КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія Програмного Забезпечення

на тему:

ЗАСТОСУВАННЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ДЛЯ СТВОРЕННЯ ДВОВИМІРНИХ ІГРОВИХ ПРОСТОРІВ

Виконав студент 4-го курсу

Олег ГРИГОРОВИЧ

_____
(підпис)

Науковий керівник:

доцент

Євгеній ІВАНОВ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем.
«25» травня 2022 р.,
Протокол № 10
Завідувач кафедри

Олександр ПРОВОТАР

_____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 29 сторінок, 20 ілюстрацій, 13 джерел посилань.

ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ІГРОВІ РІВНІ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ В ІГРАХ, АЛГОРИТМ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ, МЕТОДИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ, КЛІТИННІ АВТОМАТИ, АСИНХРОННІ КЛІТИННІ АВТОМАТИ.

Об’єктом дослідження є процеси розроблення програмного забезпечення та побудови алгоритмів процедурної генерації ігрових 2D просторів. Предметом роботи є програмний засіб для генерації ігрових 2D просторів.

Метою роботи є розробка нового алгоритму генерації 2D просторів, що буде універсальним для створення різноманітних ігрових рівнів.

Методи розроблення: комп’ютерне моделювання, методи генерації 2D простору.

Інструменти розроблення: безкоштовний, вільно поширюваний ігровий рушій Godot Engine v3.4.4, мова програмування GDscript.

Результати роботи: виконано огляд наявних алгоритмів з процедурного створення 2D просторів та відмінностей в результатах їх роботи, проаналізовано переваги та недоліки використання наявних підходів до процедурної генерації рівнів, представлено новий метод генерації 2D просторів, що базується на використанні асинхронних клітинних автоматів, розроблено програмний продукт, який дозволяє наочно демонструвати процеси побудови двовимірного простору створеним алгоритмом.

Програмний продукт, що реалізовує алгоритм процедурної генерації на основі асинхронних клітинних автоматів, може застосовуватися при створенні ігрових 2D рівнів в ігровому програмному забезпеченні.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Правильно організована процедурна генерація – це головне джерело різноманітності та реіграбельності в сучасних комп’ютерних іграх. Досягнення цих цілей дозволяє розширити кількість контенту ігрових програмних засобів та можливих сценаріїв розвитку подій, що підвищує зацікавленість оператора-гравця. Одним з основних застосувань цього інструменту є генерація ігрових рівнів.

Підвищення варіативності, спрощення написання алгоритмів без втрати різноманітності, більша універсальність алгоритмів є ключовими прагненнями у цій галузі.

Розвиток програмного забезпечення, призначеного для створення комп’ютерних ігор (далі: ігрових рушіїв) призводить до спрощення реалізації ігрових програм, проте, зазвичай, ці покращення спрямовані на створення вмісту напряму́, а не опосередковано – через алгоритм. При цьому, генерація контенту комп’ютером значно зменшує затрати часу на розробку ігрового програмного забезпечення і допомагає змістити фокус розробника на інші аспекти, як-от розробка ігрових систем, а отже дозволяє зекономити як час, так і гроші.

Актуальність роботи та підстави для її виконання. У наявних алгоритмів для генерації рівнів є принципові відмінності, які чітко визначають переваги та недоліки використання цих алгоритмів.

Поєднання існуючих підходів з новітніми технологіями розробки, а також з менш поширеними інструментами, що можуть застосовуватись в генерації двовимірних просторів, забезпечує розширення можливої сфери застосування, збільшення ефективності алгоритму, підвищення різноманітності, можливість виконання складніших завдань, та, як наслідок, процедурне створення двовимірних ігрових просторів, частини яких більш взаємопов’язані порівняно з попередниками, тощо.

Використання асинхронних клітинних автоматів для розміщення контенту та частин ігрових рівнів є перспективним джерелом інновацій, оскільки сутності, які можуть застосовуватись для полегшення їх реалізації реалізовані у більшості сучасних ігрових рушіїв, проте використання їх особливостей для генерації розміщення об’єктів зустрічається вкрай рідко.

Мета й завдання роботи. Метою роботи є створення нового алгоритму генерації 2D просторів, що буде універсальним для створення різноманітних рівнів.

Для досягнення поставленої мети необхідно виконати такі завдання:

- Дослідити існуючі методи реалізації процедурної генерації 2D просторів, визначити їх переваги та недоліки.
- Визначити вимоги до алгоритму, що створюється.
- Розробити алгоритм генерації 2D просторів, що застосовує асинхронні клітинні автомати.
- Розробити прототип гри, що реалізовуватиме алгоритм та наглядно демонструватиме результати його роботи.

Об'єкт, методи й засоби розроблення. Об’єктом дослідження є процеси розроблення програмного забезпечення та побудови алгоритмів процедурної генерації ігрових 2D просторів.

Розробці програмного засобу передувало створення алгоритму. Основу для цього склав аналіз деяких типів процедурної генерації рівнів.

В якості інструменту створення програмного засобу було обрано Godot Engine v3.4.4 [1] – ігровий рушій, що використовує скриптову мову програмування GDscript.

Програмне забезпечення є безкоштовним, вільно поширюваним, з відкритим вихідним кодом.

Godot надає багатий набір класів об'єктів для абстрагування багатьох системних функцій, що використовуються при розробці ігрового програмного забезпечення. Ключовими рисами є об’єктно-орієнтований підхід розробки, фокус

на лаконічності програмного коду та можливість експорту продукту на основні платформи: Android, IOS, MacOS, Windows, Linux.

Можливі сфери застосування. Програмний продукт та розроблений алгоритм може використовуватись в ігрових програмах з процедурною генерацією ігрових 2D просторів для швидкої ітерації змін у параметрах при генерації 2D рівнів, простій реалізації більш складних просторів, легкого збільшення кількості та реалізації більш складних правил генерації.

РОЗДІЛ 1
ПРОЦЕДУРНА ГЕНЕРАЦІЯ У ІГРОВОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

1.1 Аналіз предметної області та обґрунтування актуальності

Процедурна генерація контенту – це використання формального алгоритму для створення ігрового вмісту, який, зазвичай, створює розробник чи дизайнер. Останнє десятиліття було сповнене дослідженнями на тему методів генерації великої різноманітності ігрових елементів: ігрових рівнів, зброї, текстур та ефектів, сюжету. Серед часто згадуваних цілей застосування процедурної генерації контенту:

- зменшення розміру файлів,
- представлення адаптованого та персоналізованого досвіду,
- збільшення кількості контенту,
- менший час розробки, у порівнянні зі створенням вручну.

