p1 \in P$, $p2 \in P$

0 інакше

Зображення I є узгодженим з A, якщо (2.1):

$$\forall x \in Z^2, \forall d \in D \ : \ A[d, I(x), I(x+d)] = 1, (x+d) \in Z^2$$

Основна мета алгоритму полягає в генерації зображення I, яка є узгодженим з A. Рівняння (2.1) діє як обмеження на I і називається обмеженням суміжності [4].

2.3.2. Загальний огляд алгоритму

WFC розпочинається з аналізу вхідного зображення S, виділення локальних патернів P, побудови матриці суміжностей A та ініціалізації порожнього вихідного зображення $R, R: Z^2 \rightarrow P$.

Зображення R генерується ітеративно. На кожному циклі певним чином обирається клітина для «спостереження» - призначення ній конкретного патерну.

Внаслідок спостереження зображення може стати неконсистентним із A. Щоби уникнути цього, під час генерації підтримується деякий каталог W, який визначає які призначення будуть констистентними.

Після кожного спостереження каталог W потребує оновлення. Цей крок називається «поширенням». Він полягає у видаленні із W призначень, що більше не є консистентними із A.

Цей процес був натхненний оригінальною концепцією колапсу хвильної функції в квантовій фізиці, де хвильова функція (математична функція, що представляє ступені свободи, які може набрати спостережуваний об'єкт) безпосередньо спостерігається і колапсується в єдиний стан. Звідси й назва алгоритму. Хвильова функція в цьому випадку відповідає каталогу W.

Псевдокод алгоритму WFC:

def Run():

ExtractPatternsFromSample() - Виділення патернів із семлу

BuildAdjacencyMatrix() - Побудова матриці суміжностей

Loop until finished: - Генерація зображення

Observe() - Спостереження

Propagate() - Поширення

OutputObservations() - Інтерпретація результатів

2.3.3. Витягування патернів зі зразку

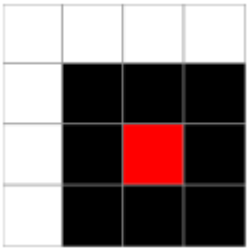

Рисунок 5 Зразок Red Maze розміром 4 × 4. Зауважте, цей зразок є періодичним [7].

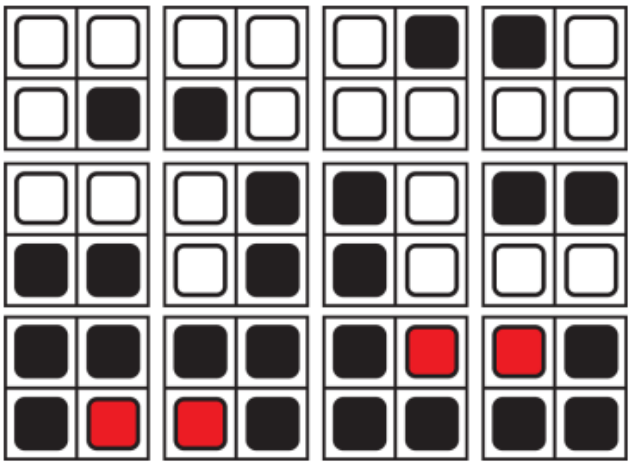

Рисунок 6 Патерни 2x2, отримані зі зразка Red Maze, з відображенням і обертанням [7].

Важливим етапом в алгоритмі WFC є витягування патернів із зразка, який задає стилістичні та структурні основи для майбутніх рівнів. Цей процес також збільшує кількість можливих конфігурацій патернів, застосовуючи різні трансформації.

Для цього із вхідного зразка почергово вибираються всі можливі фрагменти F фіксованого розміру вікна n x n.

Для кожного фрагменту із F застосовуються трансформації, такі як обертання (90°, 180°, 270°) та відображення (горизонтальне та вертикальне), що збільшує кількість можливих патернів без потреби збільшення первинної вибірки. Це дозволяє отримати складніші результати із простіших зразків, що забезпечує більшу гнучкість і різноманітність у процесі генерації.

Позначимо множину трансформацій як T, |T|=8 (4 обертання × 2 відображення).

Тоді загальна множина P витягнутих патернів: $|P| <= |F| \times |T|, F \subseteq P$.

2.3.4. Побудова матриці суміжностей

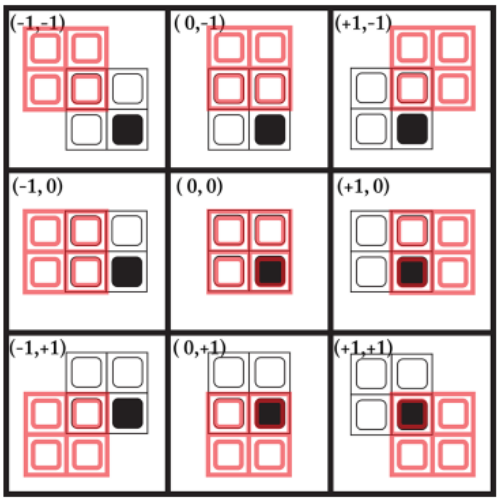

Рисунок 7 Дев'ять способів суміжності двох патернів 2 × 2 у зразку Red Maze [7].

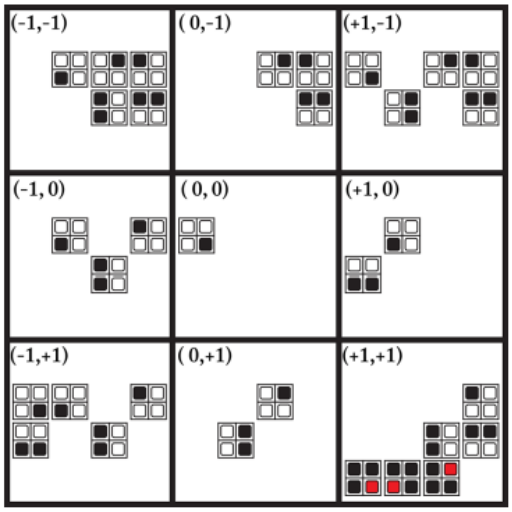

Рисунок 8 Фрагмент матриці суміжностей для першого патерну зі зразка Red Maze, який показує, які з інших патернів є суміжними з ним на певному зсуві. Матриця суміжностей зберігає таку інформацію для всіх патернів [7].

Функція BuildPropagator() будує матрицю суміжностей.

Для версії з перекриттям матриця суміжностей містить попередньо обчислені відповіді на те, чи збігаються краї двох патернів, коли вони розміщені поруч один із одним. Для простої експлісиної моделі матрицю можна створити безпосередньо з відношень, визначених дизайнером. Нас цікавить саме побудова матриці для моделі із перекриттями.

Варто зазначити, що в оригінальній імплементації матриця суміжностей називається пропагатором. Ми називаємо її матрицею суміжностей, щоб уникнути плутанини з алгоритмами пропагаторами обмежень, такими як AC3 (який є частиною WFC).

Нехай E_d - множина країв всіх патернів у напрямку d. Край $e \in E_d$ це масив тайлів розміром $N \times M$ або $M \times N$, який дорівнює перетину області патерну $N \times N$ із такою ж областю зміщеною на вектор d. Функція f задає відображення із патерну та напряму в його край:

$$f: P, D \to E_d$$

Тоді матриця суміжностей будується як:

$$A[d, p1, p2] =$$

$$1, f(p1, d) = f(p2, -d)$$

0 інакше

2.3.5. Генерація зображення

2.3.5.1. Задача виконання обмежень

Основний процес генерації в WFC - це класична задача виконання обмежень (ЗВО, англ. - CSP).

Формально ЗВО визначається трійкою (Xc, Dc, Cc), де Xc — множина змінних, Dc — область значень змінних і Cc — множина обмежень. Кожне обмеження, своєю чергою, є парою (tc, Rc), де tc — кортеж зі змінних та Rc — n−місне відношення Dc. Оцінка змінної — це функція, що позначає призначенням змінним значень, Vc : Xc → Dc. Оцінка Vc задовільне обмеження ((xl, ..., xn), Rc) якщо (Vc(xl), ..., Vc(xn)) ∈ Rc. Розв'язок — це оцінка, що задовільняє всім обмеженням [21].

Алгоритм для розв’язання ЗВО поступово призначає патерни (означує змінні), перевіряє суміжності (обмеження) та намагається знайти повне призначення змінних, тобто розв’язок. Для цього відстежуються залишкові домени для неозначених змінних. При цьому зазвичай також використовується деякий алгоритм пошуку, частіше всього це пошук з вертанням [24] (англ. backtracking) або локальний пошук, для ефективного вирішення конфліктів. Конфлікт означає, що оцінка Vc більше не задовольняє обмеженням.

В нашому випадку:

- Означенні змінних відповідає крок спостереження (observation)
- Перевірці обмежень - крок поширення (propagation)
- $Xc \equiv Z^2$ (змінні - це місця на сітці патернів)
- $Dc \equiv P$ (значення змінних - це патерни)

- $Cc \equiv A$ (обмеження визначаються матрицею суміжностей A)
- $Vc = R$ (Зображення є оцінкою змінних - призначенням патернів кожному місцю)
- Каталог W відстежує залишкові домени для неозначених змінних (в оригінальному коді ця структура називається wave)
- Оригінальний WFC не реалізує ніяких алгоритмів пошуку, а замість цього повнісітю перезапускається, якщо досягається конфлікт.

2.3.5.2. Процес спостереження (Observe)

Процес спостереження є ключовим в алгоритмі WFC. Саме від нього залежить якісь та можливість знаходження розв'язку. І саме його ми здатні контролювати для отримання потрібних нам результатів.

Спочатку кожна клітина в R не має патерну, тобто $\forall x : R(x) = -1$.

На кожному циклі алгоритму певним чином обирається клітина x для спостереження - призначення ній конкретного патерну p:

$$(x, p) : R(x) \leftarrow p, x \in Z^2, p \in P.$$

Генерація відбувається доти, доки кожна клітина не буде призначена патерну. Коли кожна клітина містить призначений патер зображення R є повним. Повне зображення є узгодженим з A, якщо воно задовольняє обмеженню суміжності в рівнянні (2.1). Неповне зображення є узгодженим, якщо його можна завершити так, щоб воно задовольняло обмеження суміжності.

За кожним спостереженням слідує процес поширення.

```
def Observe():
    SelectCell()
    If there is a contradiction:
        Return contradiction
```

If all cells are assigned, processing is completed:

Return success

Else:

SelectPattern()

Update wave with the selected pattern for the selected cell.

2.3.5.3. Процес поширення (Propagate)

Кожного разу, коли призначається патерн, є ризик, що це призначення може зробити зображення R неузгодженим. Цей ризик можна було б усунути, якби ми могли створити каталог можливих призначень для додавання до R.

Каталог W зберігає список патернів прийнятних для призначення у кожній точці. Він гарантує, що для кожного місця знайдеться такий патерн в каталозі його сусідів, що всі обмеження будуть виконані:

$$W(x,p) = 1 \Rightarrow \forall d\ \exists b : (W(x+d,b) = 1\ \wedge\ A[d,p,b] = 1)$$

$$x,(x+d) \in Z^2;\ p,b \in P$$

Крок поширення полягає в оновленні каталогу W. Якщо в результаті спостереження місцю був назначений патерн, то воно потребує оновлення всіх сусідів, оскільки вони пов’язані обмеженням суміжності. Усі значення сусідів що не задовольняють цих обмежень видаляються із каталогу, а сам сусід додається в чергу пропагації. З точки зору графіки це просцес схожий на алгоритм заливки (flood fill).

Інколи ми можемо натикнутися на конфлікти, оскільки каталог W не є ідеальним каталогом, а обчислення ідеального каталогу є NP-складним завданням [4].

Оригінальний WFC реалізує крок поширення як алгоритм узгодження дуг AC3.

---

defn Propagate():

For each cell x in the queue for update:

For each neighbor n:

For each pattern p where $W(n,p) = 1$:

Check adjacency constraints with x

If p no longer matches:

Set $W(n,p) = 0$

Add n to the queue for update

2.3.5.4. Евристики

Впродовж генерації ми хочемо обирати такі клітинки та патерни для спостереження, щоби збільшити шанси алгоритму знайти роз'вязок. Для цього використовуються певні евристики, які можуть змінюватися в залежності від потреб.

2.3.5.4.1. Евристика вибору місця

Оригінальною евристикою для вибору місця присвоєння є евристика мінімальної ентропії.

Нехай q - функція розподілу частот патернів в зразку, $q: P \to \Re$. А q_W - функція що визначає розподіл часто патернів для кожної клітинки в теперішній момент генерації, $q_W : P, Z^2 \to \Re$.

Тоді ентропія $E, E: Z^2 \to \Re$ обраховується як:

$$E(x) = log(\sum_{p \epsilon P} q_w(p)) - \sum_{p \epsilon P} q_w(p) log(q_w(p)) \; / \; \sum_{p \epsilon P} q_w(p)$$

Стратегія вибору місця з найнижчою ненульовою ентропією може здатися незрозумілою спочатку. Оскільки ми хочемо оптимізувати наші шанси на

отримання повного призначення, ми повинні робити кожен вибір так, щоб максимізувати кількість потенційно можливих повних призначень.

Нехай $Pr(x)$ - домен непризначених змінних у каталозі W для клітини x:

$$Pr: Z^2 \to 2^P, Pr(x) = \{a \mid W(x,a) = 1\}$$

Якщо припустити, що вибори змінних залишились є незалежними, то кількість залишкових повних призначень можна оцінити як:

$$r \approx \sum_{x \in Z^2} |Pr(x)|$$

Вибір змінної з найменшим доменом максимізуватиме цей добуток. А клітинка з найнижчою ентропією якраз і відповідатиме змінній із найменшим доменом.

Евристика вибору найбільш обмеженої змінної або, інакше кажучи, змінної з мінімальною кількістю залишкових значень (англ. minimum remaining values - MRV) добре відома у задачах виконання обмежень.

У фізиці, зокрема у статистичній механіці, така евристика є одним з проявів принципу максимальної ентропії.

2.3.5.4.2. Евристика вибору патерну

Для вибраного місця може існувати більше одного можливого патерну (інакше ентропія була б нульовою), тому потрібно якимось чином обрати один із них.

В оригінальному алгоритмі евристика вибору патерну є зважено випадковою, відповідно $q: P \to \Re$. Це допомагає наблизитись до вторинної мети автора: щоб патерни у вихідному зображенні мали подібний розподіл до вхідного.

2.3.6. Інтерпретація результатів

Коли в системі більше немає ентропії (тобто всі змінні мають лише одне можливе значення), ми можемо вивести остаточне згенероване зображення. Крім того, оскільки під час генерації кожна клітинка має кілька потенційних значень, ми

можемо виводити частково завершене зображення після кожного циклу спостереження та поширення. Це дозволяє створити привабливі візуалізації.

---

def OutputObservations():

    For each cell x:

        Set Rt(x) to the average color of values in Pr(x)

    Return Rt(x)

2.4. Переваги алгоритму

Алгоритм Wave Function Collapse (WFC) пропонує значні переваги для генерації ігрових рівнів порівняно з іншими методами.

Декларативність

Користувач здатен задавати параметри генерації за допомогою зображень потрібного рівня.

Універсальність

Алгоритм здатен генерувати велике різноманіття ігрових рівнів і обмежується лише фантазією користувача.

Зв'язність і структурованість

WFC гарантує, що всі елементи генерованого рівня мають логічні взаємозв'язки, створюючи координовані та змістовні ігрові простори.

Автоматичне впровадження різноманітності

На відміну від багатьох традиційних методів, які потребують чисельних налаштувань для створення різноманітності, WFC може автоматично генерувати багато унікальних рівнів з невеликої кількості вхідних патернів.

Гнучкість та контрольованість

Навідміну від інших алгоритмів, таких як Perlin noise чи клітинні автомати, розробники можуть впливати на результати генерації, забезпечуючи контроль над стилем і функціональністю рівнів.

2.5. Недоліки алгоритму

Слабка глобальна подібність

Оригінальний алгоритм слабо задовільняє умову глобальної подібності. На кожному кроці для вибору патерну керується імовірнісним розподілом, витягнутим із вхідного зображення. Через це він схильний змішувати різні глобальні властивості, внаслідок чого вихідне зображення може виглядати однорідним та випадковим в загальному. Глобальна подібність є дуже важливою, оскільки вона забезпечує збалансованість та грабельність рівнів.

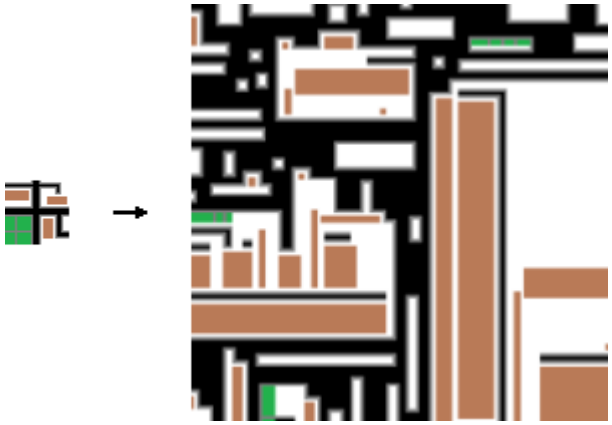

Рисунок 9 Результат роботи оригінального алгоритму. Зображення не є глобально подібним

Низька масштабованість

Оскільки за своєю природою WFC це пошук глобального присвоєння у задачі про задоволення обмежень, то алгоритм є відносно повільним. Крім того він схильний натрапляти на конфлікти, які часто вимагають повторного розгляду та зміни раніше прийнятих рішень, що може значно збільшити час генерації або завести алгоритм в глухий кут.

Ця проблема стає особливо актуальною для великих зображень та при роботі зі складними вхідними даними, де потрібно збалансувати багато обмежень одночасно. Це робить використання WFC в чистому вигляді для генерації ігрових рівнів непрактичним. Тому алгоритм вимагає оптимізацій та адаптацій для підвищення продуктивності.

Прив’язка до сітки

В алгоритмі WFC всі елементи повинні бути вирівняні відповідно до фіксованої геометричної сітки, яка визначає, де і як вони можуть розміщуватися. Через це алгоритм непридатний для генерації нерегулярних форм і тому підходить лише для використання в стилізованих іграх, де таке рішення не призводить до візуальної одноманітності та штучності. Крім того, це може накладати додаткові обмеження на дизайнерів та розробників та геймплей самої гри.

