УДК 004.42; 004.67 DOI: 10.31673/2786-8362.2024.010909

Ільїн О.О. д.т.н., **Іщеряков С. М.** к.т.н.,
Карпов А.О., Єрмоленко В.О.

РОЗРОБКА СИСТЕМИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ КАРТ НА БАЗІ АЛГОРИТМУ КОЛАПСУ ХВИЛЬОВОЇ ФУНКЦІЇ

Ilin O.O., Ishcheryakov S.M., Yermolenko V.O., Karpov A.O., Development of the system of procedural generation of game maps based on the wave function collapse algorithm. The article discusses the basic scientific principles underlying the wave function collapse algorithm in the context of creating game levels and game maps. The wave function collapse algorithm is based on the principles of quantum mechanics and the idea of a wave function for creating structurally complex objects. Based on this algorithm, a systematic approach to procedural generation is built, which allows for more efficient generation of game levels in conditions of increased game space. This paper examines in more detail the features of the application of the wave function collapse algorithm in tile-type games with voxel graphics. The platform for practical testing of the proposed adapted algorithm is Unity, MagicaVoxel, C#. Obtained results show that this approach avoids artifacts and errors in the generated game locations, and also achieves a satisfactory uniqueness and structure of the location of the elements of the game world.

Key words: procedural generation, gameplay, WFC, C#, Unity, tile-based games

Ільїн О.О., Іщеряков С. М., Єрмоленко В.О., Карпов А.О. Розробка системи процедурної генерації ігрових карт на базі алгоритму колапсу хвильової функції. У статті розглянуто основні наукові принципи, які лежать в основі алгоритму колапсу хвильової функції в контексті створення ігрових рівнів та ігрових карт. В основу алгоритму колапсу хвильової функції покладені принципах квантової механіки та ідея хвильової функції для створення структурно-складних об'єктів. На основі даного алгоритму побудований системний підхід процедурної генерації, який дозволяє більш ефективно генерувати ігрові рівні в умовах збільшення ігрового простору. В даній роботі більш детально досліджуються особливості застосування алгоритму колапсу хвильової функції у іграх тайлового типу, з воксельною графікою. Платформою для практичної апробації запропонованого адаптованого алгоритму є Unity,MagicaVoxel, C#. Отримані результати свідчать, що такий підхід дозволяє уникнути артефактів та помилок у згенерованих ігрових локаціях, а також досягається задовільна унікальність та структура розташування елементів ігрового світу.

Ключові слова: процедурна генерація, ігрові рівні, WFC, C#, Unity, тайли

Вступ

Наразі в галузі розробки комп’ютерних ігор виникає наступна етапна вимога — вдосконалення методів генерації ігрового контенту для забезпечення непередбачуваності та різноманітності умов ігри (т.з. геймплею). Відповідно до цієї потреби, надзвичайно важливим стає використання систем процедурної генерації ігрових рівнів. Із збільшенням ігрового простору та відповідного збільшення обсягу інформації та вдосконаленням технологій, виникає потреба в пошуку ефективних методів оптимізації алгоритмів генерації. Алгоритм колапсу хвильової функції, заснований на принципах квантової механіки, представляє собою потужний інструмент для досягнення цієї мети. Ця стаття присвячена розгляду важливого аспекту цього напрямку — розробці підсистеми комп’ютерної гри для процедурної генерації ігрових рівнів з використанням алгоритму колапсу хвильової функції. Детальний аналіз та висвітлення теоретичних нюансів цього алгоритму, а також його практичне застосування дозволяють зрозуміти, яким чином цей інструмент може сприяти досягненню високого стандарту в ігровому дизайні та розширенню можливостей розробників комп’ютерних ігор (геймдевелоперів).