У академічних колах дослідження процедурної генерації контенту часто виходять з підгалузі штучного ігрового інтелекту, де проблема позиціонується як створення штучного інтелекту для заміни або відтворення роботи чи творчих здібностей професійного дизайнера-людини. А також оцінювання штучним інтелектом якості створеного ним контенту та створення штучного інтелекту, який міг би взаємодіяти з людиною-дизайнером як партнером. Саме тому, незалежно від намірів розробника, кожна система процедурної генерації контенту неявно включає в себе теорію дизайну гри і елемента, що генерується.

Розглядаючи процедурну генерацію контенту як формалізацію процесу дизайну, ми можемо провести паралелі з іншими галузями, де вона застосовувалась і раніше – настільні ігри, ігри з картами і т.д.

За останні десятиліття спостерігається тенденція до різкого зростання середньої кількості людей, залучених до створення відеогри, а поруч з цим і значного зростання вартості розробки кожного проєкту [2]. Це обумовлює

фінансово-економічні підстави для використання процедурної генерації, окрім креативних.

Наявні дослідження у галузі в основному походять від спільноти дослідників ігор з технічної сторони [3]. Вони досліджують діапазон контенту, який можна створити та поширені алгоритми, які використовуються для створення процедурного вмісту. З цих робіт відомо, що широко використовуються генетичні алгоритми, псевдовипадкові генератори чисел, просторові алгоритми, моделювання та симуляція складних систем та штучний інтелект [3].

До контенту, що створюється процедурно, належать:

1) Неподільні елементи:
    - Текстури
    - Звуки
    - Будівлі
    - Поведінка
2) Ігрові простори:
    - Внутрішні простори
    - Зовнішні простори
    - Водойми
3) Ігрові системи:
    - Поведінка сутностей
    - Мережі доріг
    - Екосистеми
4) Ігрові сценарії:
    - Пазли
    - Сюжети
    - Сформовані рівні
5) Ігровий дизайн:
    - Дизайн систем
    - Дизайн ігрового світу

6) Ігрові сутності:
    - Ігрові інструменти (зброя, ресурси, тощо)
    - Неігрові персонажі та їх поведінка
    - Ігрові персонажі та їх можливості

Варто зазначити, що процедурна генерація не обов'язково має бути стохастичною: вона може як включати в себе елемент випадковості, так і бути детермінованою [3], [4]. Ця робота фокусується виключно на процедурній генерації, яка включає в себе елемент випадковості.

Випадковість є ознакою формування процесуального контенту. Хоча існують ігри, що включають детерміновану генерацію, більшість включає в себе випадкове прийняття рішень, навіть якщо детермінований характер походить від утримання випадкових ідентифікаторів постійними або керованими. Видатний ігровий дизайнер Ендрю Дулл навіть включає випадковість у своє визначення процедурної генерації контенту [4].

1.2 Огляд наявних алгоритмів та постановка завдання

У цій роботі розглядається використання процедурної генерації для створення внутрішніх та зовнішніх ігрових 2D-просторів, а також водойм, з використанням випадковості. Це – одні з класичних завдань процедурної генерації [5]. Основними проблемами при цьому є контроль над процесом генерації, забезпечення різноманітності просторів, а також зменшення складності та часу, необхідних для програмної реалізації системи.

У цьому розділі ми будемо оперувати підходами, а не конкретними алгоритмами, оскільки реалізації за визначенням є пристосованими до конкретних ситуацій та програмного забезпечення, а заглиблення у деталі є надлишковим у даній роботі, оскільки воно вже наявне у інших публікаціях [6].

1) Розділення простору на частини – доволі випадковий метод генерації, ідея – розділяємо простір на зони, після чого створюємо кімнату у кожній зоні та

з’єднуємо їх (рис.1.1.). Недоліками є відносна повторюваність результату та не універсальна програмна реалізація. Підхід широко застосовується в ігровому програмному забезпеченні протягом останніх 10 років.

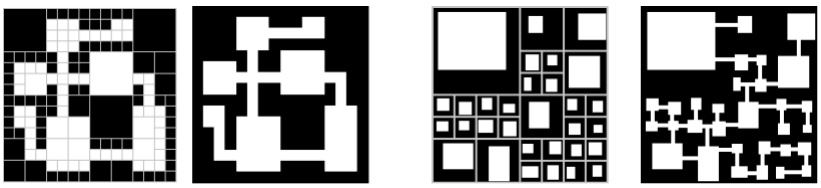

Рисунок 1.1 – Приклад двох рівнів, згенерованих розділом простору на частини

2) Збільшення простору за допомогою посередників – метод, який зазвичай базується на наявності посередника, що і прокладає сполучення, і створює кімнати (рис 1.2.). Проблема такої реалізації – непередбачуваність поведінки штучного інтелекту, що вимагає значних затрат часу на визначення правил та їх реалізацію. Протягом останніх десяти років використання є доволі обмеженим.

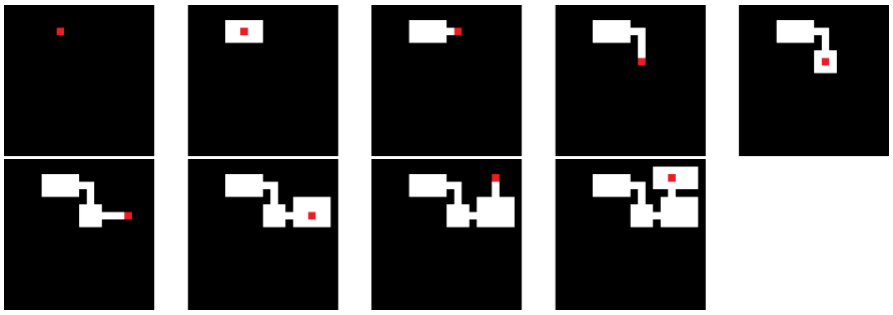

Рисунок 1.2 – Покрокове збільшення ігрового простору посередником формує рівень

3) Використання клітинних автоматів (рис.1.3.) – надзвичайно універсальний інструмент, доведено, що деякі типи є повними за Тюрінгом [7]. За допомогою них можна реалізувати навіть умовно нескінченні простори [8]. Хоча невелика кількість параметрів є доволі інтуїтивною, основним

недоліком є те, що зміна кожного параметра впливає на низку особливостей рівня і цей вплив доволі важко зрозуміти. Це створює труднощі як для дизайнера, так і для програміста. Окрім цього, неможливо досягнути результату, що мав би специфічні параметри, а отже особливості гри доволі мало пов’язані з цими параметрами, що доволі часто суперечить цілям дизайну. Використання в окремих жанрах ігрового програмного забезпечення є доволі поширеним.