2.6. Порівняння із іншими алгоритмами

Підтримка обмежень дозволяє легко направляти алгоритм та комбінувати його з іншими генераторами.

2.6.1. Perlin Noise

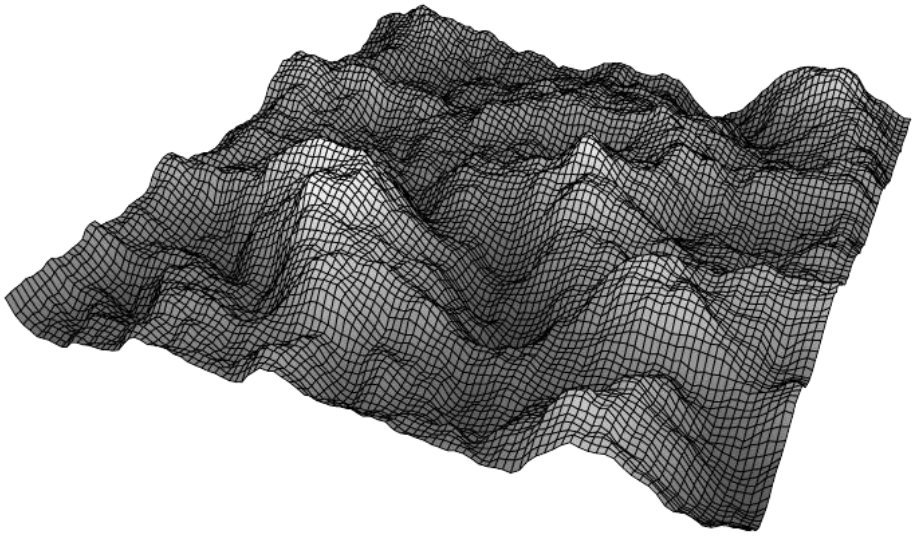

Рисунок 10 Рель'єф згенерований перлинним шумом [25].

Перлинний шум — це техніка, розроблена Кеном Перліном, яка використовується для створення природноподібних текстур та ландшафтів у візуальному дизайні та ігровій графіці. Метод заснований на випадковому градієнтному шумі, що генерує візуально плавні, неперервні текстури.

На відміну від WFC, який використовує строгі локальні правила для вироблення структурованих шаблонів, перлинний шум створює органічні форми, що робить його придатним для генерації природних ландшафтів, проте не для складних структурованих ігрових рівнів.

2.6.2. Cellular Automata

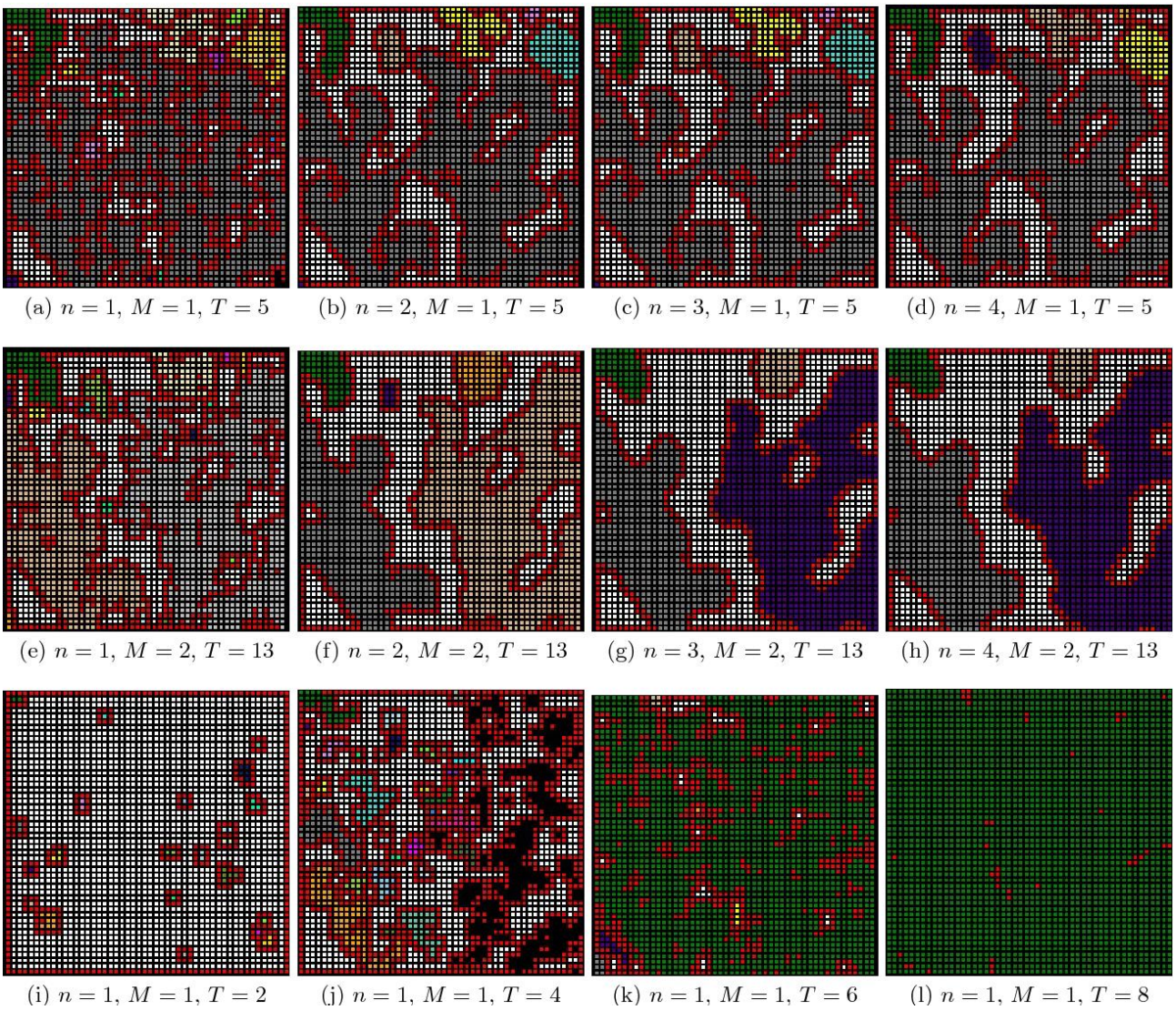

Рисунок 11 Генерація ігрового рівня за допомгою клітинних автоматів [26].

Клітинні автомати — це математична модель, що використовує набір простих правил для визначення стану клітини в сітці на основі стану її сусідів. Цей метод часто використовується для моделювання комплексних систем з простих компонентів, таких як пожежі, розповсюдження хвороб або зростання структур.

Подібно до перлинного шуму, клітинні автомати схильні створювати органічні структури. Тому вони більше придатні для генерації ландшафту ігрового рівня та різноманітних печер, аніж більш складних штучних структур. Тому цим їх використання зазвичай і обмежується.

За словами автора WFC, d-вимірний WFC здатен зафіксувати поведінку будь-яких d-1 вимірних клітинних автоматів [1].

2.6.3. Markov Chain

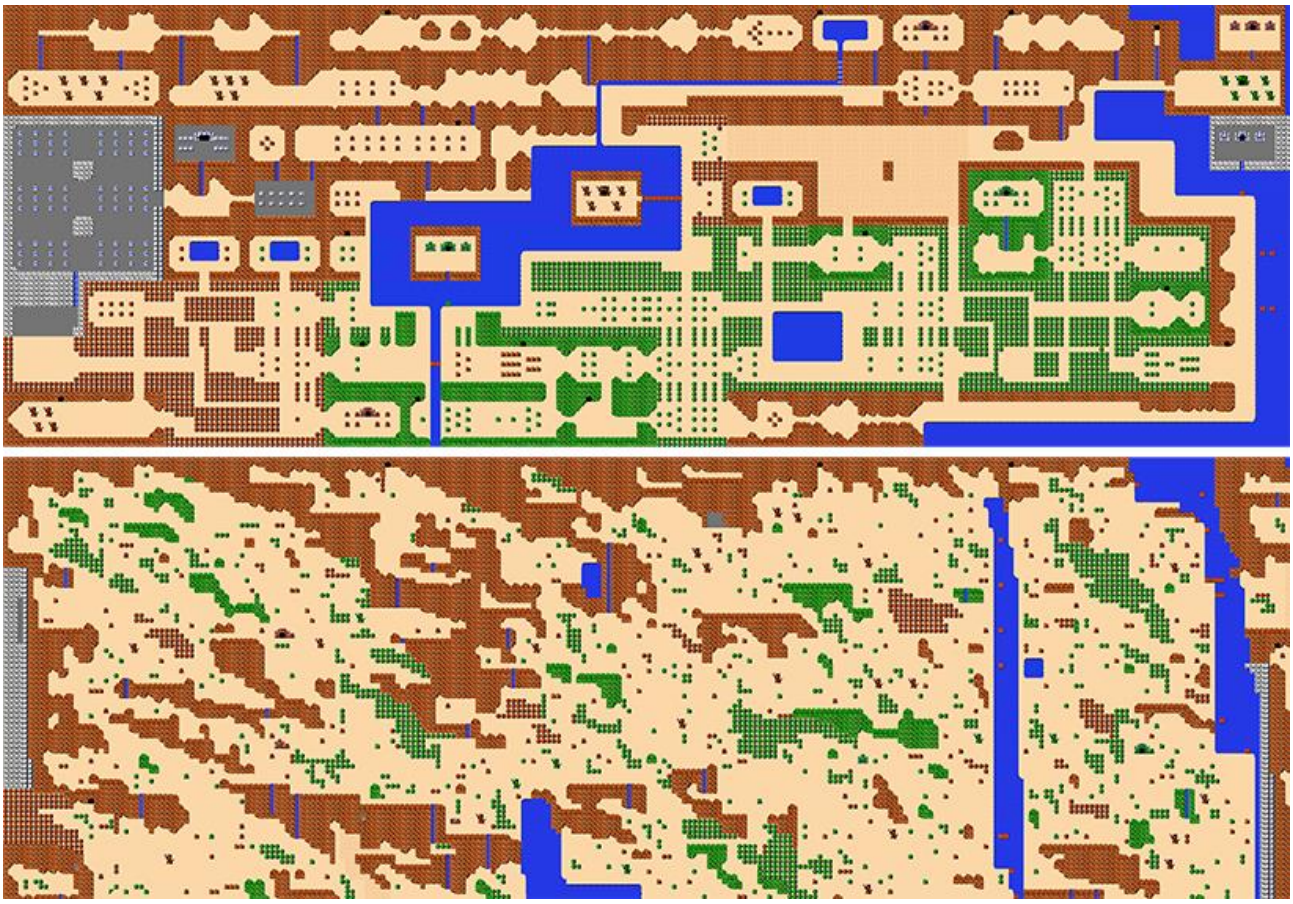

Рисунок 12 Використання Markov Chain для генерації ігрового рівня [27]

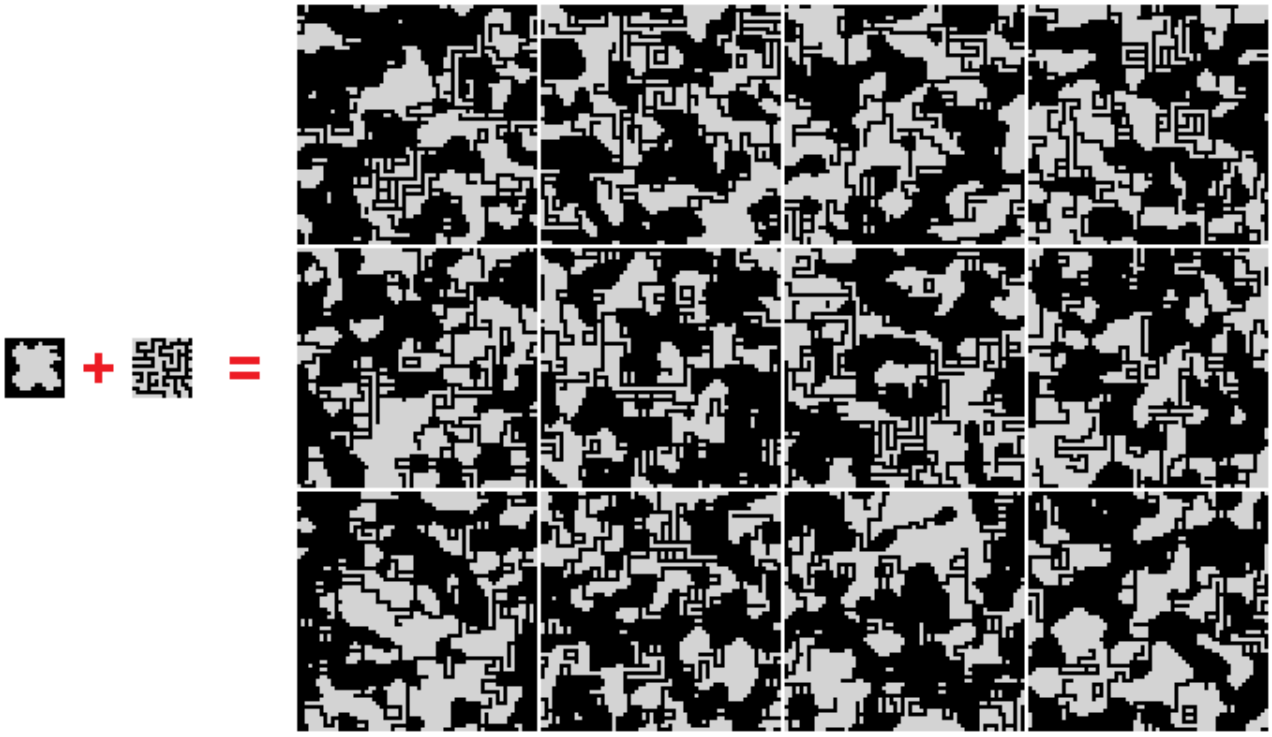

Рисунок 13 Приклад згенерованих зображень за допомогою двох зразків [28].

Markov Chain — це статистичний метод, який використовує ймовірності переходів між станами на основі існуючих послідовностей. Кожен наступний стан системи залежить лише від поточного стану, а не від послідовності станів, що його передували.

Загалом алгоритм Markov Chain схожий за принципом генерації до WFC. Він простий в налаштуванні і може керуватися одним лише вхідним зразком. Основною його перевагою є здатність точно відтворювати розподіл патернів, який був присутній у вхідних даних - умова, яку оригільний алгоритм WFC задовольняє слабо. Також він має підтримку обмежень.

Однак, Markov Chain може створювати помітні дефекти і не забезпечує високий рівень структурної цілісності. Через це він малопридатний для генерації ігрових рівнів, через їхню низьку якість та ігровий потенціал.

2.6.4. L-системи (Lindenmayer Systems)

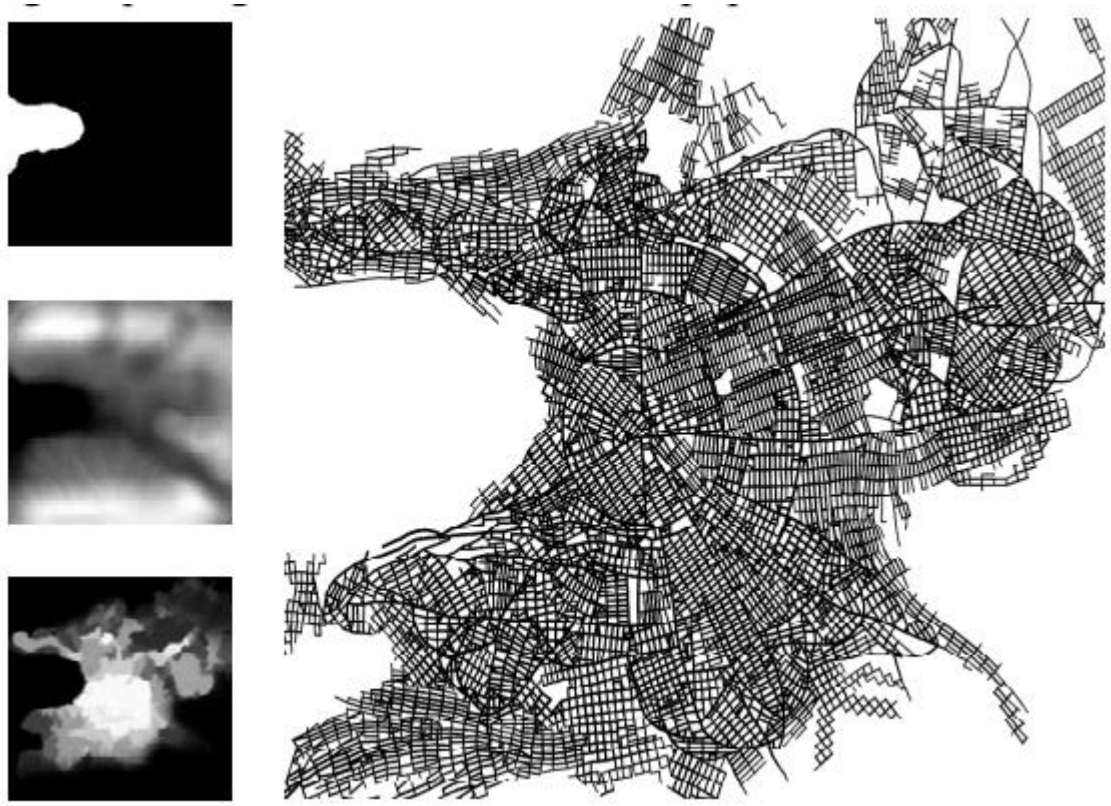

Рисунок 14 Місто згенероване за допомогою L-систем [29].

L-системи — це форма граматики переписування, яка використовується для моделювання росту рослин та фрактальних структур. Вони дозволяють створювати дуже складні зображення з досить простих правил, а також враховувати глобальні цілі та локальні обмеження

L-системи ідеально підходять для генерації фрактальних або біологічних структур. Вони зокрема були використані для генерації реалістичних міст із врахуванням законів урбаністики. L-системи здатні ефективно створювати вражаючі та реалістичні результати, що не прив’язані до геометричної сітки, навідміну від WFC.

Їх недліком є те, що вони вимагають ручного створення складних граматик, а тому потребують досвідчені спеціалістів для отримання якісних та передбачуваних результатів.

