УДК 519.854.2

Міністерство освіти та науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

ГЮІК 501600.005 ПЗ
(позначення документа)

Дослідження та оптизація методів генерації рівнів в ігрових додатках
(тема)

Виконав:
студент *2* курсу, групи *СПРм-19-1*
Королюк В.В.
(прізвище, ініціали)
спеціальності *122 – Комп'ютерні науки*
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)
Керівник *доц. Петрова Р.В.*
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри системотехніки _____ *Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____
2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Королюку Віталію Валентиновичу*
(прізвище, ім'я, по батькові)

1. Тема роботиДослідження та оптимізація методів генерації рівнів в ігрових додатках
затверджена наказом по університету від 02.11 2020 р. № 1516СТ
2. Термін подання студентом роботи до екзаменаційної комісії 18 грудня 2020р
3. Вихідні дані до роботи: Перелік використовуваних програмних засобів: ОС Windows 10, Unity 3D. Технічне забезпечення: комп'ютер зі встановленим Unity3D.
4. Перелік питань, що потрібно опрацювати в роботі 4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) 4.1 Вступ. 4.2 Аналіз предметної області. 4.2.1 Процес розробки ігрових додатків. 4.2.2 Історія індустрії відеоігор. 4.2.3 Види ігрових додатків. 4.2.4 Постановка задачі. 4.3 Дослідження існуючих методів генерації рівнів. 4.3.1 Генерація початкової геометрії "BSP". 4.3.2 Генерація рівнів за допомогою підготовлених частин. 4.4 Розробка модифікованого методу генерації рівнів. 4.4.1 Пул об'єктів. 4.4.2 Вибір інструмента реалізації. 4.4.3 Реалізація модифікованого методу. 4.5 Висновок.

5.Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 2.9 Фінальний результат алгоритму генерації початкової геометрії "BSP"3.2 Згенерований рівень 3.2 Пул об'єктів на сцені Unity 3.3 Ініціалізація Пула об'єктів 3.4 Object Pooler під час генерації рівня 3.5 Генерація рівня з використанням Object Pooler

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз вихідних даних та літератури за темою атестаційної роботи	08.11.2020	Виконано
2	Системний аналіз предметної області	11.11.2020	Виконано
3	Постановка мети та задач дослідження	15.11.2020	Виконано
5	Розробка програмного засобу	03.12.2020	Виконано
6	Оформлення пояснювальної записки та презентації	20.12.2020	Виконано
7	Подання атестаційної роботи до екзаменаційної комісії	23.12.2020	Виконано

Дата видачі завдання 02 листопада 2020 р.

Студент _____ Королюк В.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Петрова Р.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до магістерської атестаційної роботи: 68 сторінок, 40 рисунків, 2 додатки, 15 джерел. Графічна частина атестаційної роботи містить 6 плакатів.

UNITY 3D, ІГРОВІ ДОДАТКИ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ РІВНІВ, ПУЛ ОБ’ЄКТІВ

Об'єкт дослідження – процеси автоматичної генерації рівнів в ігрових додатках.

Предмет дослідження – методи та засоби автоматичної генерації рівнів в ігрових додатках.

Мета роботи – дослідити та розробити модифікований метод автоматичної генерації рівнів в ігрових додатках.

Методи дослідження – аналіз існуючих методів автоматичної генерації рівнів. Визначено підхід до вирішення проблеми автоматичної генерації рівнів в ігрових додатках. Виконано порівняння існуючих алгоритмів генерації, проведено порівняльний аналіз їх показників, виділені найефективніші алгоритми. Запропоновано модифікацію алгоритмів генерації яка задовольняє поставленій задачі.

Галузь застосування – програмний засіб використовується для автоматичної генерації рівнів в ігрових додатках.

ABSTACT

Master's thesis contains: 68 pages, 40 images, 2 applications, 15 sources.

Graphic part of the thesis contains 6 posters.

UNITY 3D, GAMEDEV, PROCEDURAL LEVEL GENERATION, OBJECT POOLER

The object of study - the processes of automatic generation of levels in game applications.

The subject of research - methods and tools for automatic generation of levels in game applications.

The purpose of the work is to investigate and develop a modified method of automatic generation of levels in game applications.

Research methods - analysis of existing methods of automatic level generation. An approach to solving the problem of automatic generation of levels in game applications is defined. The comparison of existing generation algorithms is made, the comparative analysis of their indicators is carried out, the most effective algorithms are allocated. A modification of generation algorithms that satisfies the task is proposed.

Scope - the software is used to automatically generate levels in game applications.

ЗМІСТ

ВСТУП

Сучасні персональні комп’ютери зобов’язані багатьом досягненням та нововведенням ігровій індустрії: звукові карти, відеокарти та 3D-графічні прискорювачі, швидші процесори та спеціалізовані співпроцесори, такі як PhysX, є одними з найбільш помітних удосконалень.

Звукові карти були розроблені для додавання звуку цифрової якості до ігор, і лише пізніше вони були вдосконалені для музики та аудіофайлів. Графічні карти на початку розроблялися для більшої кількості кольорів. Пізніше були розроблені графічні карти для графічних інтерфейсів користувача (GUI) та ігор.