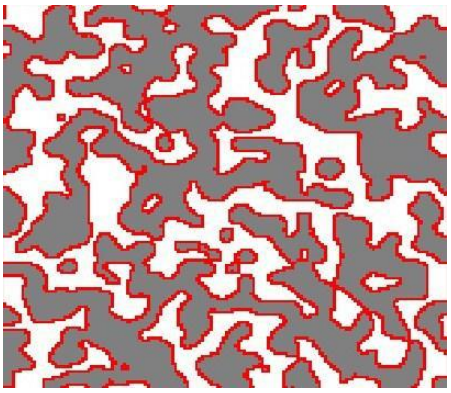

Рисунок 1.3 – Рівень зі складною структурою, створений з використанням клітинних автоматів

4) Генерація, що базується на граматиці – використання структур, що базуються на граматиці, разом з тим описуючи, як генерувати нові такі структури. Основний недолік – правила є абсолютно негнучкими (англ. hard-coded), а отже специфічними для кожної задачі. Не зважаючи на цю проблему, підхід дозволяє добре вирішувати окремі задачі, такі як планування складності [9].
5) Розширені (синтетичні) методи генерації. Є поєднанням попередніх пунктів, часто зі сторонніми ідеями чи засобами. Результати, а отже й недоліки та переваги значно відрізняються залежно від компонентів, що поєднуються, проте є одними з найбільш перспективних поруч з останніми розробками у сфері граматичної генерації [6]. При реалізації мною курсової роботи було розроблено один з таких алгоритмів, основним недоліком якого є відносна простота згенерованих просторів

6) Застосування малопоширених інструментів. Оскільки галузь процедурної генерації є відносно новою (перші комерційні проекти з сильно обмеженими можливостями було розроблено у 1980х роках) і не надто добре дослідженою, зараз існує доволі великий спектр інструментів, що не використовуються у процедурній генерації рівнів, або ж застосовуються непропорційно мало, відносно до числа задач, що вони можуть вирішувати. Одним з таких інструментів є й використання **асинхронних клітинних автоматів – клітинних автоматів з можливістю змін різних клітин незалежно одна від одної (стан, час)**, що дозволяє знівелювати частину недоліків застосування звичайних клітинних автоматів в окремих випадках, проте рішення про застосування асинхронних чи звичайних клітинних автоматів на даний момент розглядається в залежності від правил, для кожного випадку окремо [10], та, не зважаючи на це, вже доведено, що у асинхронних клітинних автоматів є вагомі переваги у окремих сферах застосування [11][12].

Проведений аналіз дозволяє зробити висновок, що використання асинхронних клітинних автоматів є потенційно перспективним для досягнення поставленої мети та визначити характеристики розроблюваного алгоритму:

1) контрольованість результату;
2) простота;
3) універсальність;
4) відносна різноманітність результату;
5) простота модифікації правил.

РОЗДІЛ 2
СТВОРЕННЯ АЛГОРИТМУ, ЩО ЗАСТОСОВУЄ АСИНХРОННІ КЛІТИННІ АВТОМАТИ, З РЕАЛІЗАЦІЄЮ ЗАДАНИХ ПАРАМЕТРІВ

Запропонований алгоритм процедурної генерації простору з використанням асинхронних клітинних автоматів (далі: алгоритм) включає три фрагменти:

1) створення початкового стану – наповнення репрезентаційної моделі;
2) використання методів, що реалізують правила переходів клітин;
3) використання методів для створення фактичної геометрії простору на основі репрезентаційної моделі.

Старіші методи були сфокусовані більше на тому, *що* вони генерували, проте зараз фокус перейшов на те, *як* вони це роблять [6]. Це зрозуміло, адже наявність значущих параметрів для контролю створюваного продукту є необхідною умовою при ускладненні алгоритмів, щоб вони не були занадто стохастичними.

Для досягнення контрольованості й будемо впроваджувати ці параметри. Оскільки важливими аспектами з точки зору дизайну є кількість часу, необхідна для завершення рівня гравцем та різноманітність результату, такими параметри будуть:

1) можливість регулювати горизонтальний та вертикальний розмір простору
2) величина клітин
3) регулювання величин, що впливають на правила (кількість сусідів певного типу, що необхідна для зміни стану
4) мінімальний розмір порожнин, що об’єднуються.
5) Набір сутностей, що реалізовується клітинами

Аби досягнути різноманітності рельєфу використаємо псевдовипадковий генератор чисел. Надалі розглядатимемо завдання алгоритму як побудову порожнин (печер) на просторі, що є одним з завдань, оскільки результати

виконання легко оцінити. Параметр розміру ігрових клітин є необхідним для збільшення універсальності алгоритму.

Завдяки тому, що алгоритм контролюється всього кількома змінними, а також результат роботи представлений у вигляді фізичних об’єктів – він є простим, універсальним та легким для розуміння.

Створимо сутність, що відповідає асинхронному клітинному алгоритму, на основі кінцевого стану якого будемо генерувати простір.

Після цього створимо початковий стан, за допомогою псевдовипадкового генератора чисел, який відповідає заданим вхідним параметрам: заповнимо простір однотипними клітинами (рис. 2.1) та добавимо випадкові клітини, що відповідають іншій сутності (рис. 2.2)