2.6.5. Texture Synthesis

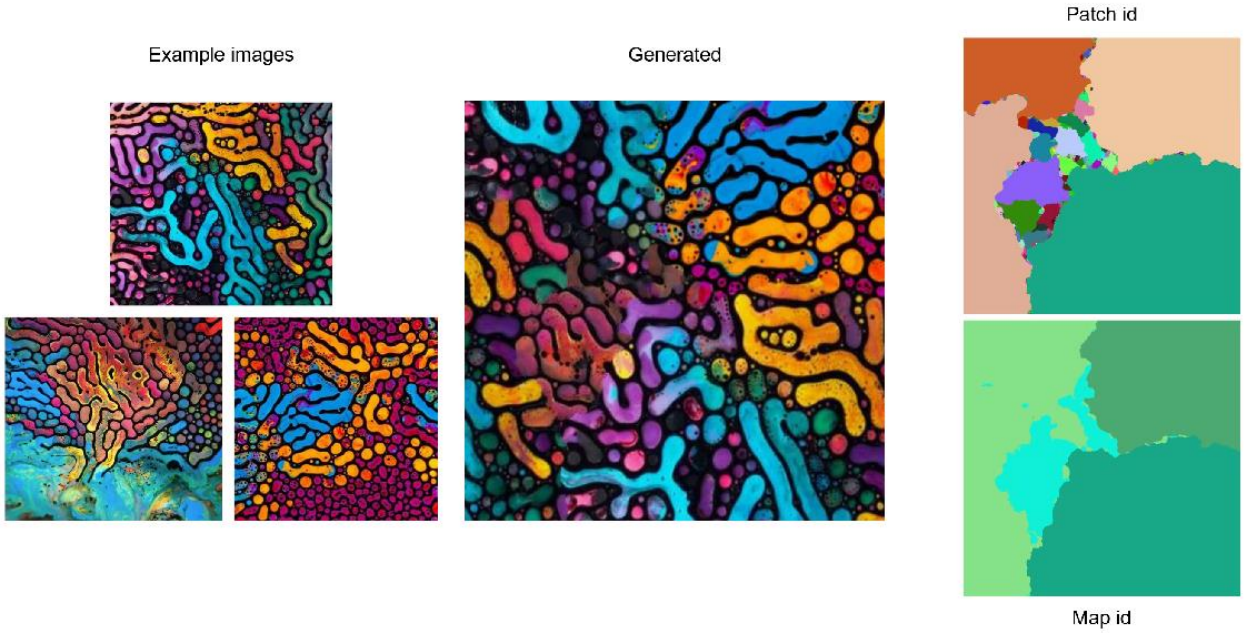

Рисунок 15 Приклад синтезу текстур [30].

Texture synthesis (синтез текстур) використовується для створення нових текстур із вхідного зразка, розширюючи або змінюючи її без видимих швів або повторень. Метод зазвичай використовує алгоритми, які аналізують імовірнісні розподіли характеристик вхідної текстури та намагаються отримати максимальну схожість результатів. Деякі алгоритми синтезу текстур можуть бути адаптовані для роботи із неграфічними даними, що не є текстурами.

Алгоритми синтезу текстур підтримують обмеженняі і можуть застосовуватись навіть для переносу стилю зображення. Не важко уявити як такий інструмент можна було би використати для генерації ігрових рівнів.

Рисунок 16 Приклад перенесення стилю [30].

Ключовою особливістю алгоритмів синтезу тексур є те що вони ефективно створюють результати максимально подібні до вхідного зразка, забезпечуючи візуальну неперервність та відсутність повторень.

Алгоритм синтезу текстур може бути значно швидший ніж WFC. Однак, він не здатен забезпечити структурну цілісність та дотримання локальних обмежень вхідного зразка, а також має проблеми з довгими кореляціями (наприклад, йому важко синтезувати текстури цегляної стіни з правильно вирівняними цеглами).

Загалом, не зважаючи на свою потужність та інтуїтивність, алгоритми синтезу текстур вкрай рідко застосовуються для генерації ігрових рівнів. Проте саме ними були натхненні алгоритми Model Synthesis та WFC.

2.6.6. Model Synthesis

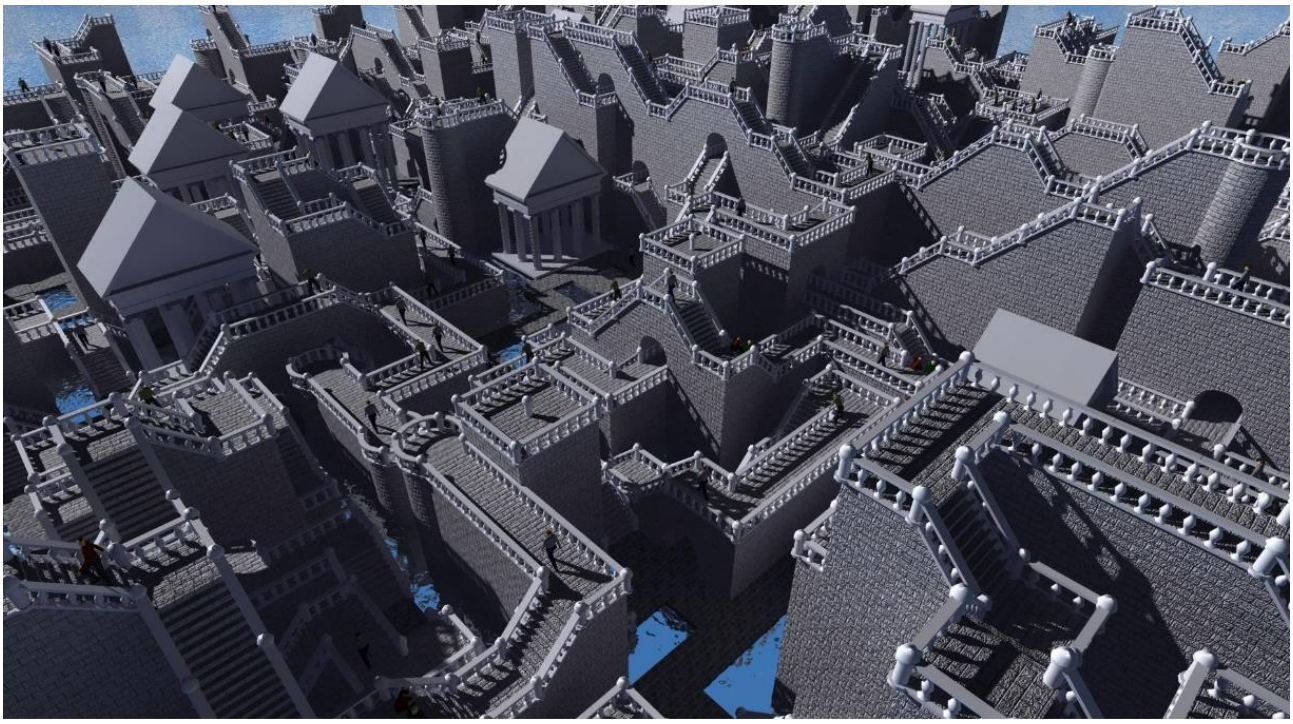

Рисунок 17 Model Synthesis [3]

Model Synthesis — це техніка генерації об'єктів та структур, розроблена Полом Меррелом у 2007 році. Вона базується на використанні великої вхідних моделей для створення нових композицій, які зберігають стилістичну та структурну цілісність оригінальних елементів. В його основі лежить вирішення задачі про виконання обмежень.

WFC, створена у 2016 році, фактично реімплементує алгоритм Model Synthesis. Основними відмінностями є те, що WFC використовує іншу евристику для вибірки клітин, а також більше зосереджений на генерації 2D зображень, а не на 3D моделей. Ключовим нововведенням WFC є ідея використання частин вхідного зображення для автоматичного визначення обмежень суміжності.Однак, WFC позбавлений можливості модифікації в блоках, що робить його менш гнучким для створення складних моделей [5].

2.6.7. Graph Grammars

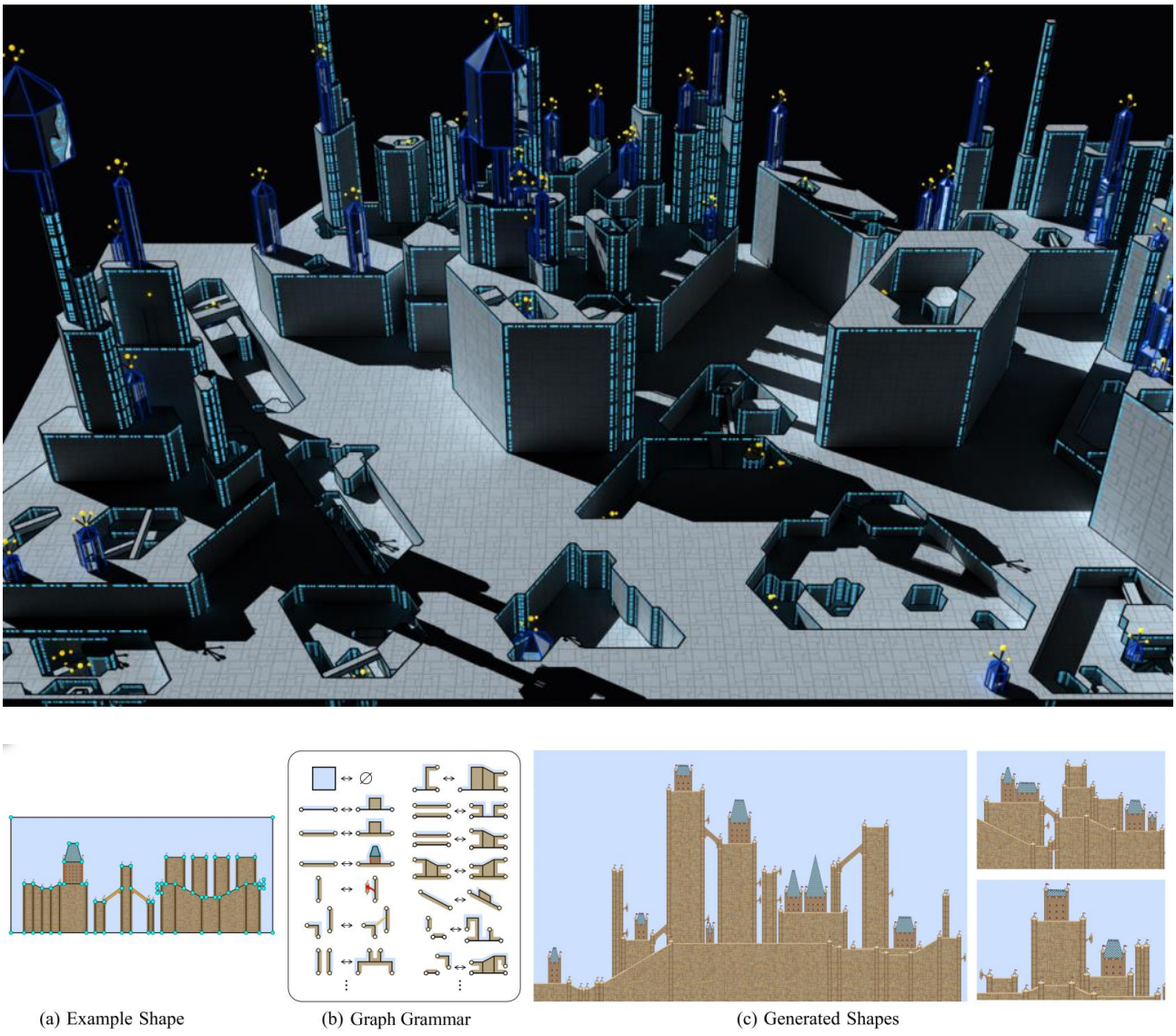

Рисунок 18 Graph grammars [18]

Graph grammars (графові граматики) - це найновіший алгоритм, розроблений також Полом Меррелом у 2023 році., який фактично є наступником алгоритму Model Synthesis. Graph grammars узагальнює ідею генерації структур на основі зразка. Однак, на відміну від Model Synthesis або WFC, він не використовує плитки та не обмежується сіткою. Натомість виходячи зі зразка будується граматика графа, яка створює локально подібні фігури.

Разом із тим, графові граматики також унаслідували основні недоліки цих алгоритмів - це слабка глобальна подібність та відносно низька ефективність генерації.

Генерація структур у графовій граматиці відбувається через застосування продукційних правил до графів. Ці правила визначають, як нові вершини або ребра додаються або змінюються у графі. Це дозволяє створювати моделювати складні системи, такі як соціальні мережі, архітектурні структури та інженерні системи.

На даний момент це імовірно найпотужніший універсальний алгоритм для генерації ігрових рівнів.

2.7. Приклади використання WFC в іграх

Використання алгоритму Wave Function Collapse у розробці ігор значно розширилося за останні роки, демонструючи його ефективність. Ось кілька прикладів, які висвітлюють це [7]:

1. Proc Skater 2016 - це гра, розроблена Джозефом Паркером, Раяном Джонсом та Оскаром Моранте у рамках ProcJam 2016. Вона стала першою грою, що використала WFC для створення скейт-парків. Користуючись вибіркою дизайнера, алгоритм зміг автоматично генерувати унікальні ігрові рівні, які відповідали стилістичним та функціональним вимогам.

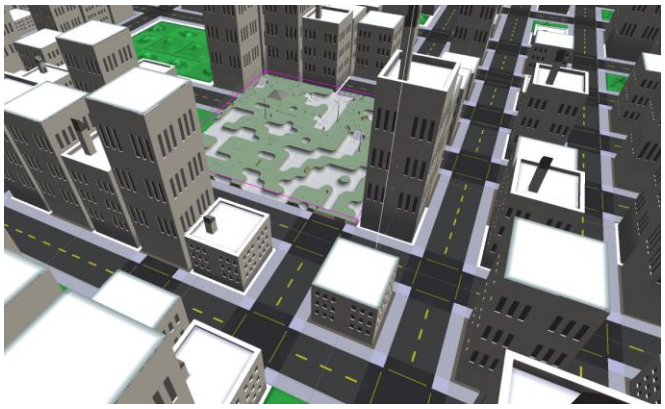

Рисунок 19 Рівень згенерований з допомогою WFC у Proc Skater 2016 [34].

2. Інструменти для Unity3D - Джозеф Паркер розробив набір інструментів для ігрового рушія Unity3D, який інтегрує концепції WFC для створення 3D об'єктів у грах. Цей набір інструментів став популярним серед інших розробників, оскільки він дозволяє швидко та ефективно генерувати складні об'єкти і структури.

3. Townscaper - Оскар Столберг експериментував з WFC, поєднуючи його з технікою "marching cubes" на нерегулярних сітках, для створення гри Townscaper, фокус якої лежить на будівництві міст. Столберг також активно ділиться своїми знаннями про генерацію міст, проводячи інтерв'ю та виступи, що допомагає розповсюдити ідеї WFC в ігровій індустрії.

Рисунок 20 Місто згенероване з використанням WFC у Townscaper [31].

4. Caves of Qud - Браян Баклев виступав на ігрових конференціях, обговорюючи, як Freehold Games використовує WFC для генерації рівнів у їх грі Caves of Qud. Він зазначав проблеми з перенавчанням, однорідністю та зв'язністю рівнів, а також інтеграцію WFC з іншими методами процедурної генерації.

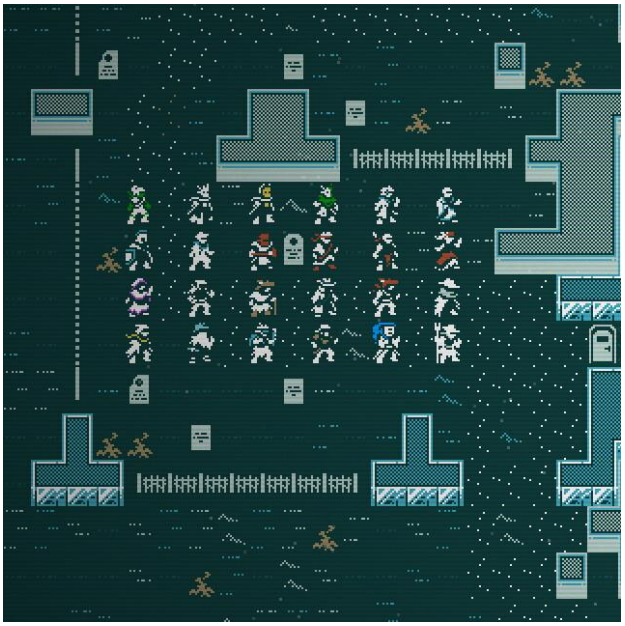

Рисунок 21 Рівень згенерований з допомогою WFC у Caves of Qud [32].

5. Maureens' Chaotic Dungeon - На Global Game Jam 2019 Енді Воллес створив інтерактивну гру, де гравці можуть впливати на структуру цію рівнів за допомогою WFC, змінюючи різні частини рівня за допомогою віртуальної зброї. Це додало новий рівень взаємодії між гравцем і генератором рівнів, розкриваючи нові можливості для геймплею.

Рисунок 22 Рівень у Maureens' Chaotic Dungeon, який здатен змінюватись під час гри за допомогою WFC [33].

Ці приклади ілюструють, як WFC відкриває нові можливості для ігрових дизайнерів, дозволяючи створювати більш багаті та змістовні ігрові середовища, а також підвищує ефективність процесів розробки.

2.8. Висновки

Теоретичний аналіз алгоритму Wave Function Collapse (WFC) розкриває його модульну структуру, що включає чотири ключові кроки: екстракцію патернів зі зразка, побудову матриці суміжностей, ітеративний процес спостереження та пропагації, а також інтерпретацію результатів. Оригінальний алгоритм використовує евристику мінімальної ентропії для вибору наступного кроку та орієнтується на розподіл патернів у вихідному зображенні.

Ця структура надає WFC гнучкість для вдосконалень та дозволяє інтегрувати його з іншими алгоритмами. Основні переваги включають простоту в управлінні, здатність зберігати локальні структурні елементи та автоматично вносити різноманітність у генерований контент.

Однак, серед недоліків WFC — обмежена глобальна подібність, прив’язка від геометричної сітки, а також низька масштабованість. Попри це, порівняльний аналіз із іншими методами показав, що WFC значно перевершує більшість аналогів у якості та простоті генерації контенту.