У обчисленнях процедурне генерування - це метод алгоритмічного створення даних на відміну від ручного, як правило, за допомогою комбінації створених людиною активів та алгоритмів, поєднаних із випадковою і обчислювальною потужністю, що генерується комп'ютером. У комп’ютерній графіці цей метод зазвичай використовується для створення текстур та 3D-моделей. У відеоіграх він використовується для автоматичного створення великої кількості вмісту в грі. Залежно від реалізації, переваги процедурної генерації можуть включати менший розмір файлу, більший обсяг вмісту та випадковість для менш передбачуваного ігрового процесу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Процес розробки ігрових додатків

Розробка будь-якої гри ґрунтується на трьох базових елементах – ігровий концепції, технології та візуальному стилі. Першим елементом є ігрова концепція, яка визначає правила гри і механіку взаємодії гравця з віртуальним світом. Як швидко гравець бігає і як високо він стрибає? Які перешкоди і яких персонажів він зустріне на своєму шляху? Яким чином він буде з ними взаємодіяти? На всі ці запитання відповідає геймдизайнер (Game designer), саме він і займається створенням ігрової концепції. Другий елемент – це технологія, яка представляє собою технічну базу для реалізації гри. Основний функціонал забезпечує ігровий движок (Game engine), а додаткові інструменти дозволяють дизайнерам і художникам створювати і редагувати вміст гри. Створенням технології, інструментарію, а так само реалізацією ігрового функціоналу і механіки зазвичай займаються програмісти. Третій базовий елемент, що лежить в основі будь-якої гри, – це візуальний стиль, який визначає вигляд гри. Створенням візуальної частини займається широке коло художників, що працюють в самих різних областях комп'ютерної графіки. Наприклад, концепт-художники розробляють унікальний вигляд ігрового світу. Віртуальні скульптори створюють моделі персонажів, аніматори анімують їх, надавши кожному руху свій неповторний характер. А художники по ігровому оточенню додають всі необхідні деталі для декорування рівнів і наповняють його спецефектами. Всі ці три елементи в сукупності служать одній меті – створенню ігрового процесу, спрямованого на розвагу гравця. Для позначення процесу взаємодії гравця з грою існує спеціальний термін «геймплей» (Gameplay).

Геймплей – це те, що відрізняє комп'ютерну гру від таких неінтерактивних видів розваг, як книги або кіно. Дивлячись кіно або читаючи

книгу, людина є лише стороннім спостерігачем. У випадку з геймплеєм ми отримуємо можливість безпосередньо взаємодіяти з грою, вирішуючи поставлені геймдизайнером завдання за допомогою ігрових механік. У будь-якій грі можна виділити два основних типи геймплея - геймплей, що становить ядро гри, і геймплей на рівнях. В англійській термінології ці два типи найчастіше називають «core gameplay» і «level gameplay». Геймплей, що становить ядро гри (core gameplay) – це набір базових правил і механік, які доступні гравцеві за замовчуванням. Наприклад, цей тип геймплея визначає характеристики персонажів, їх здібності, а також те, як вони взаємодіють один з одним. Хорошим прикладом рівня без геймплея є чисте поле або порожня кімната, де гравець може використовувати тільки дані йому за умовчанням здатності, тобто бігати, стрибати і стріляти. Отже, від дизайнера рівнів потрібно спроектувати таку локацію, яка забезпечить достатню кількість цікавих ігрових ситуацій для застосування кожної базової механіки. Таким чином, оперуючи простором, об'єктами і настройками віртуального світу, дизайнер створює геймплей на рівнях (level gameplay), що дозволяє подарувати новий унікальний досвід взаємодії з грою. В іграх-стратегіях рівні називають картами (Map), де геймплей створюється за рахунок різних особливостей ландшафту, розташування гравців і цінних ресурсів. У гонках рівні представляють собою траси з великою різноманітністю маршрутів і перешкод на них. У пригодницьких іграх геймплей будується на тому, які перешкоди, пастки, головоломки і вороги зустрінуться гравцеві. Таким чином, в іграх різних жанрів рівні за своєю структурою і зовнішнім виглядом можуть кардинально відрізнятися один від одного і носити різні назви (карти, місії, зони, етапи, завдання), але разом з тим виконувати одну і ту ж функцію - доповнювати, розвивати і створювати варіативність базового геймплею.

У процесі створення гри можуть приймати участь розробники починаючи від однієї людини і закінчуючи міжнародною командою, розсіяною по всьому світу. Розробка традиційних комерційних ігор для ПК та консолей зазвичай фінансується видавцем, і завершення може зайняти

декілька років. Інді-ігри зазвичай займають менше часу та грошей, і їх можуть виробляти приватні особи та менші розробники. Незалежна ігрова індустрія зростає, чому сприяє зростання доступного програмного забезпечення для розробки ігор, таких як платформа Unity та Unreal Engine, та нових систем розповсюдження в Інтернеті, таких як Steam та Uplay, а також ринку мобільних ігор для Android та пристрої iOS.

Перші відеоігри, розроблені в 1960-х роках, зазвичай не комерціалізувались. Вони вимагали, щоб працювали мейнфреймові комп'ютери, і вони були недоступні для широкої громадськості. Розробка комерційних ігор розпочалася в 70-х роках з появою відеоігрових консолей першого покоління та ранніх домашніх комп'ютерів, таких як Apple I. У той час, завдяки низьким витратам та низьким можливостям комп'ютерів, самотній розробник міг розробити повну гру. Однак наприкінці 80-х та 90-х років постійно зростаюча обчислювальна потужність комп’ютера та підвищені очікування геймерів ускладнювали для однієї людини створення гри для основної консолі чи гри на ПК. [1]

Розглянемо етапи розробки ігрових додатків.

- Концептуаннння.