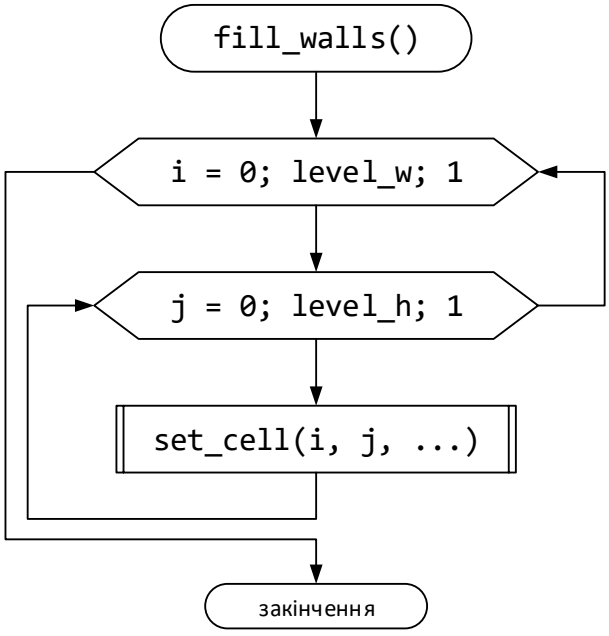

Рисунок 2.1 – Блок-схема початкового заповнення простору

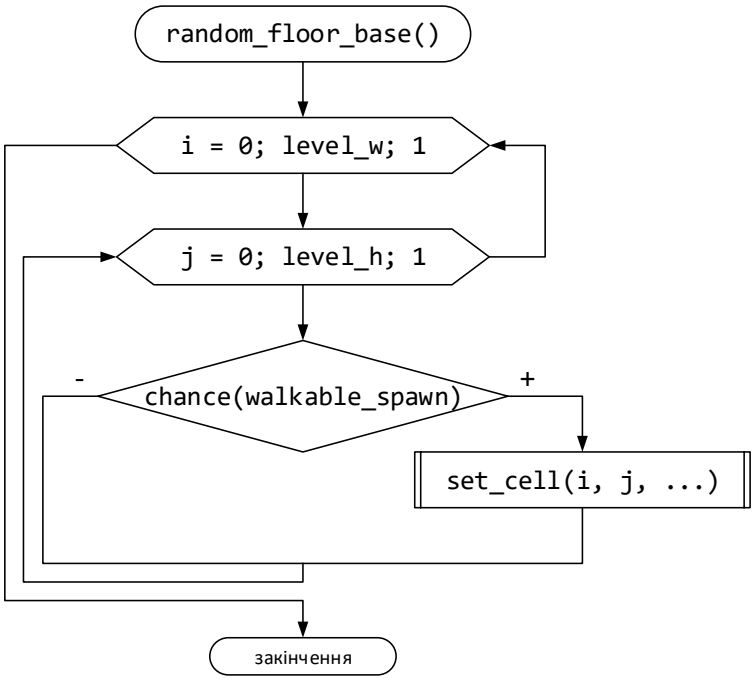

Рисунок 2.2 – Блок-схема створення початкового стану автомата

Ітеративно проведемо вказану параметрами кількість застосування правил (рис 2.3.)

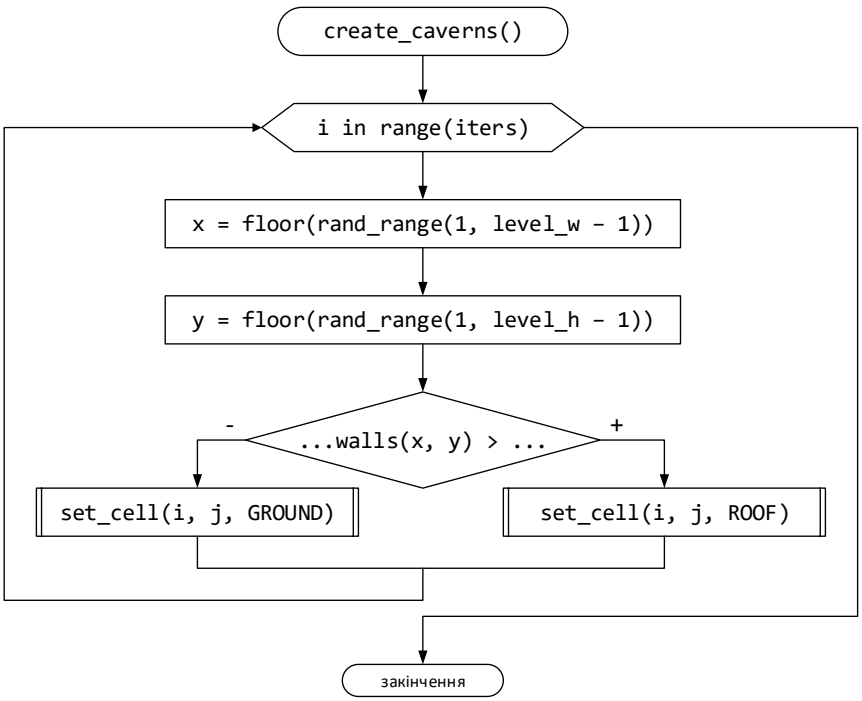

Рисунок 2.3 – Блок-схема алгоритму застосування правил

На даному етапі створено представлення простору. Для того, аби з’єднати печери у межах моделі, використовується алгоритм з’єднання порожнин, зображений на рис. 2.4.

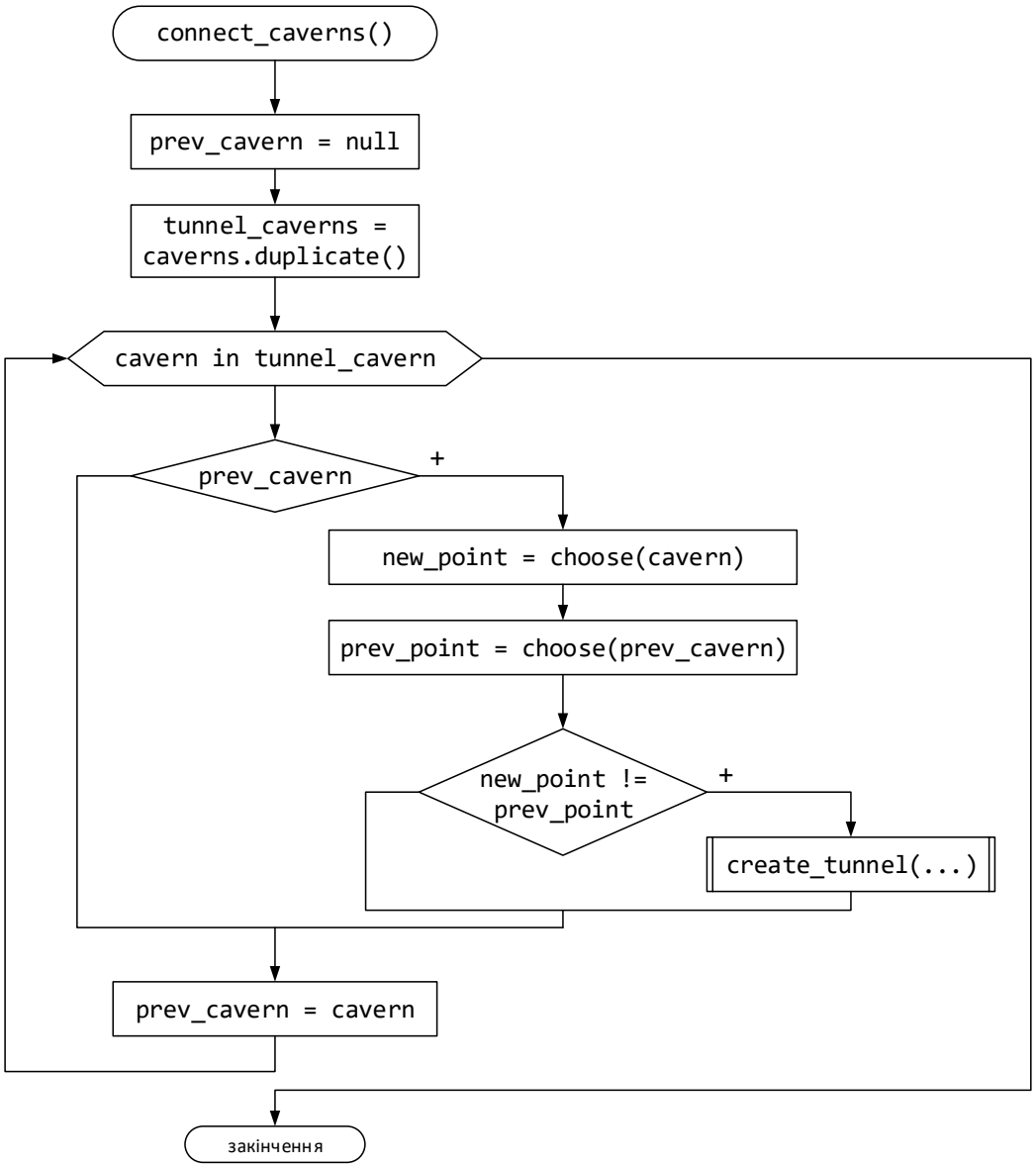

Рисунок 2.4 – Блок-схема алгоритму з’єднання порожнин

Наступним кроком є заповнення всієї ділянки, яка визначає та входить до меж згенерованого простору, сутностями, які відповідають станам цих клітин (рис 2.5).