Завдяки цим особливостям, WFC знайшов широке застосування у різноманітних ігрових проектах, де його ефективність була успішно демонстрована. Наявність відкритих джерел та спільнот, що підтримують розвиток алгоритму, також сприяє його популярності та вдосконаленню.

Розділ 3. Розробка прототипу системи генерації ігрових рівнів на основі алгоритму колапсу хвильової функції

3.1. Теза

Метою цього розділу є розробити генератор ігрових рівнів який базуватиметься на алгоритмі WFC та задовольнятиме наступні критерії:

1. Локальна подібність, яка забезпечує функціональну взаємодію об’єктів (наприклад, двері, які ведуть до стін). Інакше рівень буде неграбельним.
2. Глобальна подібність, яка передбачає загальну схожість до вхідного зразка, що забезпечує збалансованість та передбачуваність ігрового рівня.
3. Універсальність - здатність алгоритму генерувати різноманітні рівні.
4. Декларативність - користувач повинен могти керувати алгоритмом за допомогою вхідного зразка.

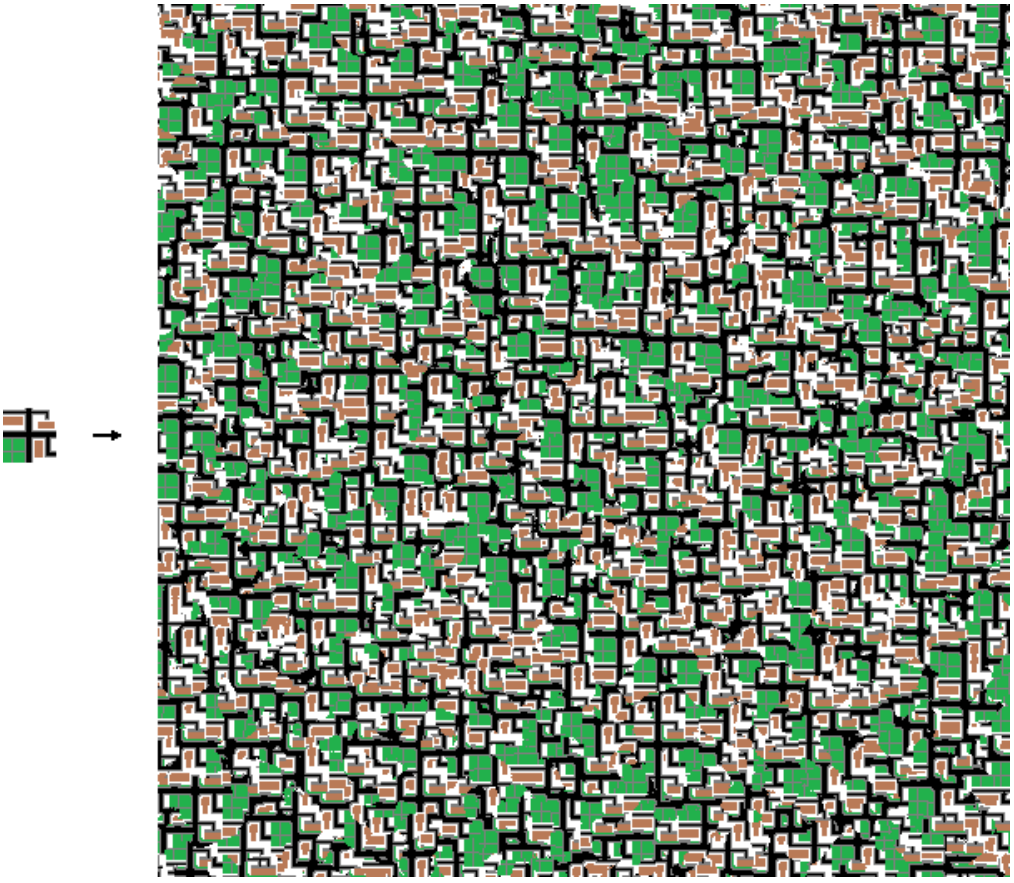

Рисунок 23 Глобальна подібність

3.2. Архітектура генератора

Ми дійшли висновку, що основною проблемою оригінального алгоритму є неможливість задовольнити умову глобальної подібності.

Алгоритм WFC в своїй основі не обробляє глобальні характеристики зображення. Він початково призначений для задоволення локальної подібності. Незважаючи на всі модифікації та додаткові обмеження, досягти консистентної та передбачуваної глобальної подібності за допомогою нього просто неможливо.

Тому логічним рішенням є використання іншого алгоритму для генерації зображення, що задовольняє умову глобальної подібності. А потім, на основі цього результату, застосувати алгоритм WFC для виправлення локальних недоліків і забезпечення умови локальної подібності.

Цей принцип поширений у задачах оптимізації: спочатку запускається швидкий, але менш точний алгоритм для отримання приблизного результату, а потім на його основі застосовується повільний, але точний алгоритм для коригування цього результату.

Архітектура пропонованого генератора:

1. Застосування наближеного алгоритму для швидкого синтезу зображення, яке задовольняє глобальну подібність.
2. Оцінка подібності згенерованого результату та детекція швів - місць які порушують вимогу локальної подібності.
3. Застосування точного алгоритму WFC для корекції зображення та досягнення локальної подібності.

Цей підхід дозволяє забезпечити досягнення як глобальної, так і локальної подібності. Крім того, ми вільні експериментувати з алгоритмами для першого кроку. Можливо навіть замінити алгоритм синтезу на вручну створений користувачем вхід, отримуючи таким чином алгоритм генерації зі змішаною ініціативою.

Основними проблемами в такій архітектурі є

1. Правильний вибір алгоритму синтезу, оскільки від цього напряму залежить кінцева якість результату.
2. Інтеграція алгоритму синтезу із алгоритмом WFC.

3.3. Алгоритм синтезу

Ми вирішили використати алгоритм із галузі синтезу текстур.

Нас зацікавили три алгоритми:

1. "Fast Texture Synthesis" [15]: Ця робота вперше запропонувала використання методу найближчих сусідів і описує процес синтезу в кількох розмірностях за допомогою побудови пірамід Гауса. Цей підхід дозволяє досягти ефективного та керованого синтезу зображень з урахуванням їхньої структури на різних частотах.
2. "Synthesizing Natural Textures" [16]: У цій роботі був вдосконалений попередній алгоритм шляхом використання генерації з використанням шматків. Крім того, демонструється метод синтезу зображень, що контролюється користувачем за допомогою вказаного зразка.
3. "Resynthesis" [17]: Цей алгоритм є найшвидшим серед усіх і узагальнює ідею генерації шматками.

Ми обрали за основу алгоритм ресинтезу Гаррісона, через його швидкість, гнучкість та універсальність.

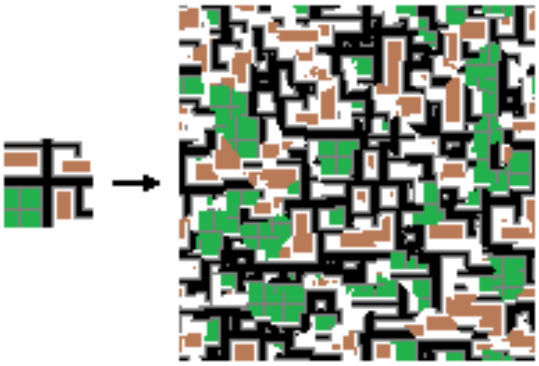

Рисунок 24 Приклад роботи алгоритму ресинтезу Гаррісона на нашому зразку

3.3.1. Трансформації вхідного зображення

Однієї з відмінностей WFC від алгоритму синтезу є те що перший генерує зображення використовуючи різні трансформації патернів, що були присутні на вході.

Останній же не враховує трансформацій, а тому має великий нахил до того яким саме чином повернуте вхідне зображення. А оскільки саме він визначає як виглядатиме вихідне зображення, це дуже сильно впливає на вихідний результат.

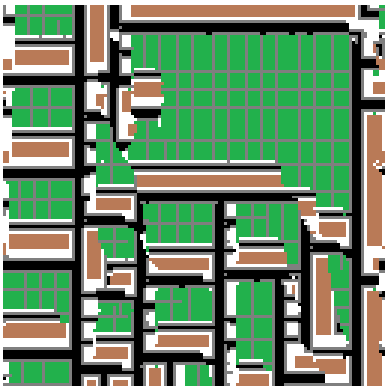

Рисунок 25 Результат роботи оригінального алгоритму

Для вирішення цієї проблеми ми модифікували оригінальний алгоритм Гаррісона таким чином, щоби він використовував різні трансформації частин зразку.

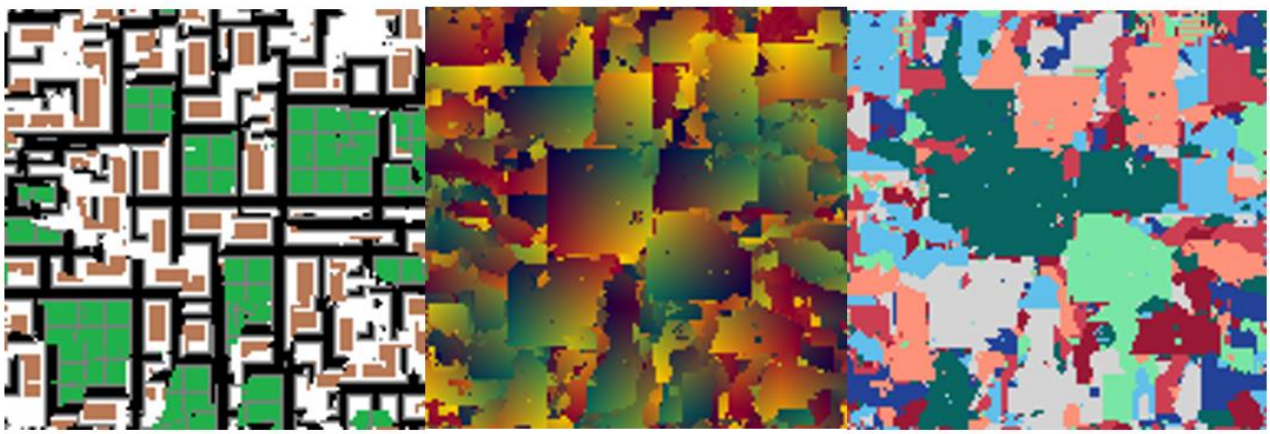

Рисунок 26 1. Результат роботи модифікованого алгоритму. 2. Карта координат патчів. 3. Карта трансформацій.

3.3.2. Евристики порядку генерації

Оригінальний алгоритм синтезу використовує випадкову евристику, що призводить до великої кількості швів. Тому ми вирішили замінити її на спіральну евристику з центру. По перше, на відміну від лінійної евристики, вона не має відхилень в певному напрямку. А по друге, вона дозволяє генерувати більш консистентні зображення і отримати виразніші структури.

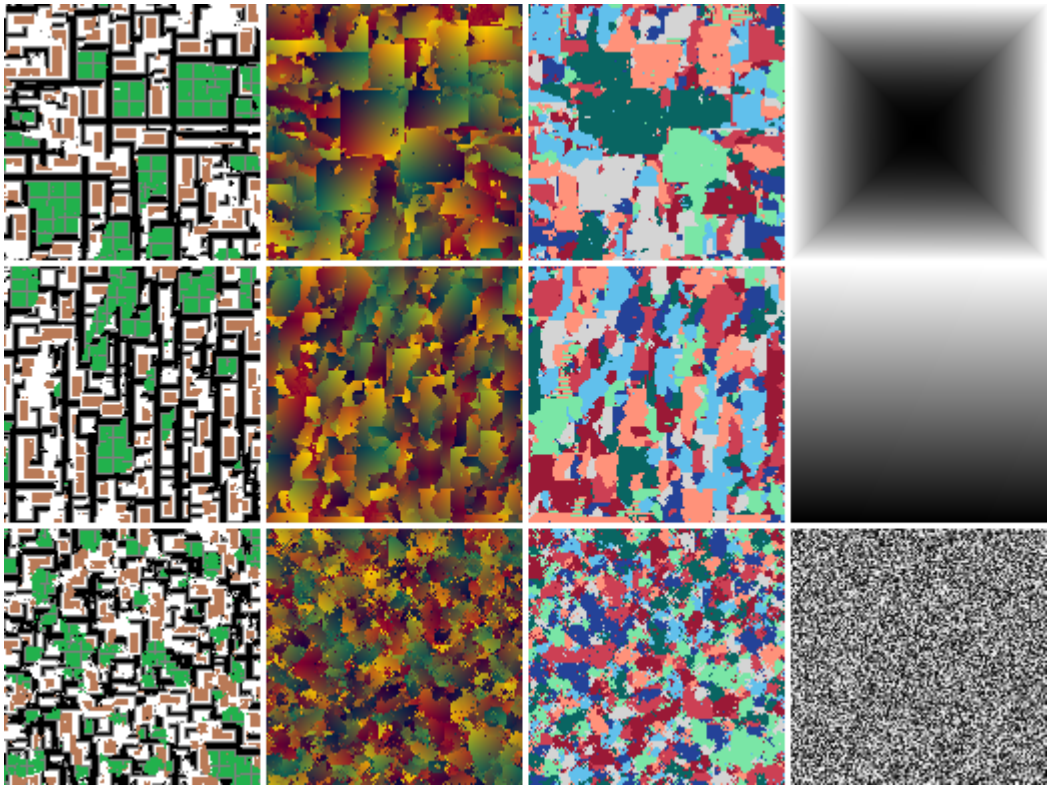

Рисунок 27 Результат синтезу із різними евристиками порядку. Із верху до низу: 1. Спіральна, 2. Лінійна, 3. Випадкова

3.4. Інтеграція алгоритму синтезу та WFC

Наступним кроком є використання синтезованих результатів для керування алгоритмом WFC. На кожному кроці алгоритму ми хочемо вибирати такі локації та патерни, які максимально наближають поточний результат до зображення, синтезованого на попередньому етапі. Для досягнення цієї мети застосовується певна оцінка подібності.

Використання оцінки подібності дозволяє алгоритму WFC ефективно визначати, де необхідно дотримуватися синтезованого зображення, а де потрібно зміни. Це дозволяє оптимізувати процес генерації та усунути структурні невідповідності.

3.4.1. Оцінка подібності згенерованого результату та детекція швів

Для перетворення зображення, заданого тайлами, у патерни ми вводимо спеціальну функцію, яка оцінює подібність кожного патерну до відповідного фрагменту зображення в кожній його точці. Цю функцію ми і називаємо "оцінкою подібності". Завдяки цій оцінці ми можемо виявляти "шви" або місця дефектів, де жоден із наявних патернів не забезпечує адекватного покриття. Саме такі місця потребують корекції з боку WFC.

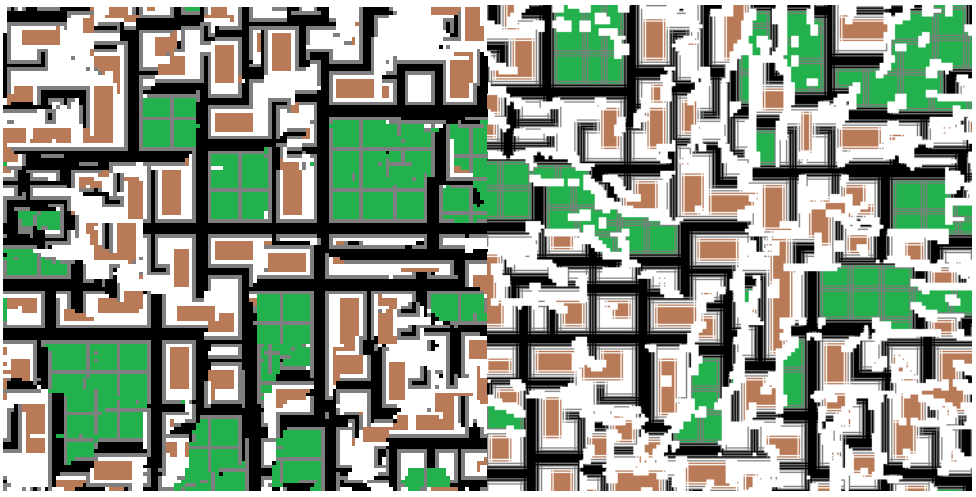

Рисунок 28 Візуалізація швів - місць, яким не відповідає жоден патерн.

Ми дослідили наступні способи оцінки подібності:

- Максимальна подібність - знаходимо лише найкращі патерни. Це значно економить обчислювальні ресурси та дає хороші результати.
- Розподіл подібностей - обчислюємо подібності для всіх патернів у кожній точці. Ресурсозатратний метод, який потребує подальших оптимізації,проте найкраще здатен справлятися із великими або численними швами.
- Метод найгіршого сусіда - тут ми виходимо із припущення, що патерн може бути лише настільки подібним, наскільки є подібними є його найкращі сусіди. Такий підхід важливий при використанні складних тайлсетів, де виконання умови суміжності двох сусідніх патернів не випливає із їх подібності.

- Пеня за неподібність - цей метод використовувався в поєднанні зі спеціальною зваженою метрикою та не показав гарних результатів.

Найкращим підходом виявилась оцінка максимальної подібності.

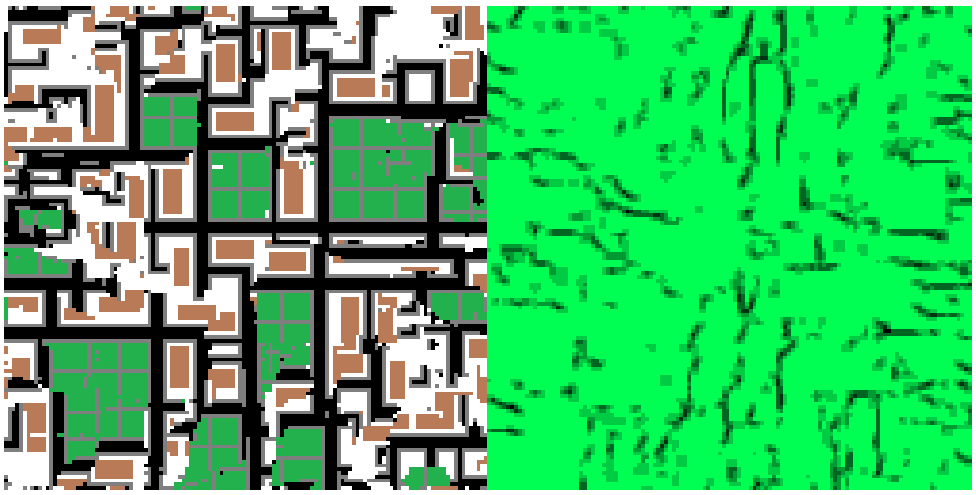

Рисунок 29 Візуалізація значень максимальної подібності.