На цьому першому кроці команда придумує концепцію гри, і проводить початкове опрацювання ігрового дизайну. Головна мета даного етапу - це геймдизайнерская документація, що включає в себе Vision (розгорнутий документ, що описує гру, як кінцевий бізнес-продукт) і Concept Document (початкове опрацювання всіх аспектів гри).

У продуктовій документації геймдизайнер формулює і зберігає свої ідеї. Виконавцю документація дозволяє правильно розуміти свої завдання по реалізації продукту. Тестувальник чітко бачить, що і як тестувати. Для Продюсера / ПМА ця документація надає матеріал для формування планів і контролю виконання завдань. Інвестор же (особливо на ранніх етапах) отримує розуміння, на що саме він виділяє кошти.

Принципово важливо, щоб вся проектна та продуктова документація підтримувалася в актуальному стані на всіх етапах розвитку проекту. Для її

ефективного використання та оновлення потрібно використовувати спеціальні інструменти. Наприклад, використання Confluence для ведення геймдизайнерської документації сильно спрощує процес паралельного внесення змін декількома учасниками розробки, а також дозволяє всім членам команди оперативно отримувати будь-яку актуальну інформацію, що стосується продукту і всіх його змін.

Серед ключових принципів формування продуктової документації варто відзначити: структурованість, захищеність від різночитань, повний опис продукту, регулярну актуалізацію.

- Створення прототипу.

Важливий етап проектування будь-якої гри - це створення прототипу. Те, що добре виглядає «на папері», зовсім не обов'язково буде цікаво в реальності. Прототип реалізується для оцінки основного ігрового процесу, перевірки різних гіпотез, проведення тестів ігрових механік, для перевірки ключових технічних моментів.

Дуже важливо на етапі створення прототипу реалізовувати тільки те, що потрібно перевірити і в стислі терміни. Прототип повинен бути простим в реалізації, тому що після досягнення поставлених перед ним цілей, від нього позбавляються. Серйозна помилка початківців розробників - нести тимчасову інфраструктуру в основний проект.

- Вертикальний зріз.

Мета вертикального зрізу - отримати мінімально можливу повноцінну версію гри, що включає в себе повністю реалізований основний ігровий процес. При цьому всі базові особливості гри присутні як мінімум в чорновому варіанті. Реалізовано мінімальний, але достатній для втілення повноцінного ігрового процесу набір контенту (один рівень або одна локація).

- Виробництво контенту.

На цьому етапі виробляється достатня кількість контенту для першого запуску на зовнішню аудиторію. Реалізуються всі особливості, заплановані

до закритого бета-тестування. Це найбільш тривалий етап, який може займати, для великих клієнтських проектів рік і більше.

На цьому етапі задіюється найбільша кількість фахівців, які займаються виробництвом всього основного наповнення гри. Художники створюють всі графічні ресурси, Геймдизайнери налаштовують баланс, програмісти реалізують і полірують все особливості гри.

- Закрите бета-тестування.

На етапі CBT продукт вперше демонструється досить широкому загалу. Серед найбільш важливих завдань на цьому етапі виступають: пошук і виправлення гейм-дизайнерських помилок, проблем ігрової логіки і усунення критичних помилок. На цьому етапі в грі присутні вже всі ключові особливості, створено досить контенту для повноцінної гри тривалий час, налаштовані збір і аналіз статистики. Тестування йде по тест-плану, проводяться стрес-тести вже із залученням реальних гравців.

- Відкрите бета-тестування.

На цьому етапі триває тестування гри, але вже на широкій аудиторії. Йде оптимізація під великі навантаження. Гра повинна бути готова для прийому великої аудиторії. У грі реалізований білінг і приймаються платежі.

На цьому етапі повністю завершується розробка нових особливостей. Відбувається feature freeze, програмісти перестають реалізовувати щось нове, а повністю переключаються на налагодження і тюнінг наявних особливостей. Геймдизайнер, продюсер і аналітики роблять висновки з зібраної на CBT статистики і перевіряють ефективність монетизації.

При цьому, до початку етапу повинна повністю функціонувати інфраструктура проекту: сайт, групи в соціальнних мережах, канали залучення, підтримка користувачів.

- Реліз гри.

Ключова мета - це отримання прибутку. Базовим для оцінки прибутковості є критерій: кількість грошей, принесених в середньому одним

гравцем за весь час (LTV - lifetime value), має перевершувати витрати на залучення цього гравця (CPI - cost per install).

На цьому етапі має бути повністю налагодженним оперування продукту (технічна підтримка, робота з ком'юніті), мають бути дотриманими маркетингові та фінансові плани, вестися роботи по поліпшенню фінансових показників, активно відпрацьовуються канали по залученню трафіку.

Команда розробки на цьому етапі займається виправленням технічних помилок, що виявляються в процесі експлуатації і оптимізацією продукту. Геймдизайнери займаються тонким налаштуванням геймплея під реальну ситуацію в ігровому світі (особливо актуально для MMO проектів). Також реалізує різні внутрішньоігрові особливості, що підтримують нові монетизаційні схеми. І звичайно йде розробка та інтеграція в продукт нового контенту, що підтримує інтерес гравців[2].