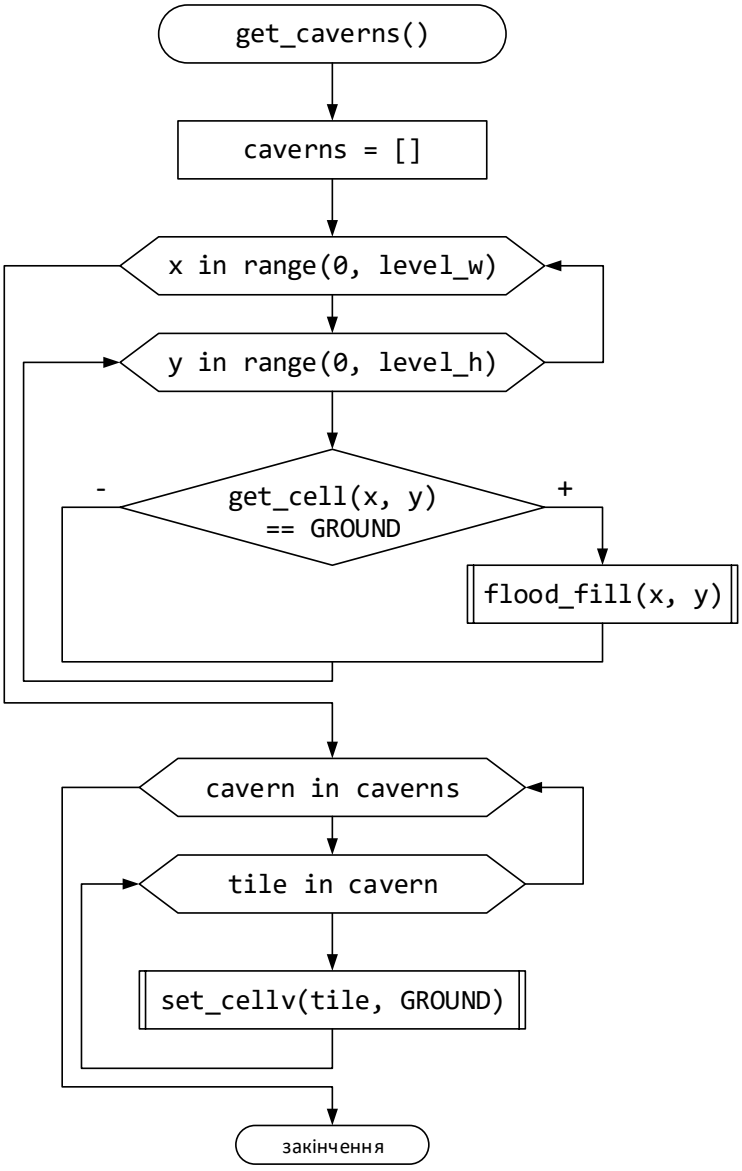

Рисунок 2.5 – Блок-схема алгоритму перетворення репрезентативної моделі в двовимірний простір

Використовуючи наведені вище міркування побудуємо алгоритм генерації 2D просторів, що застосовує асинхронні клітинні автомати.

Алгоритм 2.1 Генерація 2D просторів.

Початок алгоритму

1. Ввід параметрів: горизонтальний та вертикальний розмір простору, кількість сусідів для перетворення, розмір клітин, мінімальний розмір ділянок для приєднання, кількість стартових клітин, кількість ітерацій.

2. Заповнення простору однотипними клітинами.

3. Встановлення початкових точок.

4. Ітераційне(асинхронне) розширення печер на основі початкових точок.

5. З'єднання печер деталізованими тунелями.

Кінець алгоритму.

Блок схему програмної реалізації алгоритму зображено на рис. 2.6.

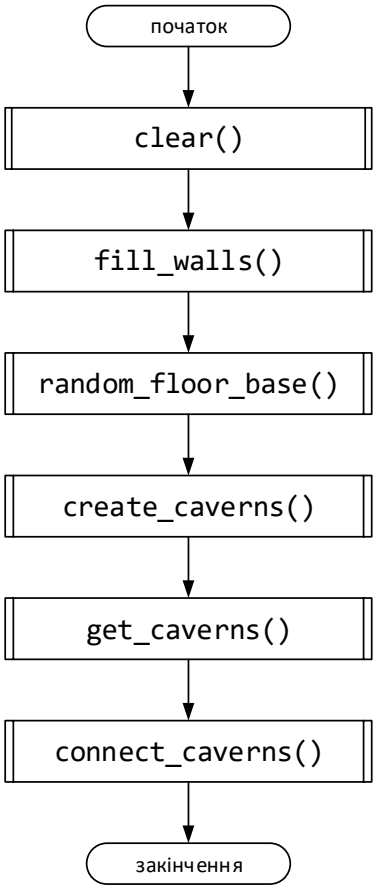

Рисунок 2.6 – Блок-схема розробленого алгоритму генерації двовимірних просторів

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОТОТИПУ, ЩО ВИКОРИСТОВУЄ РОЗРОБЛЕНИЙ АЛГОРИТМ

3.1. Програмна реалізація створеного алгоритму

Реалізуємо простий прототип ігрового програмного забезпечення, що міститиме реалізацію алгоритму. У описі навмисно упущено деякі технічні деталі реалізації для зосередження на функціоналі та реалізації, а не структурних особливостях Godot та GDscript.

Створюємо клас, який керуватиме програмою, та міститиме функцію для реагування на дії оператора (рис.3.1).

```
func _input(event):
    if event.is_action_pressed('ui_select'):
        $Caves.generate()
        $Caves.redraw()
        #draw_caves()
    if event.is_action_pressed('ui_cancel'):
        var player = Player.instance()
        add_child(player)
    #    player.position =
        play_mode = true
```

Рисунок 3.1 – Функція реагування на дії оператора

Створюємо основний вузол, у якому будемо реалізовувати алгоритм з наступною структурою(рис.3.2.).