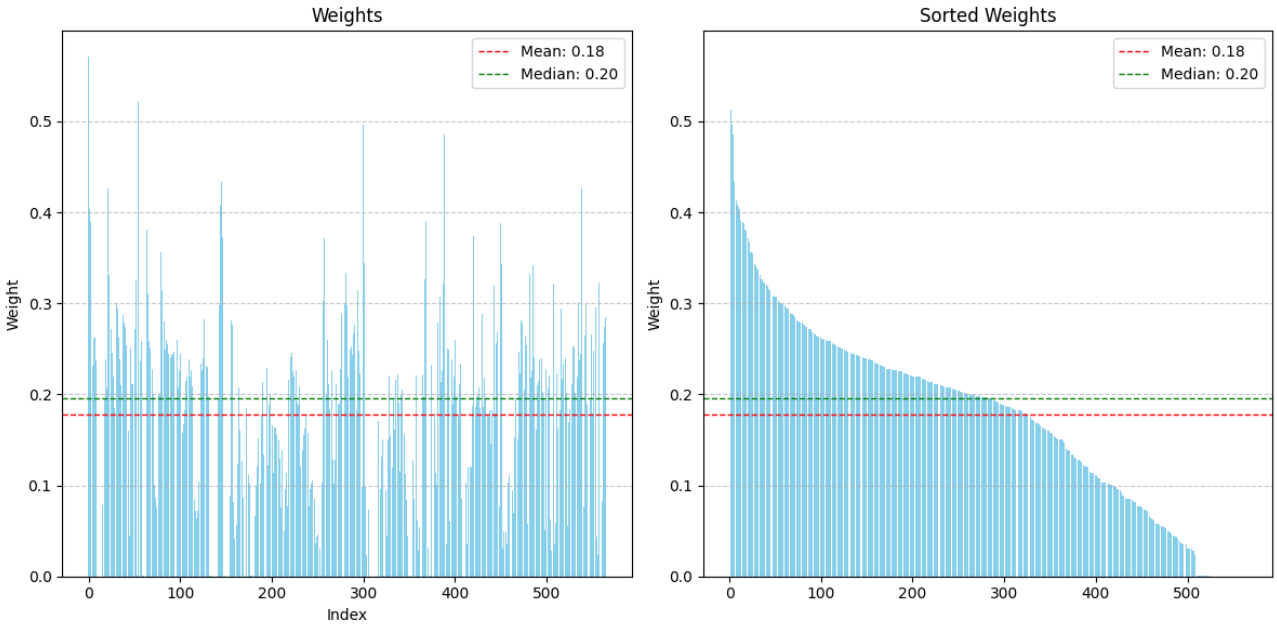

Рисунок 30 Середні значення подібностей патернів для кожної клітинки.

3.4.2. Евристики вибору значень та порядку генерації для WFC

Оригінальний алгоритм WFC використовує евристику мінімальної ентропії для визначення порядку обробки даних. Ця евристика фактично означає вибір найбільш очевидних значень. Існують дві причини для використання цієї евристики. По-перше, вона максимізує кількість можливих варіантів, що сприяє успішному вирішенню завдання. По-друге, ентропія має важливу властивість:

елементи для генерації обираються послідовно поруч із попередньо згенерованими, що мінімізує кількість конфліктів та підвищує ефективність бектрекінгу.

У нашому випадку завдання відрізняється. Ми шукаємо не будь-яке рішення, а таке, яке буде найбільш схожим на результат, отриманий на попередньому кроці. У нас вже розраховані подібності для кожного елементу, і ми повинні керуватися ними для визначення послідовності генерації.

Логічно, що елементи з найбільшими значеннями подібності повинні бути обрані першими, оскільки вони обмежуватимуть можливі значення для інших елементів. Елементи, що обираються пізніше, будуть мати меншу множину можливих значень, що призведе до збільшення похибок. Ці похибки накопичуватимуться, оскільки кожен елемент з похибкою обмежуватиме значення сусідніх елементів і розповсюджуватиме цю похибку. Тому елементи на місці швів слід обирати останніми, оскільки саме там ми можемо допистуме найбільше відхилення від прикладу.

При цьому, для елементів з однаковими значеннями подібності, оптимальним залишається вибір найбільш очевидного елементу. Іншими словами, евристика мінімальної ентропії.

Проблема полягає в тому, як знайти баланс між вибором найбільш подібного та найбільш очевидного елементів. Для досягнення цього балансу, ми провели експериментальні дослідження кількох різних евристик, результати яких наведені нижче.

- Мінімальна нормалізована ентропія - це оригінальна евристика. Не працює в нашому випадку, оскільки в результаті нормалізації ми повністю ігноруємо інформацію про подібності.
- Мінімальна ненормалізована ентропія - це узагальнення ентропії як математичної функції дозволяє одночасно враховувати як величину значення подібності патерну, так і ступінь його очевидності.

- Середня максимальна подібність - одна з найкращих евристик, використовує максимальні значення подібностей та розмиття Гауса. Визначає елементи, що найбільш віддалені від швів, і тому мають найбільшу імовірність залишитись незмінними.
- Середня максимальна подібність та мінімальна ентропія - ця евристика намагається поєднати дві відповідні евристики за допомогою параметра змішування.

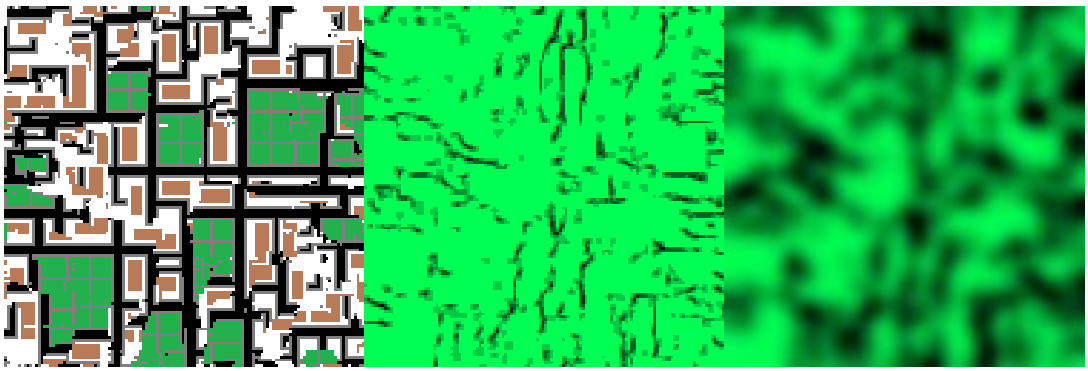

Рисунок 31 Візуалізація евристики середньої максимальнаї подібності (справа). Яскраві значення відповідають клітинкам, що будуть згенеровані першими.

3.4.3. Способи вирішення конфліктів в алгоритмі WFC

Алгоритм WFC є методом оптимізації. При роботі з складними даними ми неодмінно стикаємося з конфліктами, які впливають на швидкість алгоритму та його здатність генерувати складні результати.

3.4.3.1. Бектрекінг

Традиційним методом вирішення конфліктів для задач задоволення обмежень, зокрема для алгоритму WFC, є бектрекінг.

Бектрекінг припускає, що зерно конфлікту виникло через останні присвоєння. Проте це не завжди так. При генерації зображень, а тобто генерації у двовимірному просторі, конфлікти найчастіше виникають, коли зливаються два різні регіони зображення. При цьому часто один з них може бути згенерований набагато раніше за інший. В такому випадку, бектрекінг відмінить майже більшу частину присвоєнь, хоча більшість з них навіть не матимуть відношення до цього конфлікту.

Ситуація ускладнюється, коли одночасно зливаються кілька різних пар регіонів. Це призводить до осцилювання бектрекінгу між вирішенням конфліктів у цих місцях. Вирішення конфлікту в одному місці приводить до відміни в іншому, що знову вимагає вирішення конфліктів у першому місці. В результаті це зациклює процес бектрекінгу.

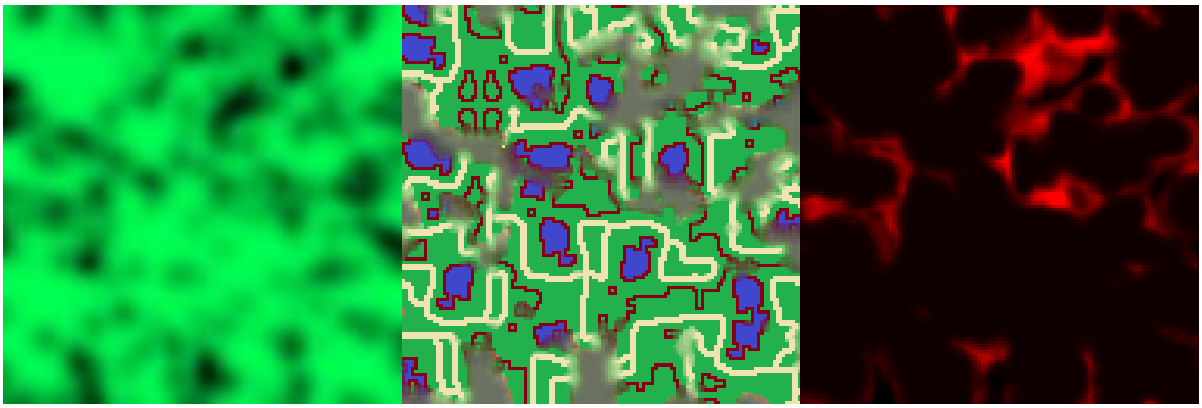

Рисунок 32 Приклад зациклення бектрекінгу через одночасне злиття багатьох регіонів. Місця що залишились так і не будуть згенеровані. 1. Порядок генерації. 2. Генерація в процесі. 3. Карта повторних генерацій, спричинених бектрекінгом.

Уникнути злиття регіонів неможливо, оскільки доступні евристики вибору, які не страждають від цієї проблеми, є примітивними, такими як сканлайн або спіральний вибір, і вони не дають якісних результатів. Навіть евристика ентропії, яка вибирає значення елементів послідовно, не врятує від цієї проблеми. Особливо гостро ця проблема постає саме в нашому випадку. Адже в нашому випадку зображення одночасно генерується із багатьох місць, неодмінно ведучи до потреби їх злиття у місцях швів.

Одним зі способів вирішення цієї проблеми є зміна евристики в певний момент на одну із примітивних, мета якої полягатиме в тому, щоб вирішувати конфлікти в проблемних місцях почергово.

3.4.3.2. Локальна модифікація

Цей метод полягає у локальному зміні значень змінних в обмежених областях проблеми, щоб знайти розв'язок без порушення існуючих обмежень. Вибіркові частини задачі модифікуються поетапно, де кожен етап спрямований на вирішення локальних конфліктів.
Локальна модифікаця має кілька переваг над бектрекінгом:

1. Масштабованість: Цей метод концентрується на локальних областях і не потребує значних ресурсів для запам’ятовування попередніх кроків, що робить його придатним для великих і складних задач.
2. Уникнення застрягань: локальна модифікація дозволяє швидко вирішувати конфлікти у місцях, де велика кількість обмежень концентрується в одному регіоні, що вимагає багато часу на аналіз і вирішення у випадку бектрекінгу.
3. Гнучкість у порядку генерації: локальна модифікація не чутлива до порядку генерації, оскільки вона не запам’ятовує попередні кроки. Це дає змогу уникнути проблемі зі злиттямм регіонів, що виникають при використанні бектрукінгу.

3.5. Інтерпретація результатів як ігровий рівень

3.5.1. Задання відображення патернів в об’єкти

Для перетворення згенерованих результатів у ігровий рівень, необхідно встановити відповідність між патернами та об'єктами цього рівня. Проблемою є складність цього завдання, оскільки навіть у простому зразку кількість екстрагованих патернів може сягати сотень. Трансформації патернів не дозволяють зменшити кількість необхідних відображень, оскільки різні трансформації одного і того ж патерну можуть кореспондувати до різних об'єктів. Додатково, різні патерни можуть відповідати одному й тому ж об'єкту, тому що кінцевий об'єкт не завжди залежить від повного контексту патерну. Наприклад, для дверей важливо, щоб вони стикувалися зі стінами, але не має значення, що розміщено перед чи за ними.

Логічним рішенням є інвертування задачі та відображення кожного об'єкта в окремий патерн. Таким чином працює проста експліцитна модель, яка часто застосовується для генерації ігрових рівнів. Основною проблемою цього підходу є необхідність ручного встановлення відносин між усіма патернами, кількість яких може бути дуже великою.

Для вирішення цієї проблеми часто застосовуються системи симетрій та маркування суміжностей. Втім, з урахуванням цих систем, проста експліцитна

модель стає подібною до моделі з перекриттями, де патерн з маркерами ребер можна уявити як патерн розміром 3×3 (або більше), де бічні тайли відображають кольори маркерів, а кутові тайли відсутні.

3.5.2. Шаблонна модель патернів

Враховуючи наведене вище, логічним кроком є спроба поєднати ці дві моделі. Ми пропонуємо відображати кожен об'єкт не в конкретний патерн, а в більш гнучкий шаблон, який може кореспондувати з багатьма патернами з оригінального зразка. Шаблон схожий на звичайний патерн, однак деякі його тайли можуть бути намірно залишені порожніми. В такому випадку порожній тайл може відповідати будь-якому іншому тайлу.

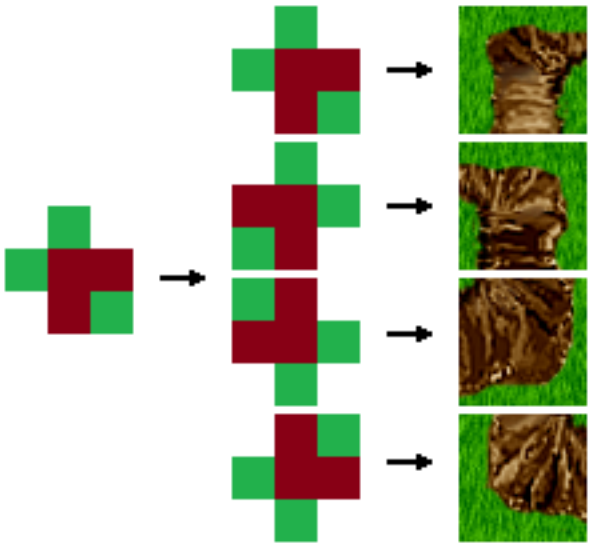

Рисунок 33 Шаблон, його трансформації та відповідні об’єкти

Цей комбінований підхід дозволяє нам використовувати переваги обох моделей. Відображення об'єктів у шаблони дозволяє явно задавати, яким чином мають стикуватися об'єкти та полегшує ідентифікацію об’єкту за патерном. Також, цей метод зберігає можливість декларативного управління генерацією контенту за допомогою вхідного зразка. Окрім того, така конфігурація патернів сприяє їх перевикористанню, збільшуючи тим самим універсальність і адаптивність системи.

3.5.3. Трансформації патернів

Трансформації патернів або шаблонів - це симетрії діедричної групи симетрій квадрата (D4) [20]. Група симетрій квадрата включає обертання та відображення. Загалом, вона містить 8 елементів:

Обертання:

1. r0 або e - тотожне перетворення (обертання на 0°).
2. r90 - обертання на 90° проти годинниової стрілки.
3. r180 - обертання на 180°.
4. r270 - обертання на 270° проти годинниової стрілки.

Відображення:

1. rv - відображення по вертикальній осі (через вісь x).
2. rh - відображення по горизонтальній осі (через вісь y).
3. rm - відображення відносно головної діагоналі.
4. rs - відображення відносно бічної діагоналі.

Формально, D4 описується як [19]:

$$D4 = \langle a, b : a4 = b2 = e, ab = ba^{-1} \rangle \quad (3.1)$$

Де a - поворот на 90° проти годинниової стрілки, b - відображення через вісь x, e - тотожність.

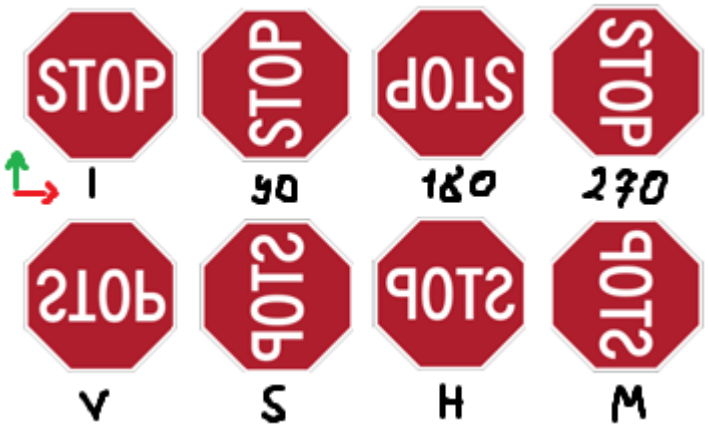

Рисунок 34 Зображення симетрій діедричної групи D4 [20].

Трансформації зокрема використовуються в самому алгоритмі WFC при екстракції патернів зі зразка для впровадження різноманітності. В нашому випадку, вони важливі для спрощення задання відображення, оскільки різні трансформації одного і того ж шаблону можуть відповідати різним, але, як правило, подібним об'єктам.

3.5.4. Групи трансформацій

Всі шаблони можна класифікувати за їхньою групою трансформацій. Кожному шаблону та його всім його можливим трансформаціям відповідає одна група. Кожна група трансформацій відповідає певній математичній підгрупі у діедричній групі симетрій квадрата. Група трансформацій визначає, які трансформації потрібно виконати із врахуванням симетричності патерну, для того щоби отримати всі можливі варіації цього патерну.

Групу симетрій можливо отримати, якщо доповнити формулу (3.1) додатковими обмеженнями, що описують природу зображення.

Всього існує 6 груп:

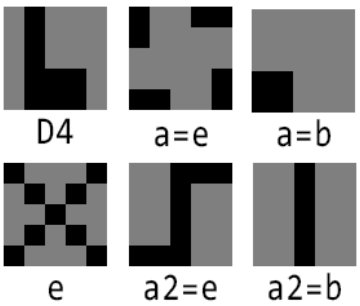

Рисунок 35 6 підгреп у групі D4

1. $D4$ - повний набір трансформацій. Відповідає зображенню у якого немає симетрій.