1.2 Історія індустрії відеоігор

У далекому 1978 році комерційно успішна гра Space Invaders сильно мотивувала великих інвесторів звернути увагу на ігрову індустрію. Ця відеогра була випущена в Японії на платформі, яку ми знаємо під назвою аркадні апарати. Гра стала настільки популярною, що викликала в країні дефіцит монет, які були потрібні для того, щоб пограти в неї. З Японії гра швидко знайшла своє поширення в Америці, де за 2 роки було продано 60 000 апаратів. Вони стояли у всіх барах, боулінг-клубах і кінотеатрах. Завдяки Space Invaders почався бум розвитку аркадних апаратів, які стали першою сходинкою еволюції індустрії ігор. У 1979 році це було справжнім проривом: американський дистриб'ютор гри Belly Midway потроїв ринок ігрових автоматів в США, довівши обсяги продажів до 1,33 млрд. доларів за рік. Багато відомих комп'ютерних ігор вперше з'явилися саме на ігрових апаратах і вже пізніше були перенести на інші платформи. Наприклад, Pac-Man, Street Fighter, Killer Instinct.

Разом з ігровими апаратами ігри розвивалися на платформах ігрових приставок. І хоча історія приставок почалася ще в 1972 році разом з релізом першої в світі домашньої ігрової приставки Magnavox Odyssey, прорив в цьому напрямку починається в 1977 році, коли в продаж надходить ігрова приставка Atari 2600. Завдяки інвестиціям Atari популяризація комп'ютерних та відеоігор виходить на новий рівень . З'являються нові великі гравці, охочі зайняти тепле місце на ринку відеоігор. Як найбільші, окрім Atari, в історію увійшли Sega, Nintendo, Sony і Microsoft.

Іншою великою ігровою платформою «з найдавніших часів» стали персональні комп'ютери. Перші ігри на них з'явилися ще в 1960-х роках. Це були текстові пригодницькі ігри, в яких спілкування між гравцем і комп'ютером здійснювалося за допомогою введення команд через клавіатуру. Подальша еволюція ігор тісно пов'язана з ростом продуктивності і можливостей комп'ютерного обладнання. Це і популяризація комп'ютерних мишей в якості інтерфейсу взаємодії людина-комп'ютер, і поява звукових карт, що дають можливість відтворювати повноцінні звуки, і щорічне поліпшення якості зображення, що виводиться, і звичайно розвиток комп'ютерних мереж, що вилилися в результаті в появу Інтернету. З еволюцією мереж на зміну звичним всім однокористувацьким клієнтським іграм приходить нове покоління ігор, які дають можливість грати разом з іншими гравцями. Найперша онлайн гра була зроблена в текстовому вигляді і запущена в мережу TelNet в 1978 році. Розробниками цієї гри були Річард Бартл і Рой Трабшоу. Вони назвали своє творіння Multi-user Dungeon, скорочено MUD. Надалі ця абревіатура стала використовуватися в якості класифікації для багатокористувацьких онлайн ігор, де абсолютно все відображається у вигляді тексту. MUD є джерелом всіх віртуальних світів, які ми сьогодні знаємо.

З появою інтернету приходять онлайн проекти: клієнтські і браузерні. В кінці 90-х зростаючі швидкості доступу в інтернет уможливлюють становлення браузера як самостійної ігрової платформи. Він виступає в ролі

операційної оболонки для ігор, дозволяючи грати без установки додаткових програм.

В середині двохтисячних настає експоненціальне зростання соціальних мереж, які стають новим способом спілкування між людьми. Розробники ігор тепер просто не можуть ігнорувати браузерну платформу і всередині неї виділяється окрема підплатформа і ігри під неї, які зараз називають соціальними. Тоді багатьом здавалося, що це колосальний прорив ігрової індустрії. Величезна кількість розробників пішла в соціальні ігри, темпи річного зростання перевищували 200% просто тому, що це була інновація і раніше такого ринку навіть не існувало. Однак насичення прийшло досить швидко. На зміну низькоякісним ігровим продуктам, виробленим з шаленою швидкістю з метою зайняти ринок, приходять високоякісні ігри з хорошою графікою і серйозним геймплеєм.

Наступний виток розвитку ігрової індустрії стався завдяки колосальному зростанню популярності мобільних телефонів, які стали новою платформою спілкування людей і новою платформою розповсюдження ігор. Ігри на Java, які були для мобільних телефонів не становили серйозної конкуренції ні браузерним, ні клієнтським, ні соціальним іграм. Однак починаючи з кінця першого десятиліття 2000-х років поява смартфонів, під які можна було розробляти потужні проекти, відкрила для індустрії ринок мобільних ігор у всій красі. Взагалі мобільна платформа реалізується зараз через два типи пристроїв: смартфони і планшети. Але так як операційні системи, технічні характеристики і способи надання ігор споживачеві на цих пристроях практично ідентичні, розробники рідко поділяють їх на різні платформи. Саме мобільні ігри зараз вважаються перспективним ринком, здатним генерувати максимальні прибутки при низькому порозі входження.

Історія становлення ігрових платформ показує, що з розвитком технологій прискорюється процес появи інноваційних платформ для ігор. Деякі з них «підривають» ринок і витісняють старі, деякі не знаходять вибухового зростання але займають свою нішу - наприклад Smart TV, ігри в

літаках, ігри на вейпах і ін. Цілком імовірно, що вже в найближчі роки ринок мобільних ігор буде витіснений з тренда ігрової індустрії більш досконалою технологією взаємодії людина-комп'ютер. Це може стати як віртуальна або доповнена реальність в поточній реалізації, так і віртуальна реальність з повним зануренням [3].

1.3 Види ігрових додатків

Для різних видів ігрових додатків алгоритми генерації рівня можуть кардинально відрізнятися. Тому необхідно розглянути основні види, для подальшого врахування їх особливостей.

- Стратегічні ігри.