Постановка завдання. Теоретична складова присвячена аналізу ключових концепцій процедурної генерації, включаючи використання алгоритмів, математичних моделей, та структур даних. Основна мета - зрозуміти, як ці підходи можна успішно впроваджувати в реальні ігрові

проекти. Практична частина надає конкретні приклади використання процедурної генерації в різних видах ігор, дозволяючи отримати уявлення про практичні аспекти впровадження цих концепцій. Втім, без уваги залишились питання, пов'язані із застосуванням цілісного набору програмних засобів (стеку технологій) для втілення даних ідей у комп'ютерній грі та з конкретними технічними умовами: типом графіки, виглядом ігрових локацій і т.п.

Також поза увагою залишились питання стосовно адекватності (т.з. «іграбельності») створюваних карт для користувачів, бо це в більшому ступені залежить від геймплею конкретної ігри. Дана стаття в більшій мірі присвячена саме цим аспектам.

Аналіз останніх досліджень. В процесі дослідження теми були проаналізовані матеріали сучасних праць таких авторів як Tanya X. Short та Tarns Adams [1], Ryan Watkins [2] та Noor Shaker, Julian Togelius, та Mark J. Nelson. [3]. В зазначених роботах значний акцент робиться на теоретичних та окремих практичних аспектах процедурної генерації в ігровому дизайні. Детально розглядаються методи та стратегії створення випадкового чи варіативного ігрового вмісту.

Метою роботи є дослідження особливостей застосування алгоритму колапсу хвильової функції для розробки ігрових карт тайлового типу, у іграх з воксельною графікою та на базі стеку технологій Unity, C# та MagicaVoxel.

Виклад основного матеріалу дослідження.

Алгоритм колапсу хвильової функції (Wave Function Collapse, WFC) базується на принципах квантової механіки та використовує ідеї хвильової функції для створення структурно-складних об'єктів. У своїй суті, WFC моделює ігровий рівень як систему хвильових функцій, де кожен піксель чи блок представляється як можлива функція стану. В подальшому ми розглянемо, як саме система хвильових функцій може використовуватися для генерації унікального контенту. Згадана система хвильових функцій походить з області квантової механіки [1], де хвильові функції описують стан частинок та їхнє поведінку у просторі. У контексті алгоритму WFC для ігрової індустрії, цей принцип використовується для моделювання можливих станів елементів ігрового середовища. В цьому випадку система хвильових функцій представляє елементи ігрового рівня як систему можливих станів, кожен з яких має свої характеристики. Наприклад, якщо розглядати окремий піксель чи блок ігрового світу, його можна уявити як хвильову функцію, яка описує різні можливі стани цього елемента.

Перед початком генерації ігрового рівня система визначає початковий стан ігрового рівня, де кожен елемент сітки ігрового поля (Рис. 1) має власний перелік можливих станів.

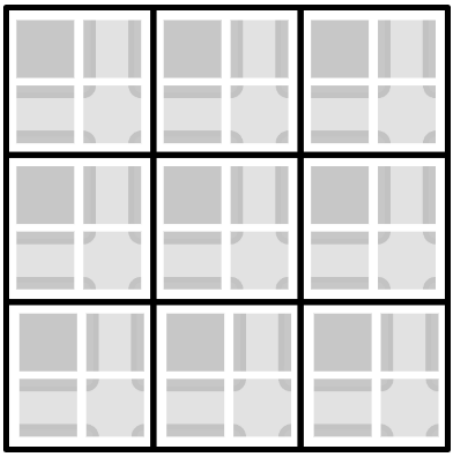

Рис. 1. Представлення ігрового рівня у вигляді сітки комірок,
для кожної з яких є власна система тайлів

На наступному кроці, шляхом ітеративного вибору та колапсу хвильової функції [1], програмна система обирає конкретний стан для кожного елемента сітки (Рис. 2), враховуючи обмеження та взаємодію з навколишніми елементами. Як можна побачити з рисунку 2, обране перехрестя (перша клітинка, яка була обрана алгоритмом довільно) не може бути сусідньою з пустою клітинкою трави. Тому друга клітинка колапсувала у продовження стежинки з першої клітинки сітки.