- Група: $D4 = \langle a, b \mid a4 = b2 = e, ab = ba^{-1} \rangle$
- Елементи: $\{e, a, a2, a3, b, ab, a2b, a3b\}$

2. $b = e$ $(b = a, b = a3)$ - симетрія відображення (або відображення = повороту на 90 чи 270)

- Група: $\langle a \mid a4 = e \rangle$ (Циклічна група C4)
- Елементи: $\{e, a, a2, a3\}$

3. $a = e$ $(a3 = e)$- симетрія обертання

- Група: $\langle b \mid b2 = e \rangle$
- Елементи: $\{e, b\}$

4. $a2 = e$ - Симетрія 180°

- Група: $\langle a, b \mid a2 = b2 = e, ab = ba \rangle$(Група Клейна V4)
- Елементи: $\{e, a, b, ab\}$

5. $a2 = e, b = e$ $(b = a2)$ - Симетрія відображення та 180° (Відображення = 180°)

- Група: $\langle a \mid a2 = e \rangle$ (Циклічна група C2)
- Елементи: $\{e, a\}$

6. $a = e, b = e$ - повна симетрія зображення

- Група: $\langle e \rangle$
- Елементи: $\{e\}$

3.6. Висновки

Аналіз існуючого алгоритму виявив, що його ключовою проблемою є відсутність можливості забезпечення глобальної подібності. Для вирішення цього питання було розроблено нову архітектуру генератора, яка інтегрує методи синтезу текстур з алгоритмом WFC.

Алгоритм ресинтезу Гаррісона було обрано як основу для алгоритму синтезу через його високу швидкість, гнучкість та універсальність. Ми адаптували його, забезпечивши можливість врахування різних трансформацій зразка, що дозволяє алгоритму краще відповідати специфічним вимогам задачі. Також ми використали спіральну евристику синтезу для підвищення цілісності результатів і мінімізації кількості швів.

Головним викликом у в цьому процесі стала інтеграція цих двох методів. Під час дослідження було розглянуто декілька різних евристик, а також здійснено аналіз поведінки бектрекінгу в різноманітних сценаріях. В результаті було вирішено застосовувати евристику максимальної подібності у комбінації з бектрекінгом, що сприяло підвищенню глобальної консистентності генерованого контенту.

Також було запропоновано нову комбіновану модель патернів, що поєднує переваги моделі із перекриттям та простої експіціитної моделі і дає змогу легко задавати відображення патернів в об’єкти, не жертвуючи декларативністю алгоритму.

Розділ 4. Програмна реалізація і тестування розробленого прототипу

4.1. Засоби розробки

Даний генератор розроблявся на ігровому рушіїі Unity із використанням мови програмування C#.

За основу для алгоритму було взято бібліотеку з відкритим кодом DeBroglie [2], яка окрім самого алгоритму реалізує й інший корисний функціонал.

4.2. Огляд системи генерації рівнів в Unity

4.2.1. Структура

Генератор побудований модульним чином, із використанням об’єктної та компонентної системи Unity. Це дозволило відокремити різні частини генератору, такі як евристики, що дає змогу легко заміняти їх на інші без потреби рекомпіляції коду.

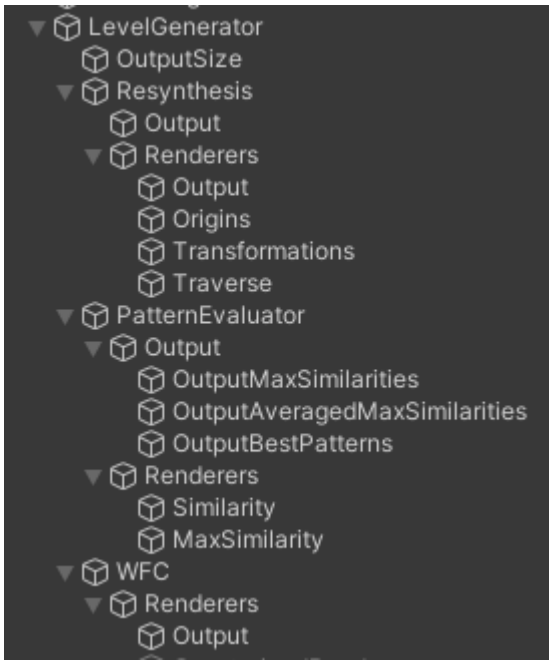

Рисунок 36 Структура генератора у сцені

4.2.2. Модель та аналіз

Більша частина необхідного для генерації аналізу, як побудова матриці суміжностей, відбувається лише один раз по натисненню кнопки користувачем, або зміні зразка. Результати аналізу зберігаються в об’єкті що наслідується від ScriptableObject та серіалізуються системою Unity. Це дає змогу багаторазово перевикористовувати модель для генерації без потреби знову аналузувати зразок, навіть після закриття редактору.

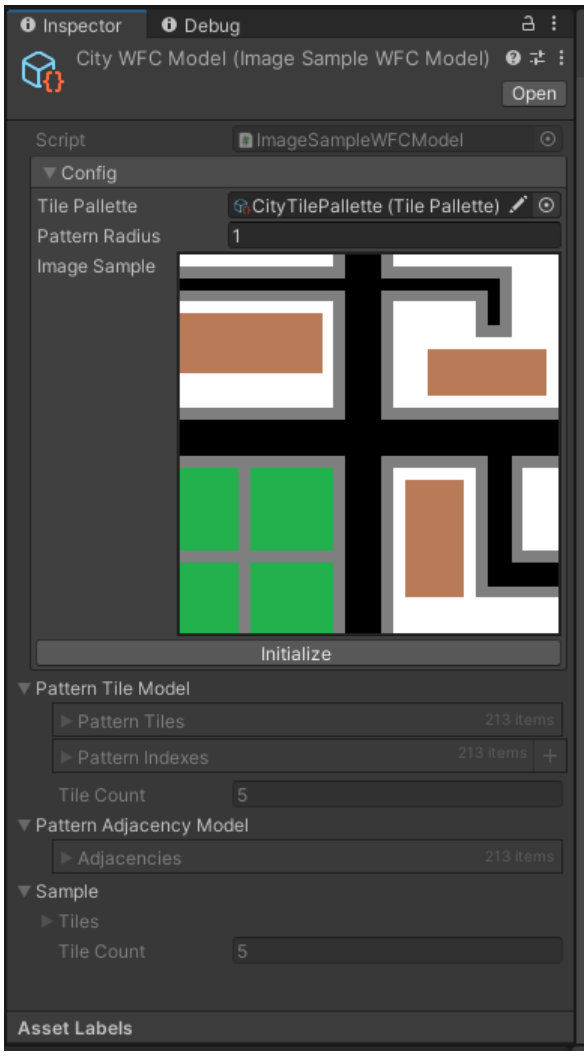

Рисунок 37 Об’єкт моделі, що зберігає всі необхідні для генерації дані

4.2.3. Конфігурація параметрів

Вся конфігурація параметрів відбувається за допомогою інспектора в редакторі Unity. Це дає змогу легко валідувати дані, введені користувачем та інтерактивно відображати зміни.

До прикладу, модуль алгоритму синтезу містить три секції налаштувань:

- Refs - тут вказуються посилання на об‘єкт моделі, інші необхідні модулі та об’єкти-сховища даних вхідних та вихідних даних.
- Params - тут задаються власне параметри відповідного модуля. Важливим параметром є Seed, тобто зерно генерації. Використання одного і того є зерна робить повторні запуски генератора детермінованими, що дозволяє порівнювати результати із різними налаштуваннями та відстежувати проблеми.
- Renderers - тут вказуються посилання на компоненти, що відповідають за візуалізацію різноманітних даних.

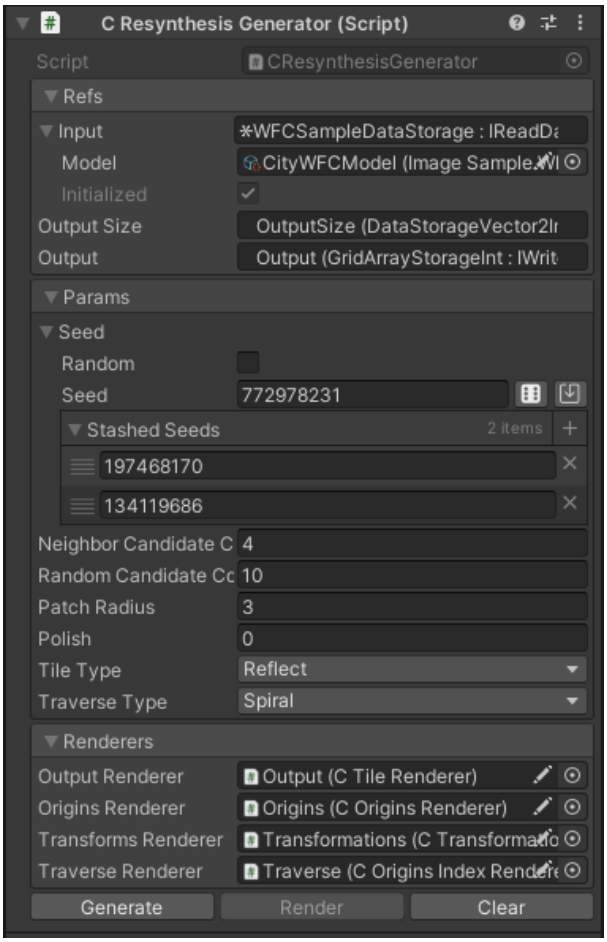

Рисунок 38 Модуль алгритму синтезу

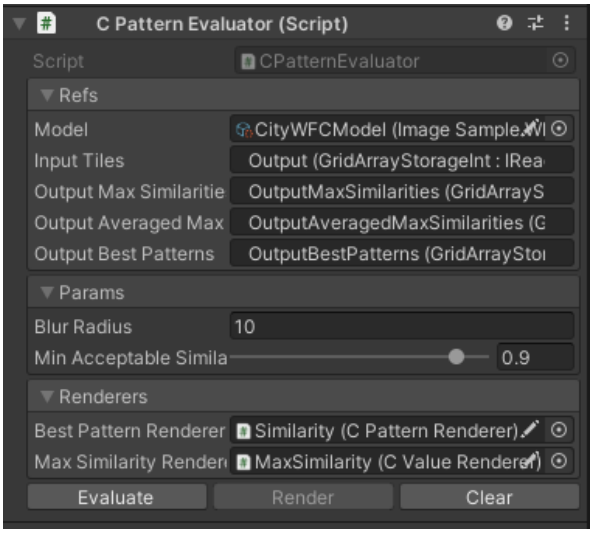

Рисунок 39 Модуль оцінки подібності

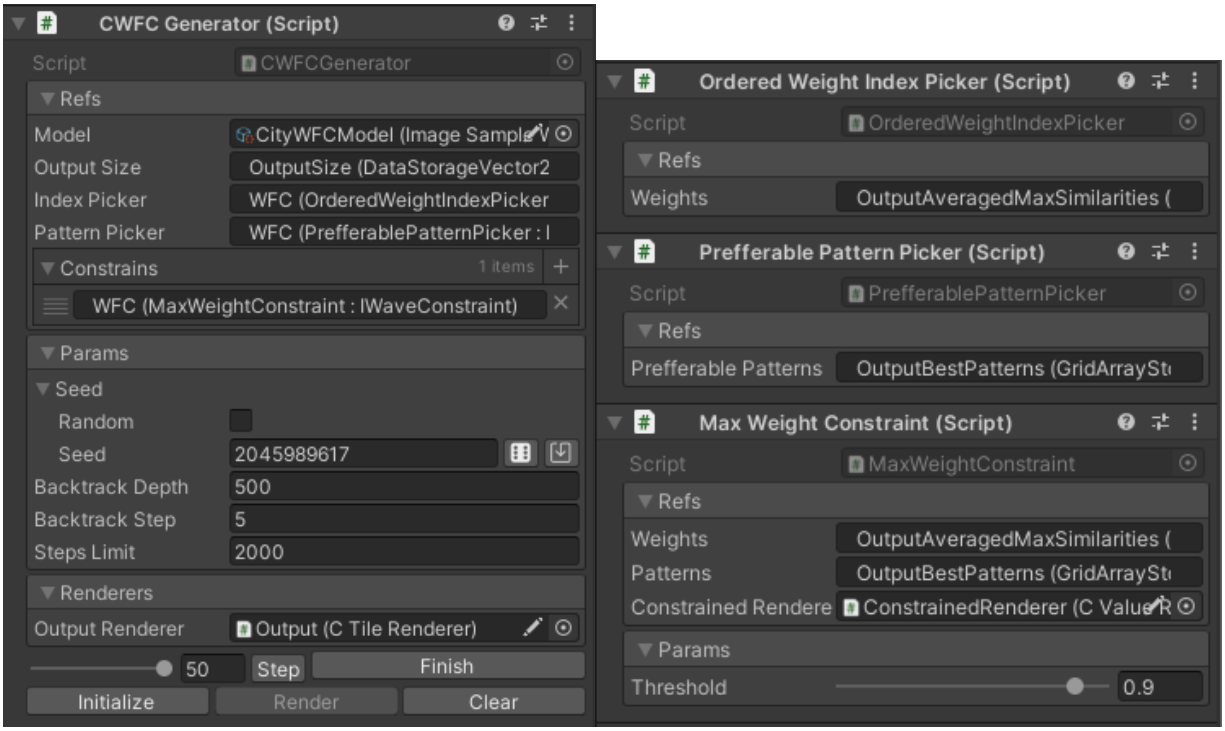

Рисунок 40 Модуль алгоритму WFC (зліва) та евристик і додаткових обмежень (справа)

4.2.4. Виконання

Модулі виконуються послідовно, використовуючи потрібні дані із попередніх модулів. Модулі можна запускати двома способами:

1. Напряму із самого модуля за допомогою відповідної кнопки.
2. За допомогою компоненту генератора, в якому явно вказується їх порядок:

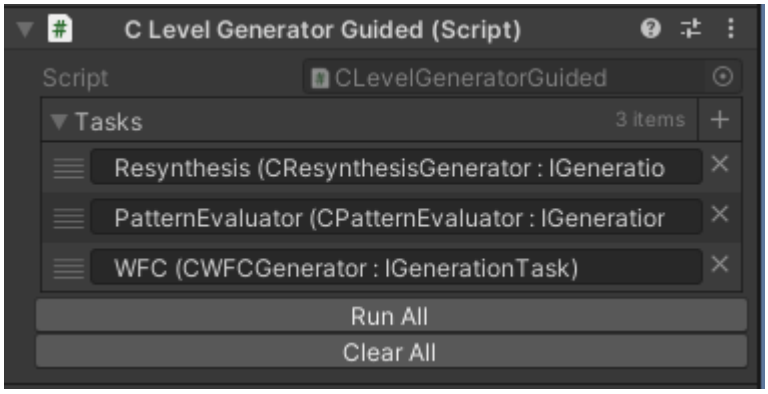

Рисунок 41

4.2.5. Контейнери даних

Результати виконання модулів зберігаються в спеціальних об’єктах-контейнерах призначених для запису-читання:

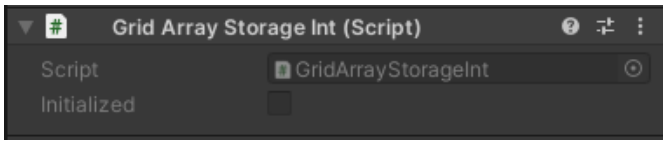

Рисунок 42 Контейнер в якому зберігається результат роботи одного із генераторів

Ці результати не серіалізуються, тобто не зберігаються між сесіями редактора, оскільки призначені лише для використання безпосередньо під час користування генератором.

Додатковою особливістю об’єктів-контейнерів є те, що вони також опрацьовують залежності. Це позбавляє корисутвача потреби вручну запускати модулі в потрібному порядку. Корисутвач може запустити будь-який із них незалежно від їх послідовності. Тоді вибраний модуль звертається до контейнеру для отримання даних. Якщо потрібні дані ще не були записані в контейнер, він посилає запит на їх отримання. Після цього ці дані будуть автоматично згенеровані відповідним модулем та покладені в цей контейнер. В результаті вибраний користувачем модуль перед своїм запуском отримає всі потрібні йому вхідні дані.

Спільні параметри, до прикладу розмір вихідного зображення, також зберігаються в об’єктах-контейнерах. Відмінністю є те, що вони серіалізуються та призначені лише для читання:

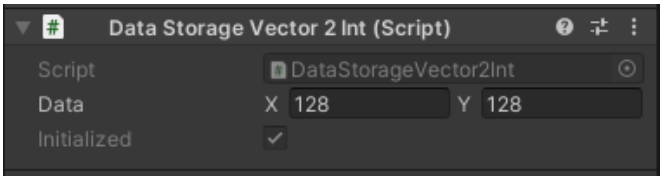

Рисунок 43 Контейнер OutputSize в якому задається розмір вихідного зображення

Використання контейнерів даних забезпечує незалежність модулів один від одного, надаючи системі гнучкість і дозволяючи користувачам компонувати генератор з різних модулів, подібно до конструктора.

4.2.6. Візуалізація

Наостанок, різного типу компоненти рендерингу використовуються для динамічної візуалізації даних. Це дає змогу не лише переглядати кінцевий результат, а також аналізувати різні проміжні дані, що важливо при виявленні проблем і тестуванні різних параметрів.