Стратегічна гра - популярний жанр комп'ютерних ігор, в якому запорукою досягнення перемоги є планування і стратегічне мислення.

Сенс таких ігор полягає в управлінні деяким ресурсом, який необхідно перетворити на перевагу над противником за допомогою оперативного плану, що розробляється з урахуванням мінливої обстановки.

Звичайними ресурсами в військових стратегіях є війська (окремі персонажі, підрозділи або армії) і позиція, які слід розвивати і використовувати для досягнення переваги і перемоги. В економічних стратегіях акцент ставиться на розвиток економічної інфраструктури.

Сучасні стратегічні ігри, як правило, поєднують у собі як військові, так і економічні ознаки (збір ресурсів, підготовка військ).

Також існують онлайн-стратегії, призначені тільки для гри в Інтернеті. Серед них можна виділити браузерні ігри та ігри, що вимагають використання клієнта.

Розрізняють покрокові стратегічні ігри та стратегічні ігри в реальному часі. На рисунку 1.1 наведено приклад стратегічної гри.

Рисунок 1.1 – Стратегічнна гра

- Покрокові стратегії.

Покрокові стратегії - ігри, в яких гравці роблять свої дії по-черзі. Поділ ігрового процесу на ходи відриває його від реального життя і позбавляє гру динамізму, в результаті чого ці ігри не так популярні, як стратегії в реальному часі.

З іншого боку, у гравця набагато більше часу на роздуми, під час здійснення ходу його ніщо не квапить, що дозволяє приділяти більше часу плануванню.

На рисунку 1.2 наведено приклад покрокової стратегії.

Рисунок 1.2 – Покрокова стратегія

- Стратегії в реальному часі.

У цих стратегіях гравці проводять свої дії одночасно. Вони з'явилися дещо пізніше покрокових, першою популярною грою цього жанру була Dune II (1992 рік), сюжет якої заснований на однойменному творі Френка Герберта.

Уже тоді сформувалися загальні принципи стратегій в реальному часі:

- економіка в грі носить допоміжний характер і націлена на збір ресурсів, в основі економіки лежать «будівлі», які можна будувати і руйнувати;
- «Будівлі» будують юніти і проводять дослідження. Деякі будівлі можуть атакувати противника;
- юніт - будь-яка бойова одиниця (піхотинець, танк, літак, корабель), яка зазвичай може атакувати інші юніти і будівлі і знищувати їх. Юніти мають параметри, головними серед яких є «життя», броня (зменшує пошкодження), швидкість;
- збір ресурсів здійснюється особливими юнітами в спеціально передбачених для цього місцях, після чого вони переносяться до

спеціального будинку на базі і тільки після цього надходять в розпорядження гравця;

- ресурси можуть бути різних видів (наприклад, золото, деревина, гроші, метал, вугілля) і витрачаються на будівництво юнітів і будівель (також на це йде час);
- всі будівлі і юніти мають радіус огляду, далі якого «бачити» вони не можуть;
- гравець може бачити, що відбувається, тільки на тих територіях, що потрапляють в радіус огляду його будівель і юнітів. Ті території, на яких він ще не був, зафарбовані чорним. Ті, на яких його війська вже були, але в даний момент не можуть їх бачити, покриті так званим «туманом війни».

Найбільш популярні серії стратегій в реальному часі: Warcraft, StarCraft.

- Глобальні стратегії.

Найбільш складні стратегії, в яких гравець управляє державою. У його руках - не тільки війна і економіка, але і науковий прогрес, освоєння нових земель і дипломатія. Ці ігри поділяються на історичні, фантастичні і космічні. Найбільш відомі представники жанру: Civilization, Master of Orion, серія Total War.

На рисунку 1.3 наведено приклад глобальної стратегії.

Рисунок 1.3 – Глобальнна стратегія

- Шутери.

Шутер - жанр комп'ютерних ігор, в яких успіх гравця у великій мірі залежить від його швидкості реакції і здатності швидко приймати тактичні рішення. Дія таких ігор розвивається дуже динамічно і вимагає концентрації уваги і швидкої реакції на події в грі. При цьому в якості основного засобу прогресу в грі, як правило, використовується будь-яка зброя.

На рисунку 1.4 наведено приклад шутера.

Рисуонк 1.4 – Шутер

- ММОРПГ.

ММОРПГ - розрахована на багато користувачів ролева онлайн-гра, (зазвичай в жанрі фентезі), в якій гравці можуть взаємодіяти один з одним в ігровому світі. Вони можуть допомагати один одному у виконанні завдань, обмінюватися речами, торгуватися, битися в дуелях. Гравцеві пропонується вигаданий герой, яким можна керувати.

У багатьох MMORPG є поділ на види ігрових серверів:

- PvE - гравець проти навколишнього світу. Гравець в основному взаємодіє з ігровим світом, покращуючи характеристики свого героя, добуваючи йому кращий одяг, вбиваючи сильних монстрів;
- PvP - гравець проти гравця. Головну роль грає протистояння, спілкування і обмін ігровими речами між гравцями;
- RP - рольові ігри. У них потрібно створити історію для свого персонажа, придумати йому ім'я, яке б йому відповідало.

На рисунку 1.5 наведено приклад ММОРПГ гри.

Рисунок 1.5 – ММОРПГ

- Квест.

Квест - пригодницька гра, один з основних жанрів ігор, що вимагають від гравця рішення розумових завдань для просування по сюжету. Сюжет може бути визначеним або ж давати безліч результатів, вибір яких залежить від дій гравця.