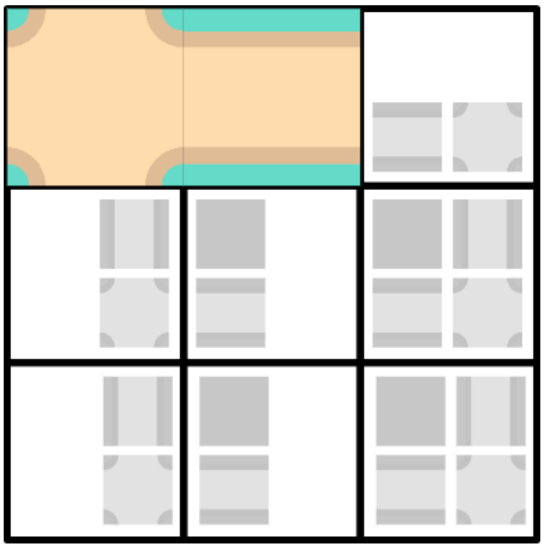

Рис. 2. Колапс двох сусідніх комірок ігрової сітки

Завдяки такому підходу система хвильових функцій дозволяє досягти великої варіативності і структурної складності у генерованих ігрових рівнях. Цей підхід виокремлюється своєю здатністю адаптуватися до різноманітних структур та завдяки цьому забезпечує значну унікальність згенерованих рішень.

Одним із головних елементів алгоритму є система тайлів, яка забезпечує різноманітність положень елементів ігрового поля. Тайл (від англійського tile – плитка) – це найменша структурна одиниця алгоритму WFC і саме вона є предметом хвильової функції, яка відповідає за вибір того чи іншого тайлу у певній клітинці сітки ігрового поля. В нашому випадку, вона представлена у вигляді чотирьох малих клітинок усередині кожної з «великих» клітин сітки (Рис. 1). Під час роботи алгоритму всі тайли існують всередині кожної великої клітини у стані суперпозиції (звідси й назва колапсу хвильової функції, у якій світло існує як хвиля і частка одночасно). Керуючись системою правил, заданих програмістом, алгоритм колапсу хвильової функції, розповсюджуючись довільно у всі боки від випадково обраної «стартової» клітини сітки (примітка, тайл у стартовій клітині на першому кроці так само вибирається довільно), вирішує, які тайли із загального списку суперпозиції вибрати таким чином, щоб скласти логічний, структурований "малюнок" ігрового рівня. Наприклад, нами було встановлено, що кількість тайлів стежок не може бути більшою за кількість тайлів трави. Таким чином, якщо у алгоритму постає питання – що вибрати в порожній клітці сітки – ще один тайл окремої стежки або тайл трави – він вибере тайл трави, щоб не перевантажувати локацію зайвими елементами. Принцип того, як алгоритм розуміє, які тайли поєднуються, а які – ні, буде розглянуто далі.

Обрання засобів розробки та візуалізації системи WFC. Розробка системи генерації ігрових рівнів на базі WFC вимагає застосування відповідного інструментарію. З метою практичної реалізації розглянутого алгоритму було використано платформу розробки ігрових додатків Unity, 3D-редактор MagicaVoxel та мову програмування C#.

Платформа Unity є кросплатформним інструментом для розробки ігор. Зручність його графічного інтерфейсу, гнучкість та універсальність середовища, а також можливість писати скрипти мовою C# дозволить реалізувати всі переваги алгоритму WFC, а також візуалізувати його в будь-якому зручному представленні без потреби розробляти власний ігровий «двигун».

У 3D-редакторі MagicaVoxel буде виконано побудову та рендеринг всіх тайлів, які в подальшому були використані в середовищі розробки Unity.