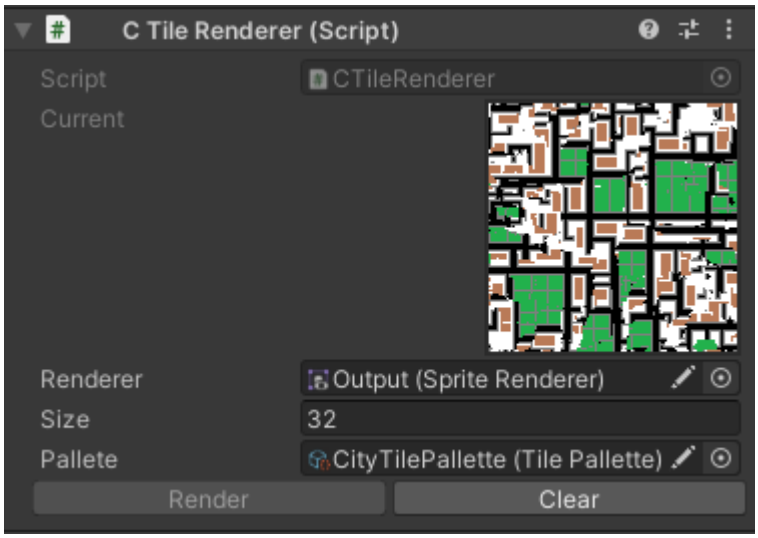

Рисунок 44 Візуалізатор тайлів

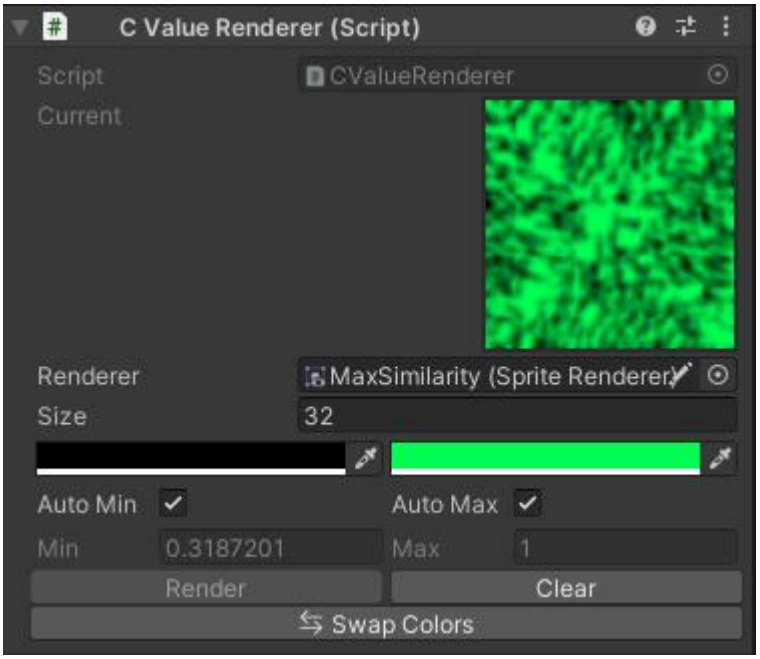

Рисунок 45 Візуалізатор числових даних. Дозволяє задавати колір та діапазон даних, динамічно відображаючи зміни без потреби перерахунку даних.

4.3. Результати експериментальних досліджень

4.3.1. Порівняння результатів з оригінальним WFC

Нижче наведені результати роботи обох генераторів - оригінальний зліва, та наший справа:

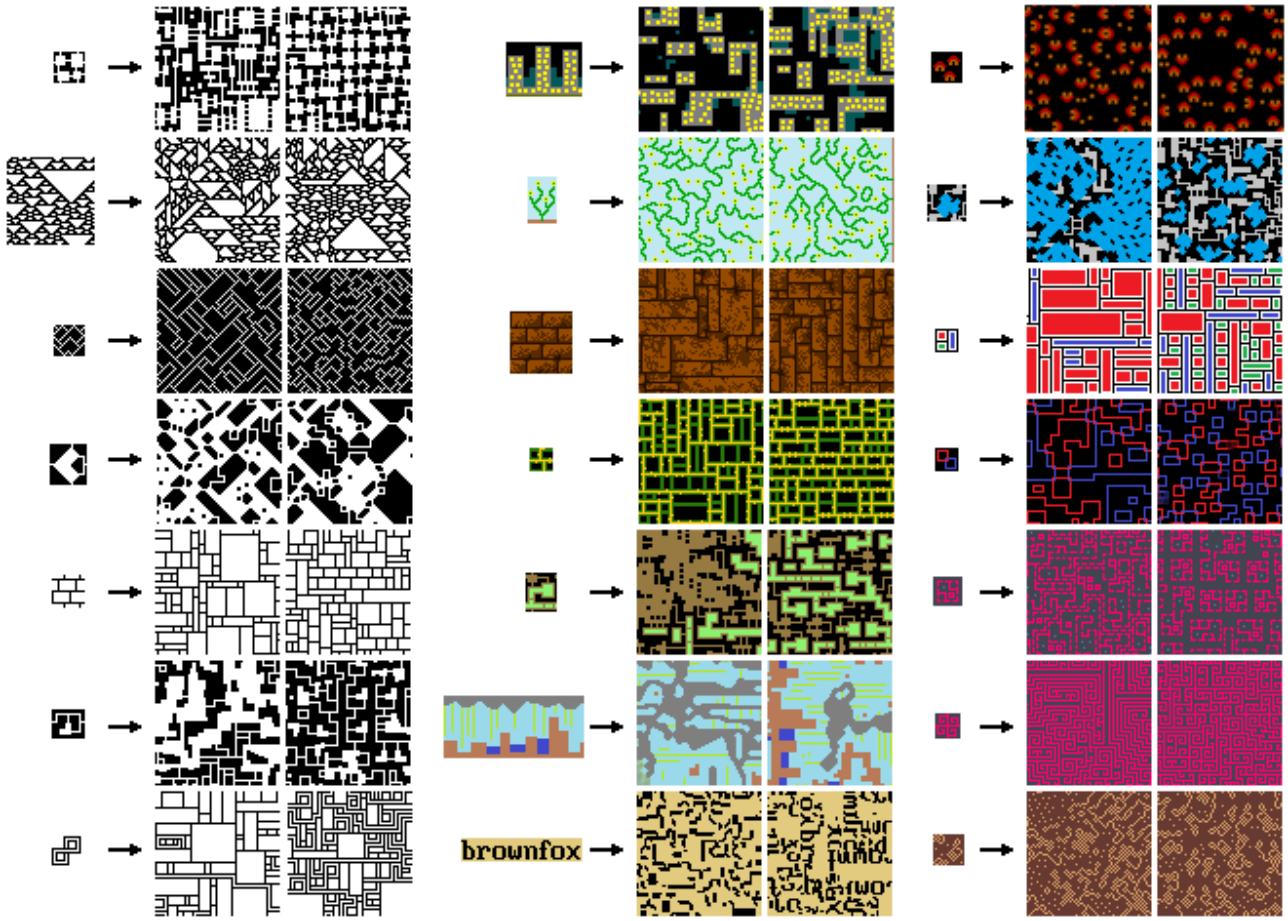

Рисунок 46 Оригінальний метод зліва, наший - справа

Як можна помітити, наш генератор краще справляється із дотриманням глобальної подібності та збереженням структурних особливостей зображення.

Особливо це помітно на більш складних зразках,що задають спражній ігровий рівень:

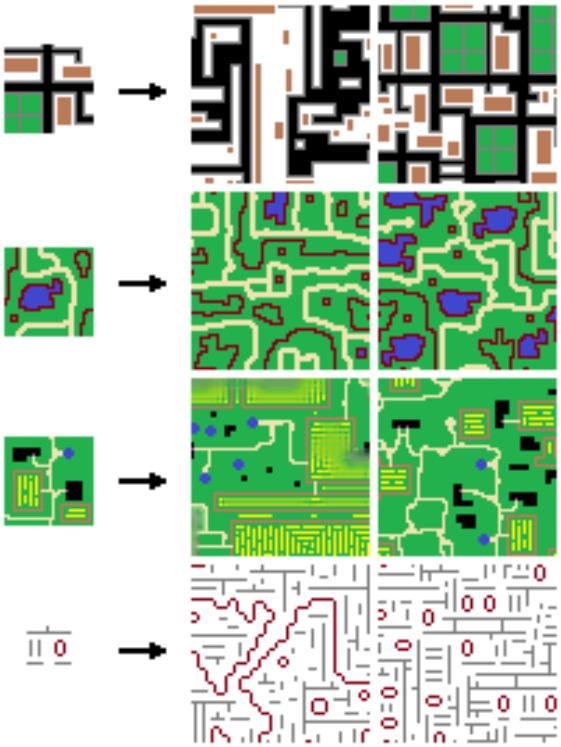

Рисунок 47 Оригінальний метод зліва, наший - справа

Оригінальний алгоритм WFC породжує дуже багато помилок та спотворень зразку, а в більшості складних випадків зовсім не здатен повністю згенерувати результат.

Ми також спробували використати великі зображення, які генерувалися оригінальним алгоритмом із тайлсету, де кожен тайл мав розмір 9x9. Ми подаємо ці зображення як зразок та аналізуємо їх, використовуючи звичайні патерни 3x3:

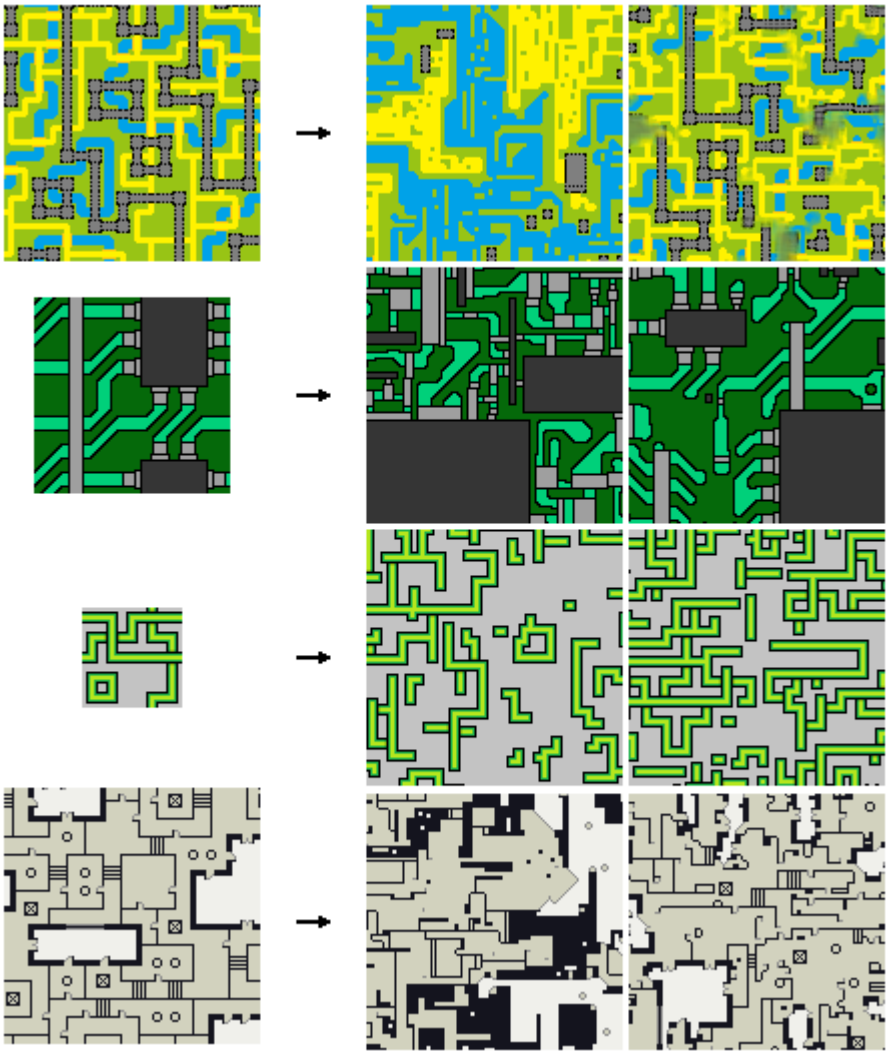

Рисунок 48 Оригінальний метод зліва, наший - справа

Наш алгоритм допускає деякі структурні помилки через невеликий розмір патернів. Проте, як можна бачити, він значно випереджає оригінальний метод. Ці приклади показують що наш алгоритм може бути вдосконалений для роботи із зразками, що є складнішими в десятки разів.

На дуже великих зображеннях якість нашого алгоритму колосально відрізняються від оригінального методу:

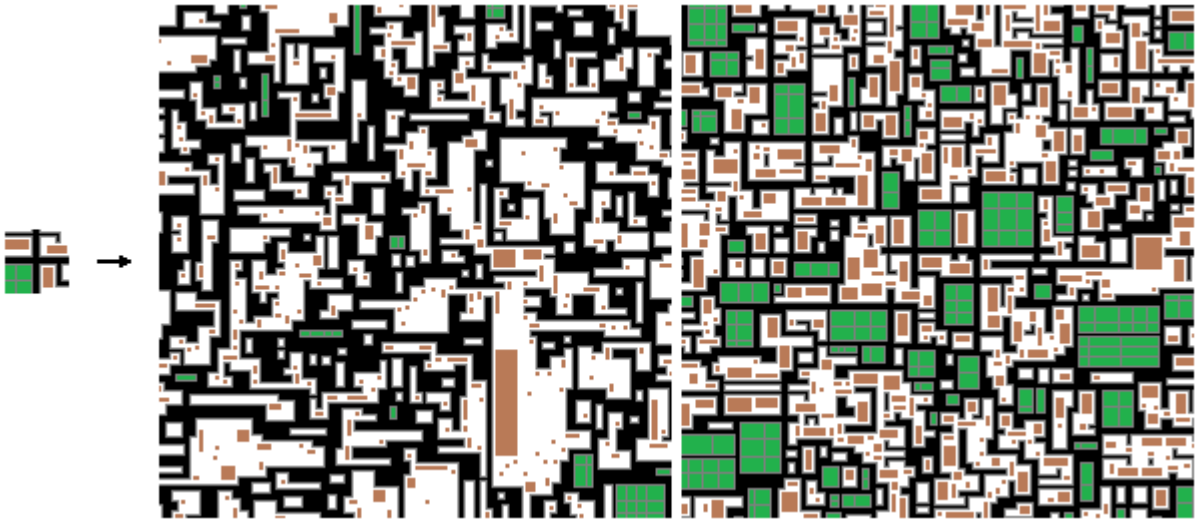

Рисунок 49 Оригінальний метод зліва, наший - справа

4.3.2. Генерація за малюнком

Наш генератор також показав перспективу у використанні малюнків для контролю генерації:

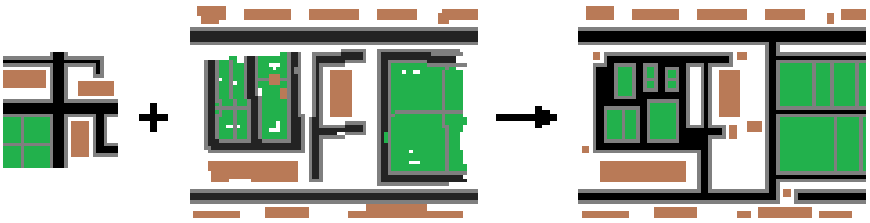

Рисунок 50 використанні малюнків для контролю генерації

Проте даний метод ще потребує удосконалення для того щоби могти працювати із більш схематичними малюнками.

4.3.3. Порівняння аналітичних показників

Наш алгоритм має значну перевагу у кількості кроків спостереження, що необхідні для повної генерації зображення:

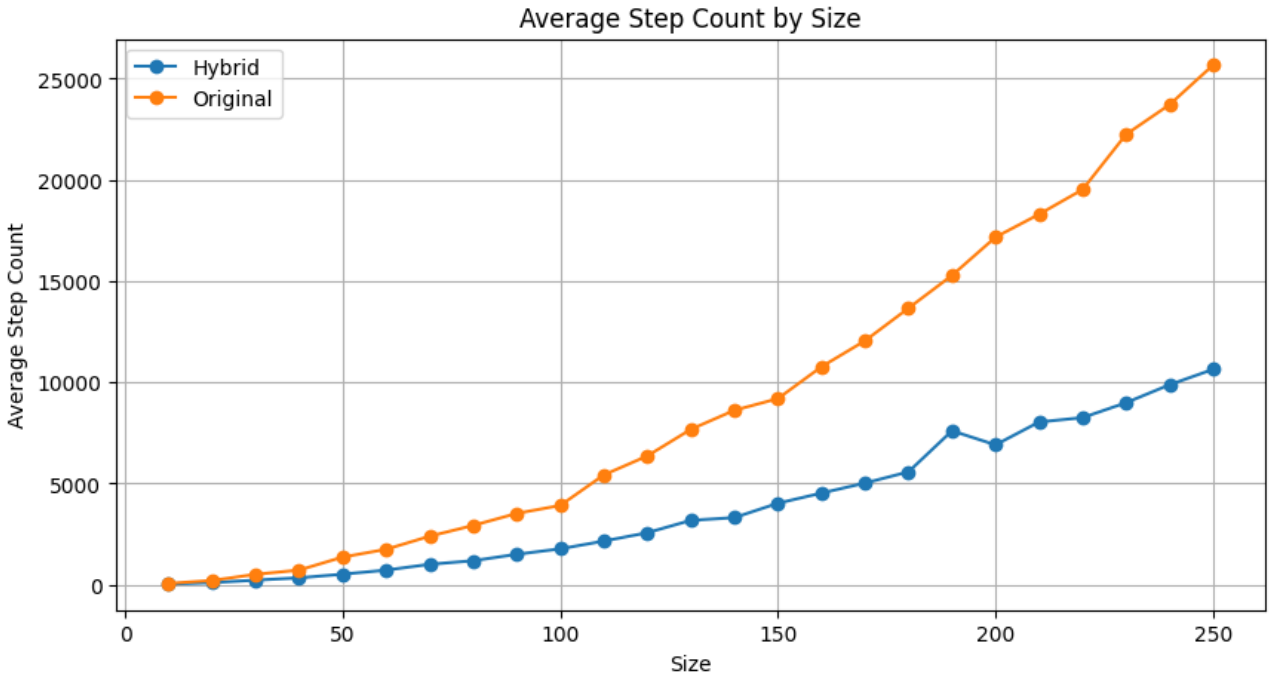

Рисунок 51 Кількість кроків пропагації в залежності від розміру. Розрахунок брався як середнє значення із 10 успішних запусків із випадковим зерном.