Перші «предки» квестів з'явилися на початку 1970-х, коли програміст і спелеолог Вільям Кроутер розробив програму під назвою Colossal Cave Adventure для ЕОМ марки PDP-10. Інтерфейс гри був текстовим, а сюжетом - пригоди героя у великій печері.

З розвитком обчислювальної техніки і появою домашніх комп'ютерів, що відрізняються розвиненою графічною системою, жанр квестів також отримав подальший розвиток.

Квести придбали перші графічні ілюстрації того, що відбувається, які спочатку були суто декоративними. Гравець як і раніше керував діями персонажа, вводячи послідовності команд на клавіатурі, а зображення залишалося статичним і служило лише для стимулювання уяви грає.

- Симулятори.

Симулятор - це програмне забезпечення або пристрої, що імітують управління будь-яким пристроєм. Можуть застосовуватися як для навчання, так і для розваги.

Широко відомі ігри-симулятори автомобілів, наземних бойових апаратів, космічних кораблів, літаків, вертольотів, підводних човнів і надводних бойових суден, поїздів. Так само існує ряд ігор-симуляторів реального життя.

Основним принципом симулятора є точне відтворення особливостей якоїсь тематичної області (наприклад: автосимулятор повинен максимально точно відтворювати фізичні особливості машин).

Автосимулятори, наприклад, діляться на два напрямки:

- аркадні симулятори (наприклад, серія Need for Speed). Такі ігри розраховані на те, щоб приємно провести час, тому управління сильно спрощено;
- реалістичні (наприклад, rFactor, Live for speed, iRacing). Такі ігри переконливо імітують поведінку автомобілів і доступні лише для умілих водіїв. Навіть ті, хто бере участь в реальних гонках, застосовують автосимулятори для тренувань і розваги.

На рисунку 1.6 наведено приклад симулятора.

Рисунок 1.6 – Гра-симулятор

- Економічні симулятори.

Економічні симулятори - стратегії, в яких військова сторона відсутня в принципі, так що від гравця потрібно тільки налагоджувати економіку (і зробити це набагато складніше, ніж в неекономічних стратегіях). Іноді в них присутні суперники (конкуренти), але військових дій з ними все одно не відбувається. Економічні симулятори - дуже специфічний жанр.

- Економічні онлайн ігри.

У зв'язку із загальним глобальним розвитком мережі Інтернет набирають силу економічні онлайн-ігри, тобто ігри, що представляють собою якийсь світ зі своїми правилами і законами, в якому гравцеві надається можливість займатися активною комерційною діяльністю: будувати торгові і виробничі підприємства і споруди, займатися розробкою природних ресурсів, вести активну фінансову і банківську діяльність в рамках ігрового простору.

Ігрові обороти гравців в таких іграх часом досягають декількох десятків тисяч доларів. Купуються і продаються найрізноманітніші природні копалини, об'єкти комерційної нерухомості, банки і навіть цілі компанії.

- Файтинги.

Файтинг - жанр відеоігор, що імітують рукопашний бій малого числа персонажів в межах обмеженого простору, званого ареною.

В більшості файтингів гравцеві не потрібно переміщатися по довгому рівню і не можна вийти за межі арени, а бій складається з непарного числа окремих раундів і не є безперервним.

Менш значними і необов'язково присутніми ознаками жанру є використання численних шкал для зображення життєво важливих показників персонажів і промальовування бійців на арені в профіль.

Важливою особливістю файтингів є їх націленість на змагання, а не на співпрацю гравців, що робить ігри цього жанру придатними для кіберспортивних чемпіонатів. Зазвичай файтинги надають гравцеві можливість вести бій в режимі «один на один» проти комп'ютерного супротивника або іншого гравця, рідше - дозволяють боротися одночасно трьом або чотирьом гравцям на одній арені[4].

На рисунку 1.7 наведено приклад файтинга.

Рисунок 1.7 - Файтинг

1.4 Постановка задачі

Проаналізувавши предметну область можна сказати, що в багатьох видах ігрових додатків використовується процедурна генерація рівнів. Завданням даної атестаційної роботи буде дослідження існуючих методів генерації рівнів в ігрових додатках та їх оптимізація, тобто створення модифікованого методу який вирішить частину проблем існуючих алгоритмів.

2 ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ ГЕНЕРАЦІЇ РІВНІВ

2.1 Генерація початкової геометрії: "BSP"

Двійкове розбиття простору (англ. Binary space partitioning) - метод рекурсивного розбиття евклідового простору в опуклі безлічі і гиперплощини. В результаті об'єкти отримують представлення у вигляді структури даних, так званого BSP-дерева.

Алгоритм досить простий. Спочатку створюємо прямокутник, розміром з усе ігрове поле(рисунок 2.1):