Слід зазначити, що існує два способи роботи з даним алгоритмом. Перший спосіб – це застосування разом із алгоритмами машинного навчання, якому надається зразок рівня, який необхідно згенерувати. Аналізуючи з його допомогою алгоритм WFC самостійно генерує (створюючи власну систему правил) і самі тайли і, відповідно, локацію з них. Цей спосіб має декілька нюансів. Перший - його неможливо використовувати для генерації 3D-структур через надмірну комплексність задачі [2] та, як відповідно, незадовільність результатів. Другий - неможливість створювати тайли вручну, через що створені локації виглядають занадто штучно і можуть поєднувати в собі безліч помилок і артефактів. Другий спосіб полягає в «ручному» створенні всіх тайлів і подальшому написанні правил для їх коректного розташування в просторі. Позитивна сторона цього методу полягає в тому, що його можна використовувати як у 2D, так і в 3D середовищах. Хоча від розробника і передбачається ручна візуалізація всіх елементів-тайлів, але практична складова реалізації стає набагато простішою та універсальнішою. Таким чином, в якості практичної реалізації було обрано шлях ручного написання системи правил та ручна візуалізація елементів у тривимірному просторі.

Організація 3D-тайлів та їх взаємодія між собою. Перший крок на етапі виконання практичного завдання - створення та рендер усіх необхідних тайлів, які в подальшому використовуватимуться алгоритмом під час генерації ігрового рівня. Для особливої наочності всі елементи будуть виконані в стилі воксельної графіки [4], що полегшить їх зв’язування між собою.

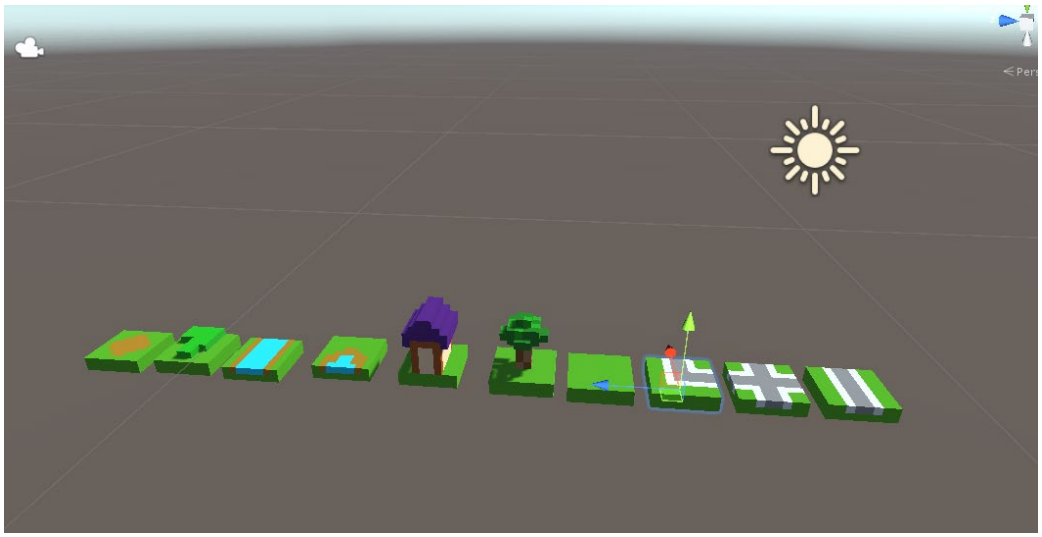

Рис. 3. Графічне представлення тайлів, з яких буде складатися зображення ігрової карти

Створюючи тайли до роботи з алгоритмом процедурної генерації потрібно враховувати їх важливу складову – зв’язуваність одного з іншими, як за кольорами, так і структурно, за рельєфом. У нашому випадку це означає, що межі тайлів, що стикаються, повинні збігатися по висоті і відтінку. Звіряючи ці значення, алгоритм перевіряє, чи можна встановити різні тайли поруч. За аналіз кольорів в алгоритмі відповідає метод *GetVoxelColor*. Функціонал інструментарію Unity дозволяє отримувати інформацію про піксель/полігон об'єкта через вбудований метод Raycast. Raycast - це деякий промінь, що випускається з певного об'єкта в

певному напрямку заданої довжини (або нескінченний) для визначення колізій (зіткнень) з об'єктами. У місті зіткнення (колізії) промінь повертає значення у вигляді коду потрібного нам кольору та значення true/false щодо наявності потрібної колізії. У процесі роботи алгоритму всі тайли "прострілюються" цими променями по всій висоті та ширині, після чого отримані значення колізії та кольору порівнюються між собою.