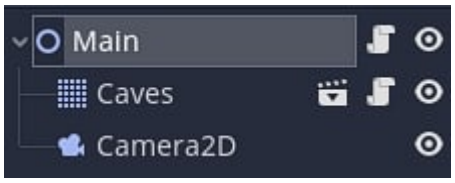

Рисунок 3.2 – Створення основного вузла

Тут міститься структура, що зберігатиме початковий стан автомата та генеруватиме простір на основі репрезентаційної моделі, і вузол камери та керуючий вузол, створений раніше. Після цього створюємо скрипт, який і буде відповідати за процедурну генерацію. Спочатку розміщені змінні, що регулюють поведінку алгоритму (рис.3.3.).

```
export(int)   var level_w         = 80
export(int)   var level_h         = 50
export(int)   var iters    = 20000
export(int)   var neighborhood     = 4
export(int)   var walkeable_spawn = 48
export(int)   var minimal_cave_size = 80
enum Tiles { GROUND, ROOF, WATER, TREE }
```

Рисунок 3.3 – Змінні, що регулюють поведінку алгоритму

Після цього функція, що відповідає за процес генерації: створення репрезентативної моделі, початкового стану автомата, зміну станів та перетворення кінцевого стану у фізичний простір, яка викликається при запуску програми та оператором (рис.3.4):

```
func generate():
    clear()
    fill_walls()
    random_floor_base()
    create_caverns()
    get_caves()
    connect_caves()
```

Рисунок 3.4 – Функція, що відповідає за процес генерації

Також створено допоміжні функції, що відповідають за перетворення репрезентативної моделі та з’єднання порожнин (рис. 3.5).

```
func flood_fill(tilex, tiley):
    # flood fill the separate regions of the level, discard
    # the regions that are smaller than a minimum size, and
    # create a reference for the rest.

    var cavern = []
    var to_fill = [Vector2(tilex, tiley)]
    while to_fill:
        var tile = to_fill.pop_back()

        if !cavern.has(tile):
            cavern.append(tile)
            set_cellv(tile, Tiles.ROOF)

            #check adjacent cells
            var north = Vector2(tile.x, tile.y-1)
            var south = Vector2(tile.x, tile.y+1)
            var east  = Vector2(tile.x+1, tile.y)
            var west  = Vector2(tile.x-1, tile.y)

            for dir in [north,south,east,west]:
                if get_cellv(dir) == Tiles.GROUND:
                    if !to_fill.has(dir) and !cavern.has(dir):
                        to_fill.append(dir)

    if cavern.size() >= minimal_cave_size:
        caverns.append(cavern)
```

Рисунок 3.5 – Функція, що відповідає за заповнення печер у фізичному просторі

Для з’єднання порожнин, що наявні у стані автомата, який застосовується для побудови простору, використовуються елементи розширення простору посередником (див. розділ 1.2). Якщо дві порожнини мають розмір, більший за значення, визначене параметрами, то між ними прокладається шлях сутністю зі стохастичною поведінкою, що має більшу ймовірність руху до цілі, ніж від неї, а також цей шлях додатково розширюється для досягнення більшої схожості з навколишнім рельєфом.

Для наглядності результату роботи алгоритму реалізовано три правила, за якими відбувається зміна стану:

1) кількість сусідів певного типу більша за мінімальну межу;
2) кількість сусідів, що не належать певному типу, більша за межу;
3) кількість сусідів вказаних типів рівна певному значенню (рис.3.6).

```
func check_nearby_ground(x, y):
    var count = 0
    if get_cell(x, y-1)   == Tiles.GROUND:  count += 1
    if get_cell(x, y+1)   == Tiles.GROUND:  count += 1
    if get_cell(x-1, y)   == Tiles.GROUND:  count += 1
    if get_cell(x+1, y)   == Tiles.GROUND:  count += 1
    if get_cell(x+1, y+1) == Tiles.GROUND:  count += 1
    if get_cell(x+1, y-1) == Tiles.GROUND:  count += 1
    if get_cell(x-1, y+1) == Tiles.GROUND:  count += 1
    if get_cell(x-1, y-1) == Tiles.GROUND:  count += 1
    return count
```

Рисунок 3.6 – Функція для визначення кількості сусідів певного типу

Після цього, було реалізовано персонажа гравця та реалізовану схему керування ним для переміщення згенерованими за допомогою алгоритма просторами.

3.2. Візуалізація роботи створеного прототипу

Прототип створений за допомогою ігрового рушія Godot Engine.

Введені значення параметрів алгоритму зображено на рис. 3.3.

Після запуску прототипу відбувається генерація репрезентативної моделі та її перетворення у простір, з подальшим відображенням останнього. Спершу наведено простий приклад генерації з двома сутностями – жовтими клітинами, якими може переміщатись ігровий персонаж, та зеленими, якими він переміщатись не може, без розширення прокладених посередником тунелів (рис. 3.7).

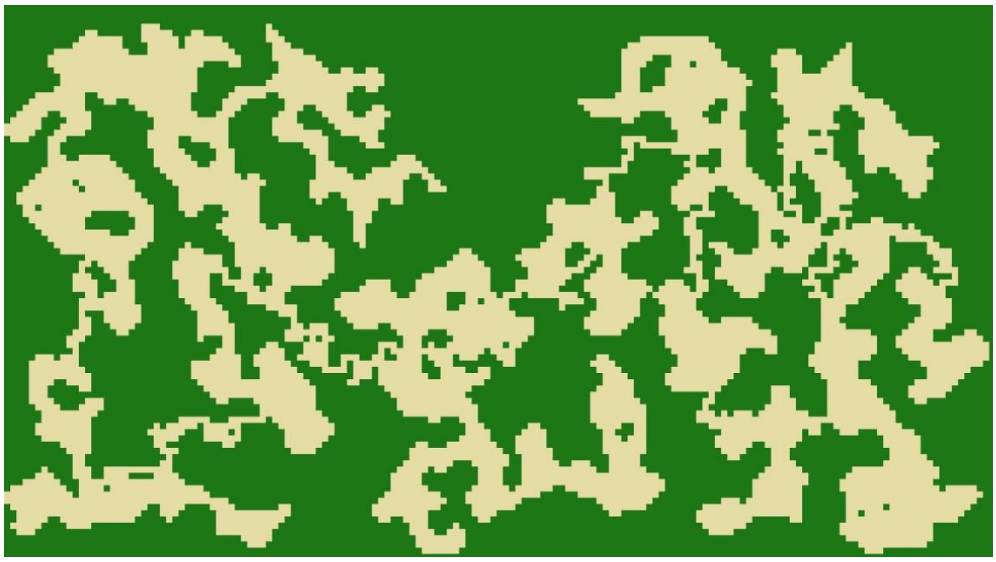

Рис.3.7. Простір, отриманий в результаті генерації, з використанням двох простих сутностей та без розширення тунелів.