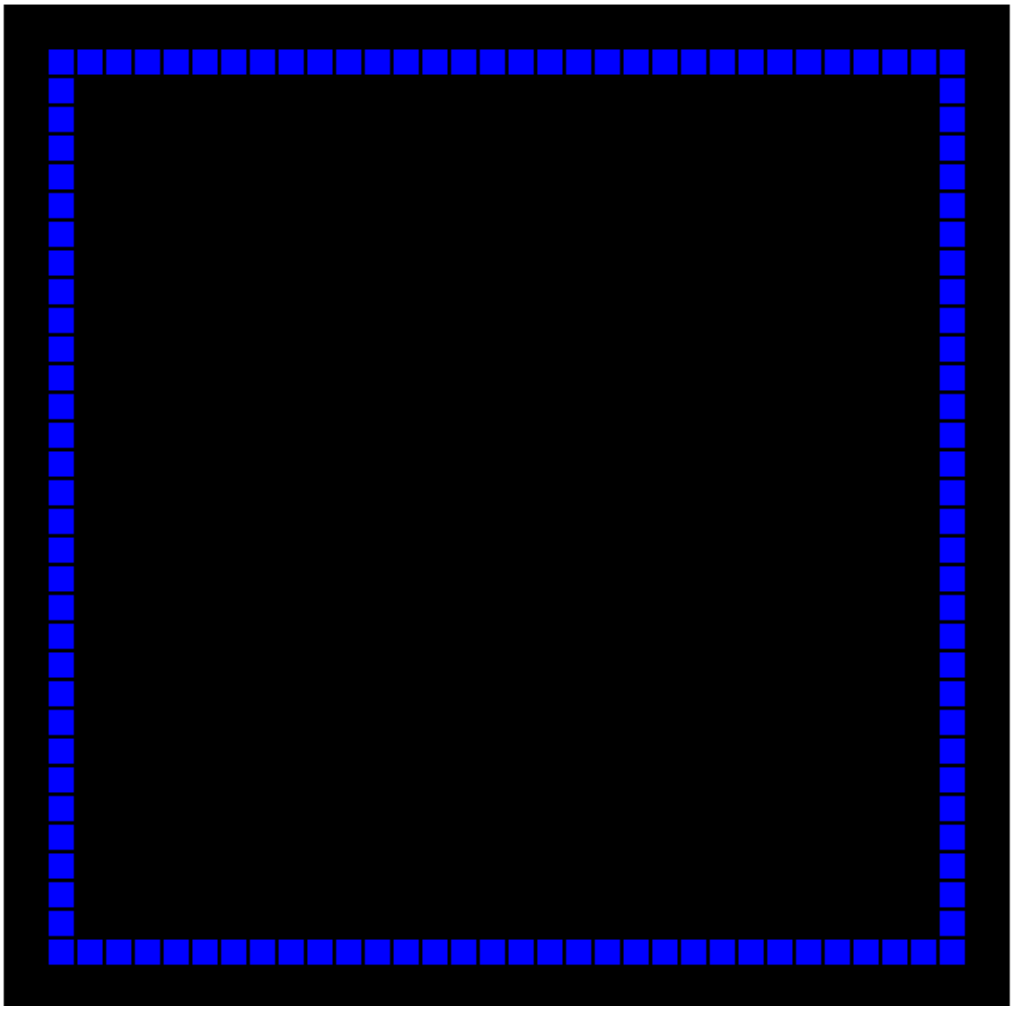


Рисунок 2.1 – Створення прямокутника розміров з усе ігрове поле

Потім ділимо його рандомно на дві частини - або горизонтально, або вертикально. Де буде проходити лінія поділу теж вибираємо випадковим чином(рисунок 2.2):