Враховуючи той факт, що на згенерованій локації не може бути однакова кількість різних тайлів (припустимо, кількість тайлів трави не може дорівнювати кількості тайлів з будинками), в алгоритм також була впроваджена система ймовірностей. Кожен тайл має власну "вагу". Чим більше значення ваги - тим більша ймовірність того, що при спірній ситуації алгоритм віддасть перевагу вагомішому тайлу. Завдяки цьому отриманий результат виходить досить структурованим та не перевантаженим зайвими елементами.

Практична реалізація алгоритму WFC. Алгоритм, який покладено у дану практичну реалізацію, складається з наступних кроків.

Крок 1. Створення сітки тайлів розміром 15 на 15 клітинок. Під час ініціації програми в координатах 0.0.0 тривимірного простору створюється готова заповнення область, що складається з клітин, у кожній з якої знаходяться всі тайли одночасно в стані суперпозиції.

Крок 2. Вибір стартового тайла. У випадковій позиції на карті вибирається випадковий тайл, через що клітина колапсує.

Крок 3. Вибір наступних клітин, що межують з стартовим тайлом. Встановивши стартовий тайл, алгоритм отримує потрібні вихідні дані для подальшої роботи. Знаючи колізію та набір відтінків окремих вокселів вихідного (та всіх наступних) тайлу, алгоритм може визначити, які зайві (не сумісні) варіанти під час обрання йому варто викинути. У ситуації, коли після звіряння всіх параметрів суперпозиції залишається кілька тайлів, алгоритм випадково вибирає один із них. Чим вищий параметр "ваги" у тайла, тим більший шанс, що той заповнить порожню клітку. Робота алгоритму завершується, коли кожна клітинка зі всієї сітки "схлопується". Якщо алгоритм потрапляє у безвихідну ситуацію, він перезапускається.

Аналіз роботи розробленої системи процедурної генерації. В результаті програмної реалізації алгоритму WFC з наведеними вище особливостями, на генерацію однієї (маленької за мірками сучасного геймдевелопменту) ділянки 15x5 тайлів витрачається близько 30 секунд. Витративши більше часу на оптимізацію алгоритму, можна досягти набагато вражаючих результатів. Виходячи з візуального зображення отриманої локації (Рис. 4) можна дійти висновку, що параметр ваги працює коректно.