Це досягається за рахунок того, що більшість патернів початково констистентні між собою, а тому вони обираються на етапі пропагації, а не спостереження. Це добре видно на візуалізаціях:

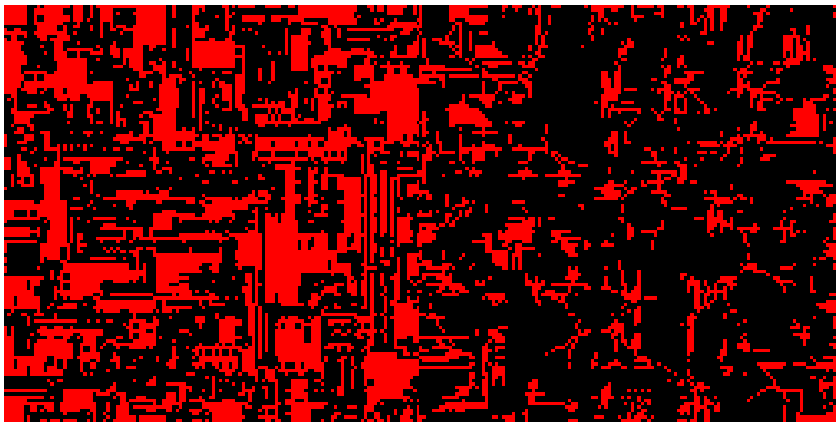

Рисунок 52 Карта кроків спостереження. Червоним позначені патерни, вибрані на етапі спостереження, чорним - на етапі пропагації. Зліва - оригінальний алгоритм, справа - наший.

Тестування підтвердло наше припущення про неефективність бектрекінгу при одночасному злитті багатьох регіонів:

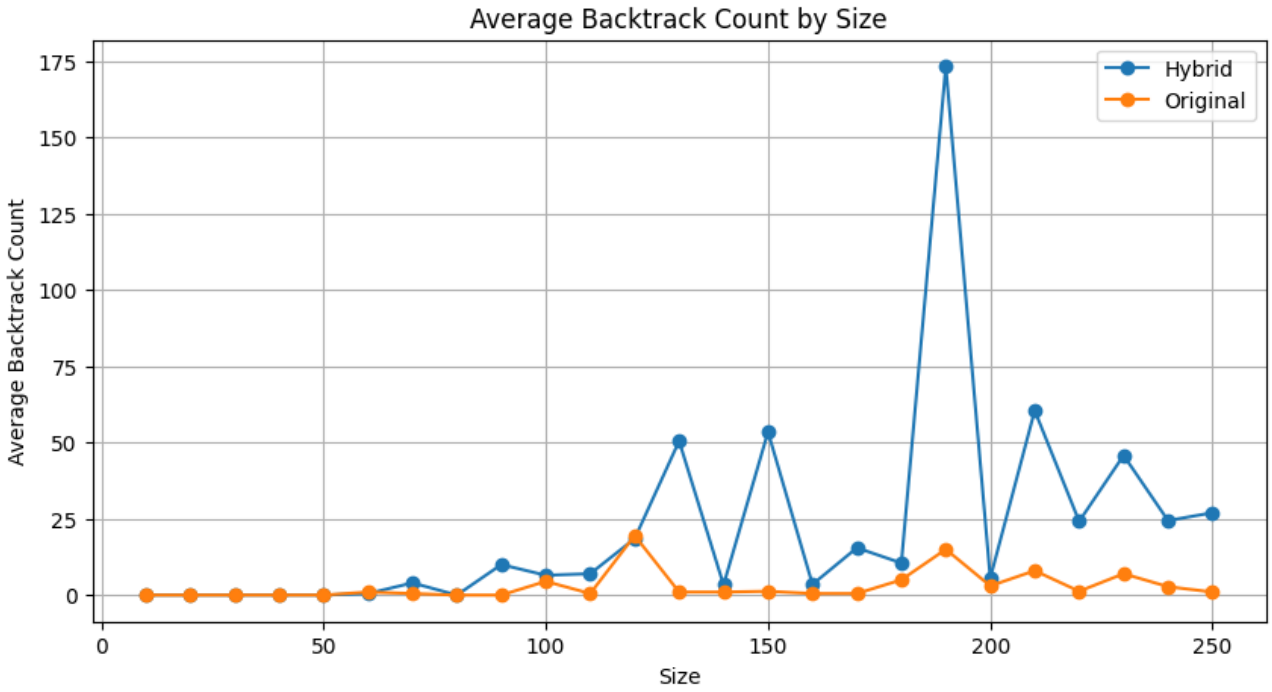

Рисунок 53 Кількість відмін бектрікінгу в залежності від розміру. Розрахунок брався як середнє значення із 10 успішних запусків із випадковим зерном.

Особливо проблема застрягання бектрекінгу присутня при роботі зі зразками, що містять шляхоподібні структури. Це пов'язано із особливістю евристики вибору нашого алгоритму. В таких випадках вона змушує алгоритм бектрекінгу відкидати більшу частину згенерованих результатів. Це відбувається через потребу у вирішенні конфліктів, зерно яких було посіяне дуже рано. Одним із способів вирішення цієї проблеми імовірно може бути заміна алгоритму бектрекінгу на локальний пошук.

На бектрекінг припадає значна частина часу генерації, оскільки він спричиняє відкидання вже згенерованих даних і веде до їх перестворення. Скільки саме часу витрачається на бектрекінг напряму залежить від зерна генерації та обраних евристик.

Незважаючи на це, наш алгоритм виявився асимптотично швидшим, що видно на великих розмірах зображень. При цьому швидкість самого лише алгоритму WFC в нашій реалізацій вдвічі перевищує швидкість оригінального WFC на зображеннях великого розміру, що свідчить про кращу масштабованість нашого підходу:

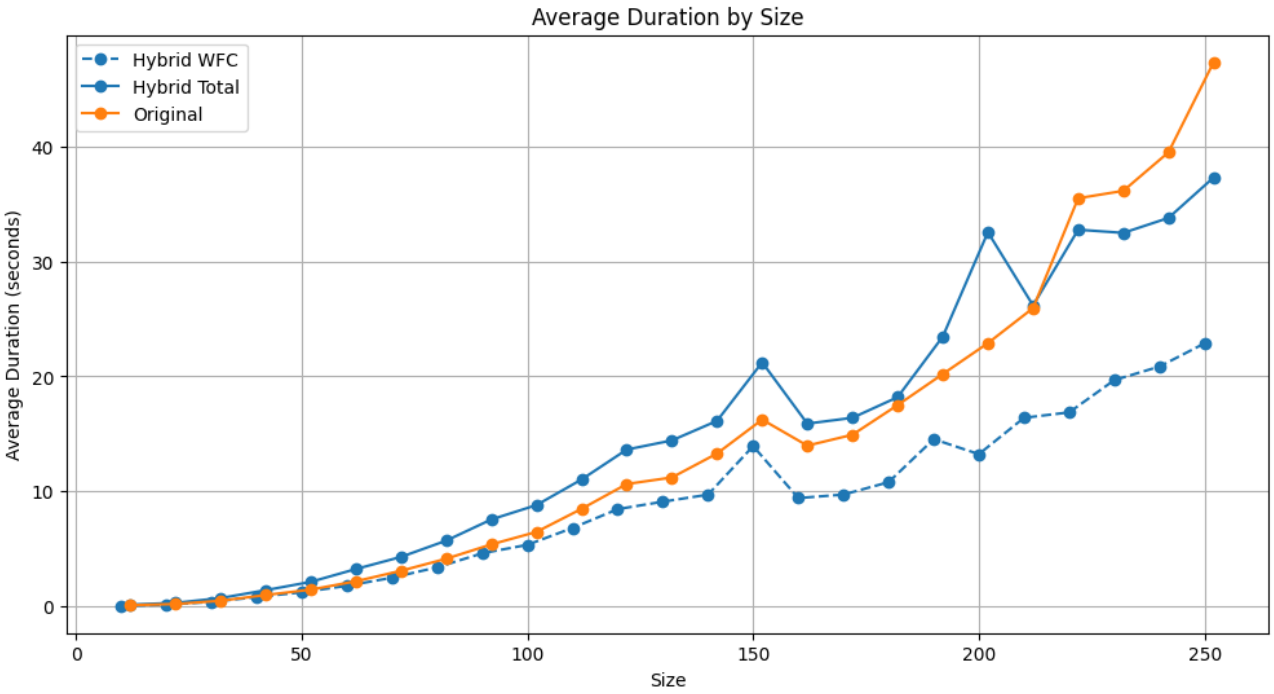

Рисунок 54 Час виконання в секундах в залежності від розміру. Розрахунок брався як середнє значення із 10 успішних запусків із випадковим зерном.

Точно можна сказати, що пріоритетним напрямком оптимізації нашого алгоритму є покращення евристик та бектрекінгу задля отримання якомога більшого коефіцієнту корисної роботи, яку виконує алгоритм.

4.4. Висновки

Ми розробили гібридний генератор, який відповідає встановленим критеріям: глобальній та локальній подібності, універсальності, а також декларативності. Система, створена на платформі Unity, виявилася винятково гнучкою та зручною у використанні. Генератор ефективно демонструє свої можливості на практиці, часто перевищуючи оригінальний алгоритм за якістю результатів та ефективністю виконання.

Основні переваги гібридного генератору включають здатність досягати глобальної подібності та збереження макроструктурних особливостей зображень. Він також ефективно справляється зі складними зразками.

Проте, генератор має декілька недоліків, таких як доволі тривалий час генерації та тенденція до затрягання при роботі зі зразками, які містять шляхоподібні структури. Це виявило необхідність у подальшому удосконаленні або заміні

алгоритму пошуку, що використовується для вирішення задачі виконання обмежень.

Висновки

Алгоритми процедурної генерації контенту відіграють важливу роль в ігровій індустрії. Проте вони стикаються з проблемами обмеженої універсальності та складності управління. Алгоритм Wave Function Collapse використовує методи на основі обмежень та принци алгоритмів синтезу текстур, які дозволяють адресувати ці виклики.

В рамках дослідження ми провели глибокий аналіз теоретичних основ алгоритму колапсу хвильової функції (WFC). У роботі описано математичну основу алгоритму WFC, сформульовано задачу виконання обмежень (CSP) та її вирішення за допомогою методу узгодження дуг AC3. Описано модульну структуру алгоритму WFC та обґрунтовано використання різних евристик, таких як MRV. Виділені переваги та недоліки алгоритму WFC, виявлено потенціал для подальших вдосконалень. Порівняно його із іншими існуючими методами: Perlin Noise, Cellular Automata, Markov Chain, Lindenmayer Systems, Texture Synthesis, Model Synthesis, Graph Grammars.

На базі Unity та мові C# розроблено прототип системи генерації ігрових рівнів, яка інтегрує вдосконалений алгоритм синтезу текстур П. Гаррісона та алгоритм WFC, реалізований бібліотекою DeBroglie. Описано різні методи оцінки подібності патернів та виявлення швів. Випробувано різні евристики вибору патернів та порядку генерації. Досліджено поведінку пошуку із поверненням, виявлено проблему злиття регіонів та запропоновано рішення. Використано нову шаблонну модель патернів та діедричну групи симетрій квадрата D4 для задання відображень патернів в об’єкти та інтерпретації згенерованих результатів як ігрових рівнів. Наведено огляд інтерфейсу та описано спосіб роботи із застосунком на Unity.

Тестування розробленої системи показало що вона задовольняє заданим критеріям: локальній та глобальній подібності, універсальності та декларативності. Створено порівняльні таблиці нашого та існуючого алгоритмів, які демонстують кращу якість результатів генерації та здатність обробляти

складні зразки. Виявлено потенціал використання системи для генерації за схематичними малюнками користувача. Проаналізовано ефективність прототипу на різних розмірах зображення. Аналітичні результати показали асимптотичну перевагу часу виконання над існуючим алгоритмом WFC. Кількість кроків спостереження які робить алгоритм WFC в нашій реалізацій при цьому майже в 2 рази менша. Аналітично підтверджено проблему пошуку із поверненням та злиття регіонів.

Підсумовуючи, алгоритм WFC показав свою перспективність та іноваційність. Його інтеграція з методами синтезу зображень дозволяє досягнути високого рівня універсальності та декларативності, що є рідкісними властивостями для інших методів процедурної генерації. Це відкриває нові можливості для дизайнерів та розробників ігор і не лише. Однак, було виявлено деякі недоліки, що підкреслює важливість проведення подальших досліджень в області процедурної генерації ігрових рівнів з використанням WFC.

Список використаної літератури

1. Gumin M. WaveFunctionCollapse [Електронний ресурс] / M. Gumin. — Режим доступу: https://github.com/mxgmn/WaveFunctionCollapse.
2. BorisTheBrave DeBroglie [Електронний ресурс]. — Режим доступу: https://github.com/BorisTheBrave/DeBroglie.
3. Merrell P. Model Synthesis [Електронний ресурс] / P. Merrell. — Режим доступу: https://paulmerrell.org/model-synthesis/.
4. Merrell P. Model Synthesis [Текст] : дис. / P. Merrell. — 2021. — Режим доступу: https://paulmerrell.org/wp-content/uploads/2021/06/thesis.pdf.
5. Merrell P. Comparison [Текст] / P. Merrell. — 2021. — Режим доступу: https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf.
6. Stangl M. Wave Function Collapse Is Constraint Solving In The Wild [Електронний ресурс] / M. Stangl. — Режим доступу: https://umm-csci.github.io/senior-seminar/seminars/spring2017/stangl.pdf.
7. Smith A. Wave Function Collapse Is Constraint Solving In The Wild [Електронний ресурс] / A. Smith. — Режим доступу: https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.pdf.
8. Smith A. [Текст] / A. Smith. — Режим доступу: https://escholarship.org/content/qt1fb9k44q/qt1fb9k44q_noSplash_1c5dcf5090d4595f7605b2653c89b245.pdf?t=qwpcsb.
9. Yuhe N. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm [Електронний ресурс] // ArXiv. — 2023. — Режим доступу: https://arxiv.org/pdf/2308.07307.
10. Han H. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm [Електронний ресурс] / H. Han. — Режим доступу: https://www.researchgate.net/profile/Honglei-Han/publication/348204502_Automatic_Generation_of_Game_Levels_Based_on_Controllable_Wave_Function_Collapse_Algorithm/links/6007c39f299bf14088aa6c08/Automatic-Generation-of-Game-Levels-Based-on-Controllable-Wave-Function-Collapse-Algorithm.pdf.

11. Arunpreet S. [Текст] // ACM. — Режим доступу: https://dl.acm.org/doi/pdf/10.1145/3337722.3337752.
12. Wauthier T. GW 2223 [Текст] / T. Wauthier. — Режим доступу: https://tristanwauthier.com/PDF/GW_2223_Tristan_Wauthier_EN_Paper.pdf.
13. Gumin M. TextureSynthesis [Електронний ресурс] / M. Gumin. — Режим доступу: https://github.com/mxgmn/TextureSynthesis.
14. Lin L. Real-Time Texture Synthesis By Patch-Based Sampling [Текст]. — Режим доступу: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2001-40.pdf.
15. Li-Yi W. Fast Texture Synthesis using Tree-structured Vector Quantization [Текст]. — Режим доступу: https://graphics.stanford.edu/papers/texture-synthesis-sig00/texture.pdf.
16. Michael A. Synthesizing Natural Textures [Текст]. — Режим доступу: https://www.cs.princeton.edu/courses/archive/fall10/cos526/papers/ashikhmin01a.pdf.
17. Harrison P. Texture Synthesis, Texture Transfer, and Plausible Restoration. [Текст] : дис. — Режим доступу: https://logarithmic.net/pfh-files/thesis/dissertation.pdf.
18. Merrell P. Modelling & Graph Grammars [Електронний ресурс] / P. Merrell. — Режим доступу: https://paulmerrell.org/grammar/.
19. Definition: Dihedral Group D4 [Електронний ресурс]. — Режим доступу: https://proofwiki.org/wiki/Definition:Dihedral_Group_D4.
20. Дієдральна група [Електронний ресурс] // Вікіпедія. — Режим доступу: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D1%94%D0%B4%D1%80%D0%B8%D1%87%D0%BD%D0%B0_%D0%B3%D1%80%D1%83%D0%BF%D0%B0.
21. Задача виконання обмежень [Електронний ресурс] // Вікіпедія. — Режим доступу: https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D0%B0%D0

%BD%D0%BD%D1%8F_%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D1%8C.

22. General video game level generation [Текст] // University of Malta. — 2016. — Режим доступу: https://www.um.edu.mt/library/oar/bitstream/123456789/81555/1/General_video_game_level_generation_2016.pdf.
23. Rolling Your Own Finite-Domain Constraint Solver [Текст] // Game AI Pro. — Режим доступу: https://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter26_Rolling_Your_Own_Finite-Domain_Constraint_Solver.pdf.
24. Пошук з вертанням [Електронний ресурс] // Вікіпедія. — Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D0%B7_%D0%B2%D0%B5%D1%80%D1%82%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC.
25. Scratchapixel. Perlin Noise Terrain Mesh [Електронний ресурс]. — Режим доступу: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise-terrain-mesh.html.
26. Johnson D., Yannakakis G. N. Cellular automata for real-time generation of infinite cave levels [Текст] // Semantic Scholar. — Режим доступу: https://www.semanticscholar.org/paper/Cellular-automata-for-real-time-generation-of-cave-Johnson-Yannakakis/4e764b67f54b9974b7b9d1bcdb631b676b8b88b6.
27. Milne A. C. Markov [Електронний ресурс] // Andrew C. Milne. — Режим доступу: https://www.andrewcmilne.com/markov.
28. Gumin M. ConvChain [Електронний ресурс] / M. Gumin. — Режим доступу: https://github.com/mxgmn/ConvChain.
29. Yoav P. Procedural Modeling of Cities [Текст] // ETH Zurich. — 2001. — Режим доступу: https://cgl.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf.

30. Embark Studios. Texture Synthesis [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/EmbarkStudios/texture-synthesis.
31. Townscaper [Електронний ресурс] // Steam. — Режим доступу: https://store.steampowered.com/app/1291340/Townscaper/.
32. Caves of Qud [Електронний ресурс] // Steam. — Режим доступу: https://store.steampowered.com/app/333640/Caves_of_Qud/.
33. Andy Makes. Global Game Jam 2019: Maureen's Chaotic Dungeon [Електронний ресурс] // Tumblr. — Режим доступу: https://www.tumblr.com/andymakesgames/182363131350/global-game-jam-2019-maureens-chaotic-dungeon.
34. [Авт. невід.] [Електронний ресурс] // 51WMA. — Режим доступу: https://www.51wma.com/en/games/1492.
35. Texture synthesis and remixing from a single example [Електронний ресурс] // Medium. — Режим доступу: https://medium.com/embarkstudios/texture-synthesis-and-remixing-from-a-single-example-faf5f4e8a5b8.