Наявність артефактів (неприродні частини рельєфу, які не відповідають своєму оточенню) демонструє, що тунелі варто розширювати (рис.3.8.).

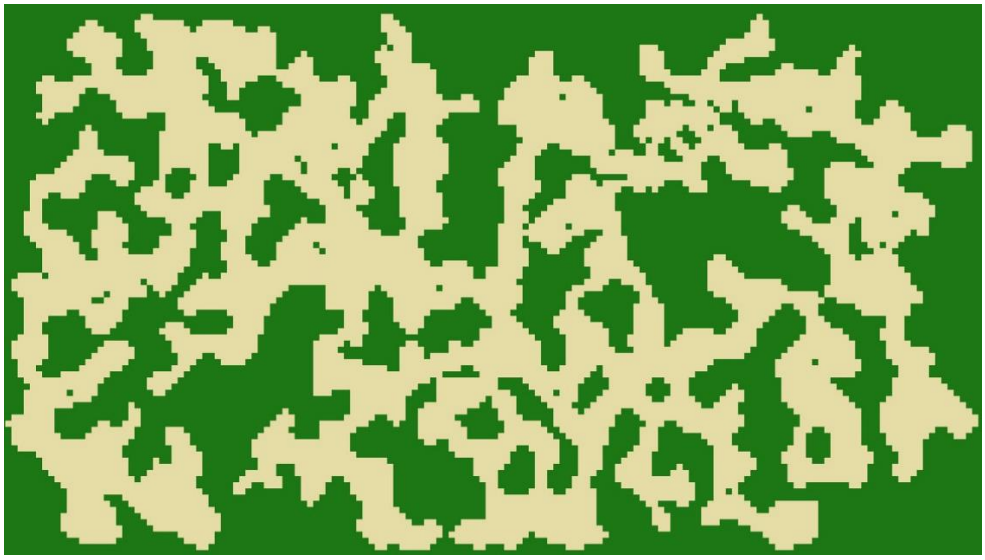

Рис.3.8. Простір, згенерований з тими ж параметрами, та позбавлений артефактів за допомогою розширення тунелів.

Для демонстрації універсальності алгоритму слід також розглянути результати генерації з іншими параметрами, а саме розміром простору, сутностями інших типів іншими значеннями, необхідними для зміни станів (рис.3.9).

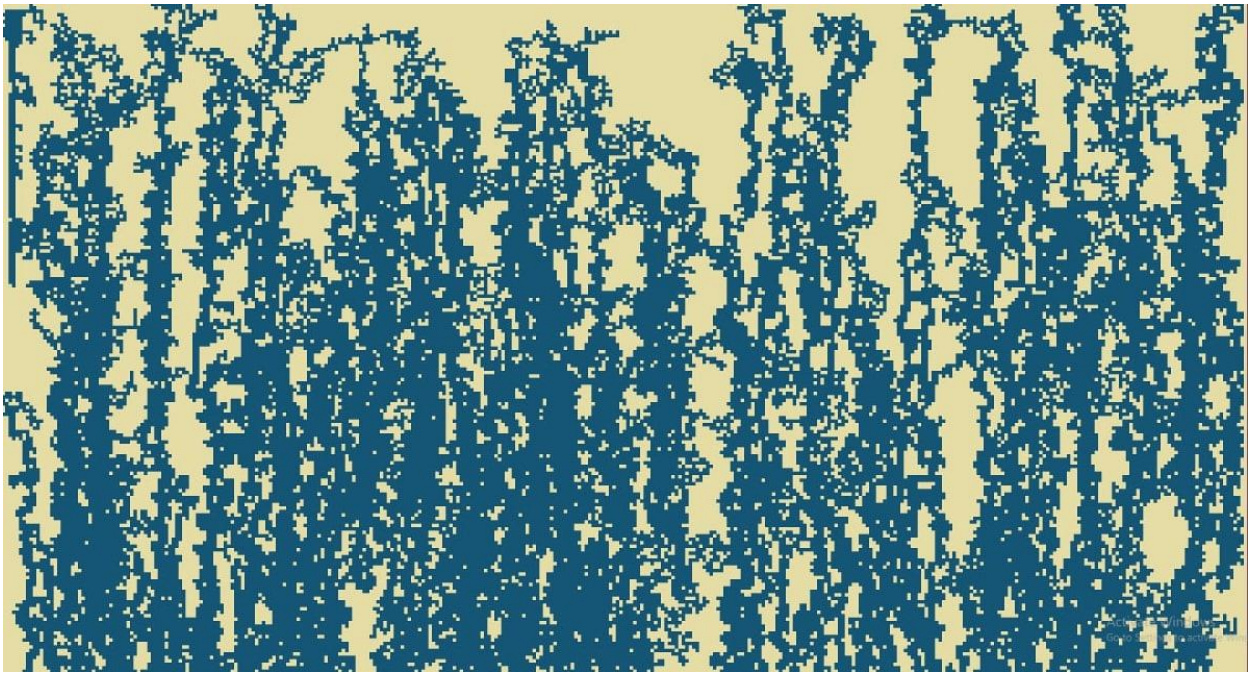

Рисунок 3.9 – Фрагмент значно більшого простору з новими параметрами та іншими сутностями. Сині клітини сповільнюють рух персонажа-гравця.

Слід зауважити, що генерація просторів з двома сутностями є доволі тривіальною задачею, проте залишається широко поширеною. Для демонстрації універсальності алгоритму нижче наведено також результати його роботи з застосуванням більшого набору правил та трьома сутностями (рис.3.10.).

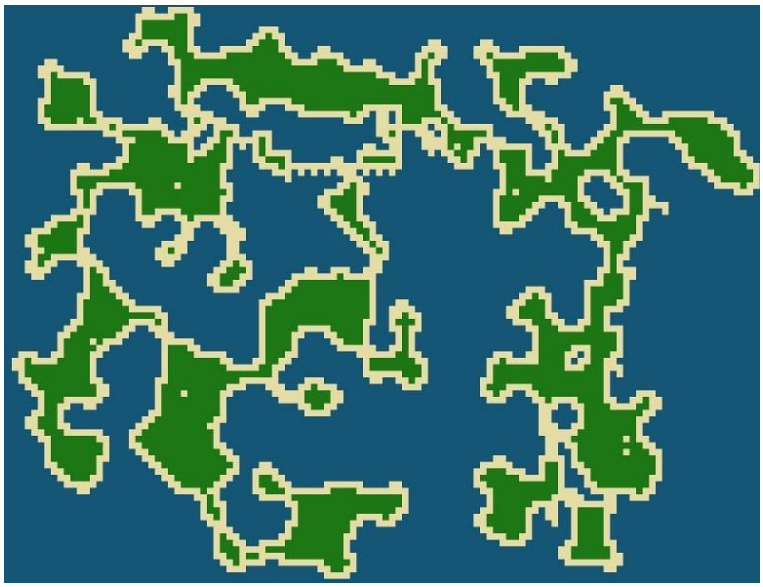

Рисунок 3.10 – Результат генерації з трьома сутностями та більшим набором правил

Тут сутності мають інші властивості, ніж раніше. Зелені клітини дещо сповільнюють переміщення гравця, жовті – дозволяють вільно рухатись, синіми пересуватись не можна.