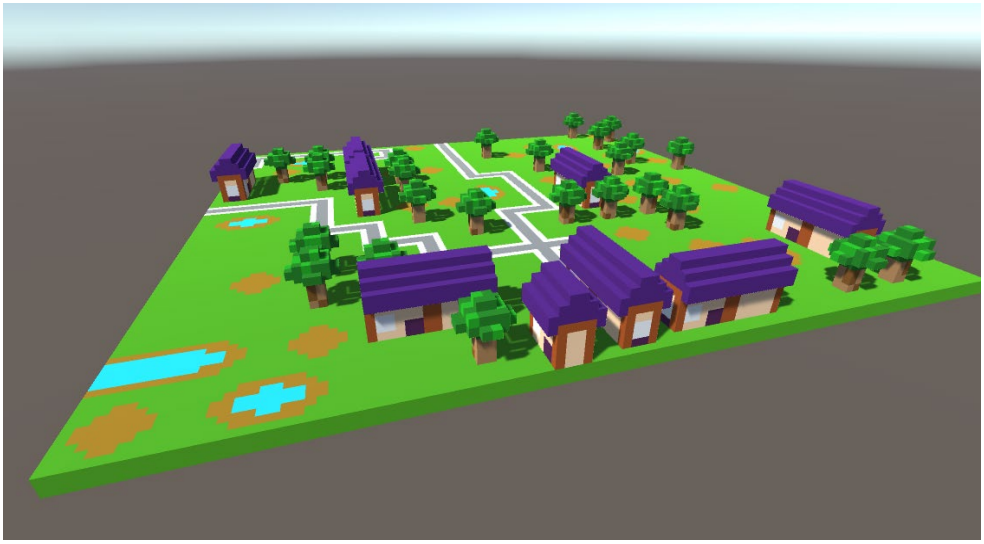

Рис. 4. Зображення ігрової локації, отриманої під час першої спроби

Тайл трави займає найбільшу кількість клітин, а тайли будинків та водойм – найменшу (Рис.4). За час роботи алгоритм робить в середньому від 120 до 170 ітерацій, що хоч і є досить великою кількістю, піддаючи процесор підвищеному навантаженню та витратам часу, але з іншої сторони, видає більш впевнений (якісний) результат для подальшого застосування у ігрі. Так само можна помітити, що один із тайлів (край водойми) не був використаний алгоритмом через відсутність симетрії. Обертаючи його в різні боки, алгоритм так і не знайшов клітини, в яку його можна було поставити. Цього можна було б уникнути, якби кількість вокселів на верхньому шарі була однаковою по обидва боки.

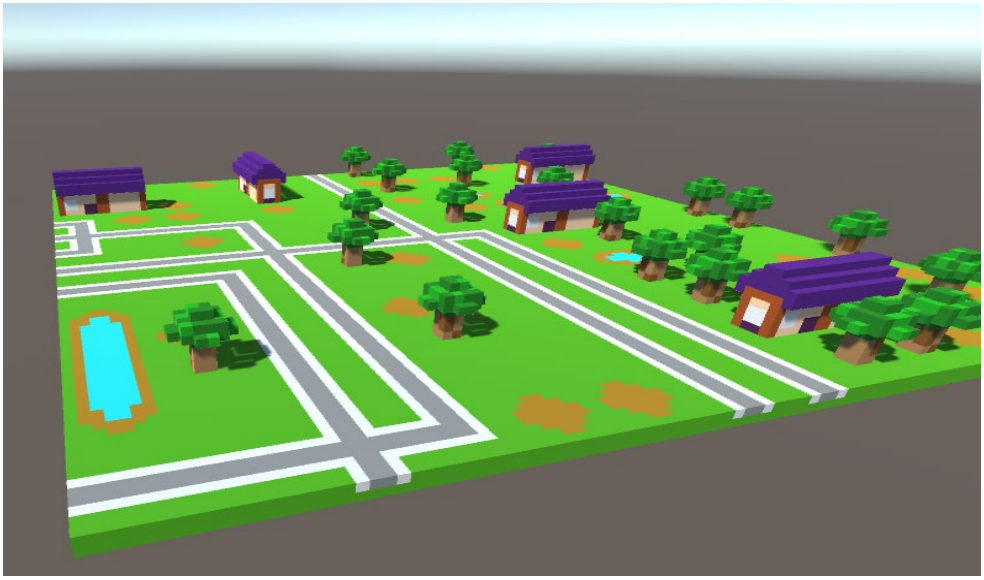

Рис. 5. Зображення ігрової локації, отриманої під час другої спроби

На рис. 5 та рис. 6 зображені ще дві спроби здійснити генерацію ігрового поля. На рис.6 в налаштуваннях правил для тайлів було накладено заборону на використання тайлів з доріжками.

Таким чином, розроблені ігрові локації вийшли придатними для використання під час ігор.

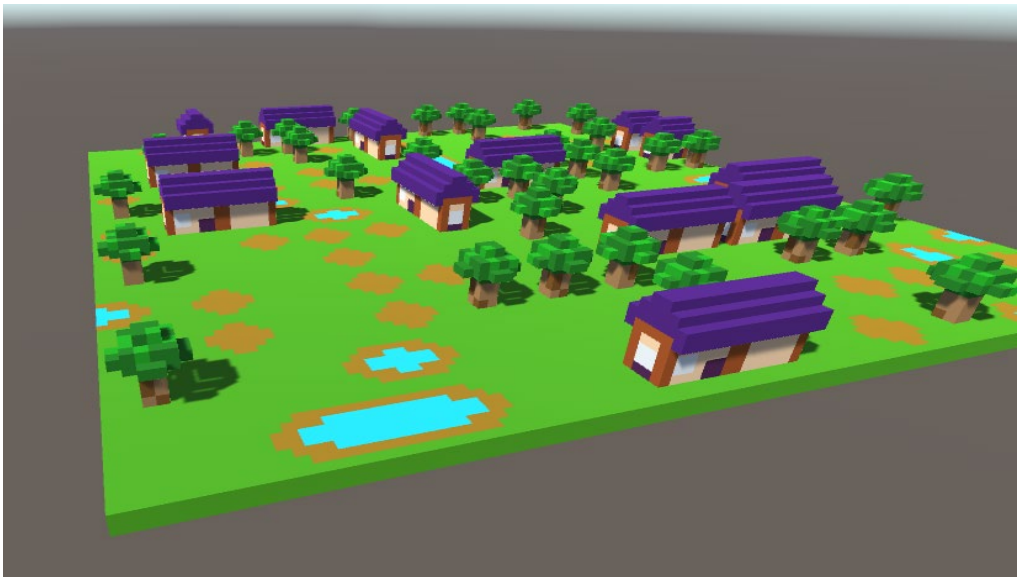

Рис. 6. Зображення ігрової локації, отриманої під час другої спроби

Висновки

В роботі проведено аналіз роботи алгоритму колапсу хвильової функції (WFC, Wave Function Collapse) та розроблено програмну систему процедурної генерації ігрового рівня в середовищі розробки Unity за допомогою мови C#. Завдяки використанню алгоритму WFC вдалося уникнути артефактів і помилок у згенерованих локаціях, а також досягти задовільної унікальності та структури розташування елементів ігрового світу. Практичні результати наочно демонструють те, як алгоритм, користуючись системою правил, здатний гармонійно розставляти та об'єднувати елементи. Даний алгоритм активно використовується в сучасному геймдевелопменті, підвищуючи ефективність підходу процедурної генерації та створюючи неймовірні світи з унікальним досвідом для кожного гравця.

Список використаної літератури:

1. Tanya X., Tarn Adams - Procedural Generation in Game Design: CRC Press.-2017.- 339p.
2. Ryan Watkins - Procedural Content Generation for Unity Game Development: Packt Publishing.-2016.-371p.
3. Noor Shaker, Julian Togelius, and Mark J. Nelson - Procedural Content Generation in Games: A Textbook and an Overview of Current Research.- Springer.- 2016.
4. Voxel (article in Wikipedia) [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Voxel (дата звернення 11.12.2023)

Автори статті

Ільїн Олег - доктор технічних наук, професор, Державний університет інформаційно-комунікаційних технологій

Іщеряков Сергій - кандидат технічних наук, доцент, Державний університет інформаційно-комунікаційних технологій, Київ, Україна

Карпов Артем - студент, Державний університет інформаційно-комунікаційних технологій

Єрмоленко Вадим - старший викладач, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

Authors of the article

Ilin Oleh - Doctor of Science (technic), Professor, State University of Information and Communication Technologies, Kyiv, Ukraine.

Ishcheryakov Serhiy - PhD, Associate Professor, State University of Information and Communication Technologies, Kyiv, Ukraine.

Yermolenko Vadim - senior lecturer, State University of Information and Communication Technologies, Kyiv, Ukraine.

Karpov Artem - student, State University of Information and Communication Technologies, Kyiv, Ukraine.