Також, при натисканні відповідної клавіші, оператор може перейти в ігровий режим, що дозволяє йому керувати ігровим персонажем, який має можливість переміщуватись створеними просторами.

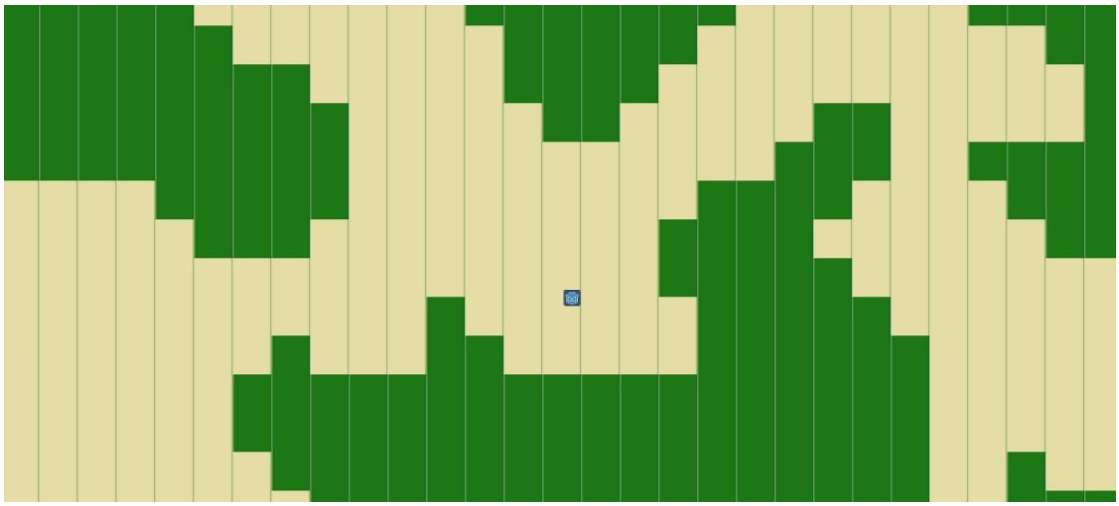

Рисунок 3.11 – Персонаж, контрольований оператором, переміщується ігровим простором, зображеним на рис.3.8.

Створений прототип доступний для завантаження за адресою [13].

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи розв’язано задачу програмної реалізації алгоритму процедурної генерації ігрових 2D просторів з застосуванням асинхронних клітинних автоматів. При цьому отримано наступні результати.

1. Досліджено існуючі методи реалізації процедурної генерації 2D просторів, що дозволило визначити їх переваги та недоліки.
2. Визначено вимоги для створюваного алгоритму, які було використано при розробці алгоритму.
3. Розроблено алгоритм генерації 2D просторів, який застосовує асинхронні клітинні автомати, що дозволило розробити прототип гри.
4. Розроблено прототип гри, що реалізує алгоритм та наглядно демонструє результати його роботи.
5. Використання асинхронних клітинних автоматів за умови малої кількості правил генерує простори, структурно схожі до тих, що згенеровані з використанням звичайних клітинних автоматів, проте має ряд суттєвих переваг з точки зору розробника.

Створений алгоритм генерації ігрових 2D просторів володіє такими перевагами порівняно з існуючими алгоритмами:

1. Універсальність – може бути використаний для генерації просторів у низці ігор, зокрема тих, що належать до жанру roguelike.
2. Інтуїтивність – простота контролю результату та настроювання роботи алгоритму.
3. Підконтрольність набору правил разом з широкою варіативність результату.
4. Простота розширення – алгоритм добре працює зі збільшенням кількості правил та сутностей, а їх доповнення є простим.

З огляду на свої переваги, створений алгоритм може застосовуватись у низці ігрового програмного забезпечення різних жанрів, за умови застосування ними процедурної генерації. Особливо доцільним є застосування у ігровому програмному забезпеченні жанру Roguelike.

Створений прототип доступний для завантаження за адресою [13].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Godot Engine [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org/.
2. Takatsuki Y. Cost headache for game developers [Електронний ресурс] / Y. Takatsuki. – 2007. – Режим доступу до ресурсу: http://news.bbc.co.uk/2/hi/business/7151961.stm.
3. Procedural Content Generation for Games: A Survey / M.Hendrikx, S. Meijer, J. Van Der Velden, A. Iosup. // ACM Transactions on Multimedia Computing Communications and Applications. – 2013.
4. Smith G. Understanding procedural content generation: A design-centric analysis of the role of PCG in games [Електронний ресурс] / Gillian Smith. – 2014. – Режим доступу до ресурсу: https://www.researchgate.net/publication/266655606_Understanding_procedural_content_generation_A_design-centric_analysis_of_the_role_of_PCG_in_games.
5. Moss R. 7 uses of procedural generation that all developers should study [Електронний ресурс] / Richard Moss – Режим доступу до ресурсу: https://www.gamasutra.com/view/news/262869/7_uses_of_procedural_generation_that_all_developers_should_study.php.
6. Shaker N. Procedural Content Generation in Games: A Textbook and an Overview of Current Research / N. Shaker, J. Togelius, M. Nelson., 2016. – 224 с.
7. Wolfram S. Cellular Automata and Complexity: Collected Papers / Wolfram., 1994
8. Johnson, L., Yannakakis, G.N., Togelius, J.: Cellular automata for real-time generation of infinite cave levels. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, 2010.
9. Mircea Traichioiu, Sander Bakkes, Djederik Roijers – grammar based Procedural Content Generation from Designer-provided Difficulty Curves., 2015
10. Buvel, R.L. and Ingerson, Thomas E. Structure in asynchronous cellular automata. Physica D, 1:59–68, 1984.

11. Tatsuya Yamashita, Teijiro Isokawa, Ferdinand Peper, Ibuki Kawamata, and Masami Hagiya. Turing-completeness of asynchronous non-camouflage cellular automata. За ред. Alberto Dennunzio, Enrico Formenti, Luca Manzoni, та Antonio E. Porreca,, Processings of automata 2017, випуск 10248 Лекційних нотаток з Комп'ютерних наук, 187–199 с., Springer, 2017.
12. Thomas Worsch. Towards intrinsically universal asynchronous CA. Natural Computing, 12(4):539–550 с., 2013.
13. Посилання на розроблений прототип [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/sozzo/Diploma_project.