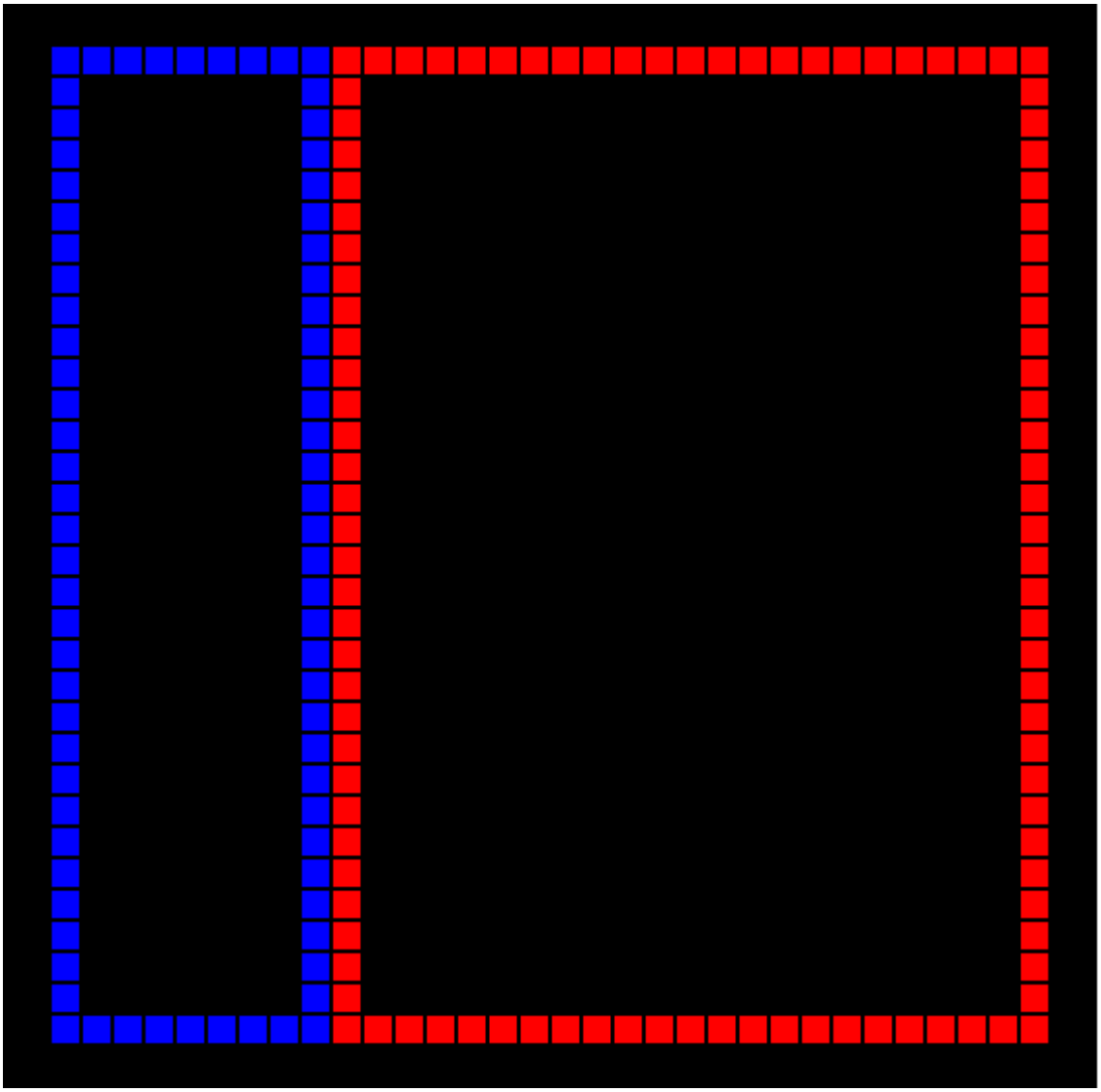


Рисунок 2.2 – Розділений на дві частини прямокутник

Рекурсивно проробляємо те ж саме для нових прямокутників (рисунок 2.3):

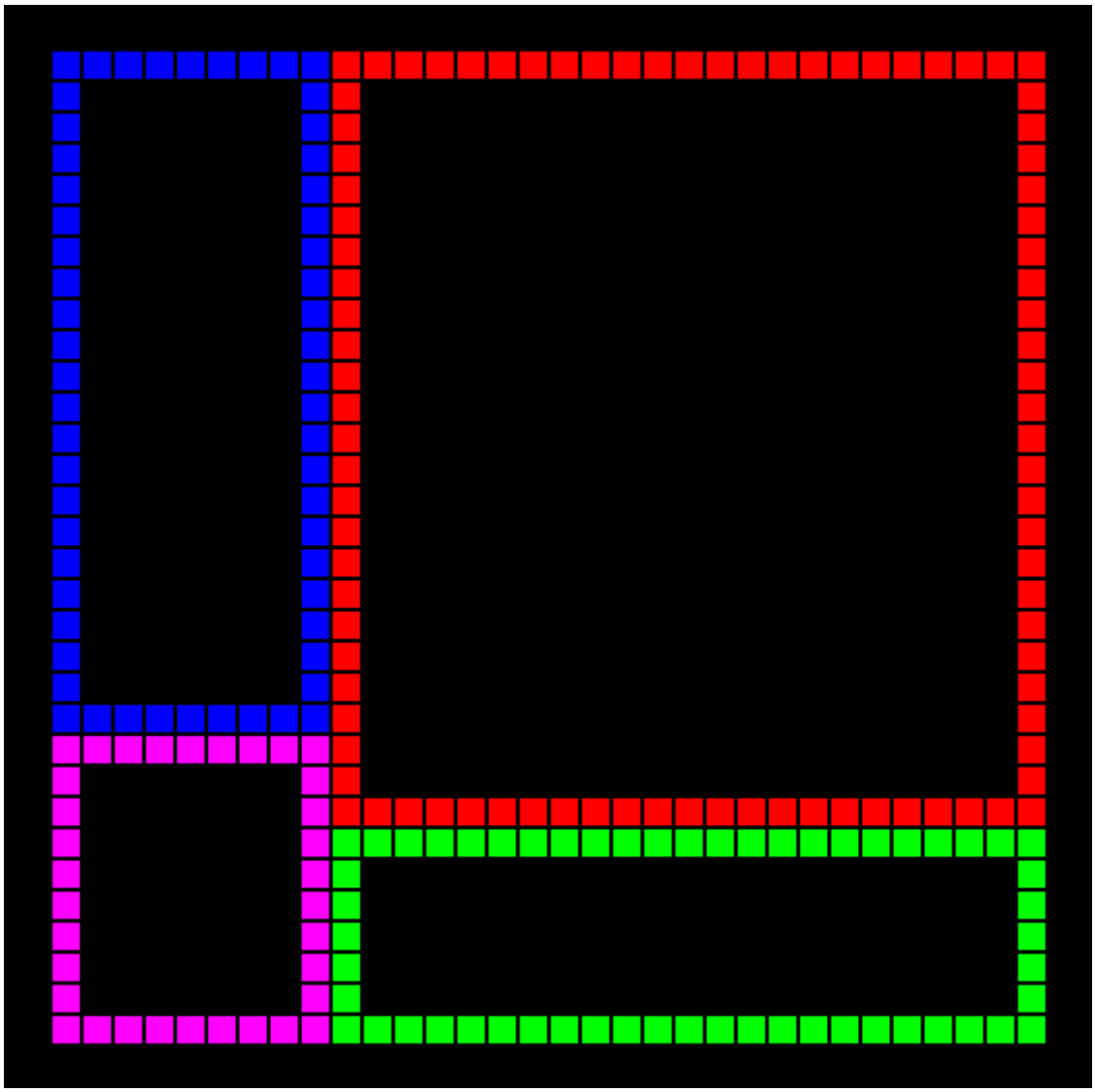


Рисунок 2.3 – Рекурсивно додали нові прямокутники

І ще раз, до певної межі (рисунок 2.4):

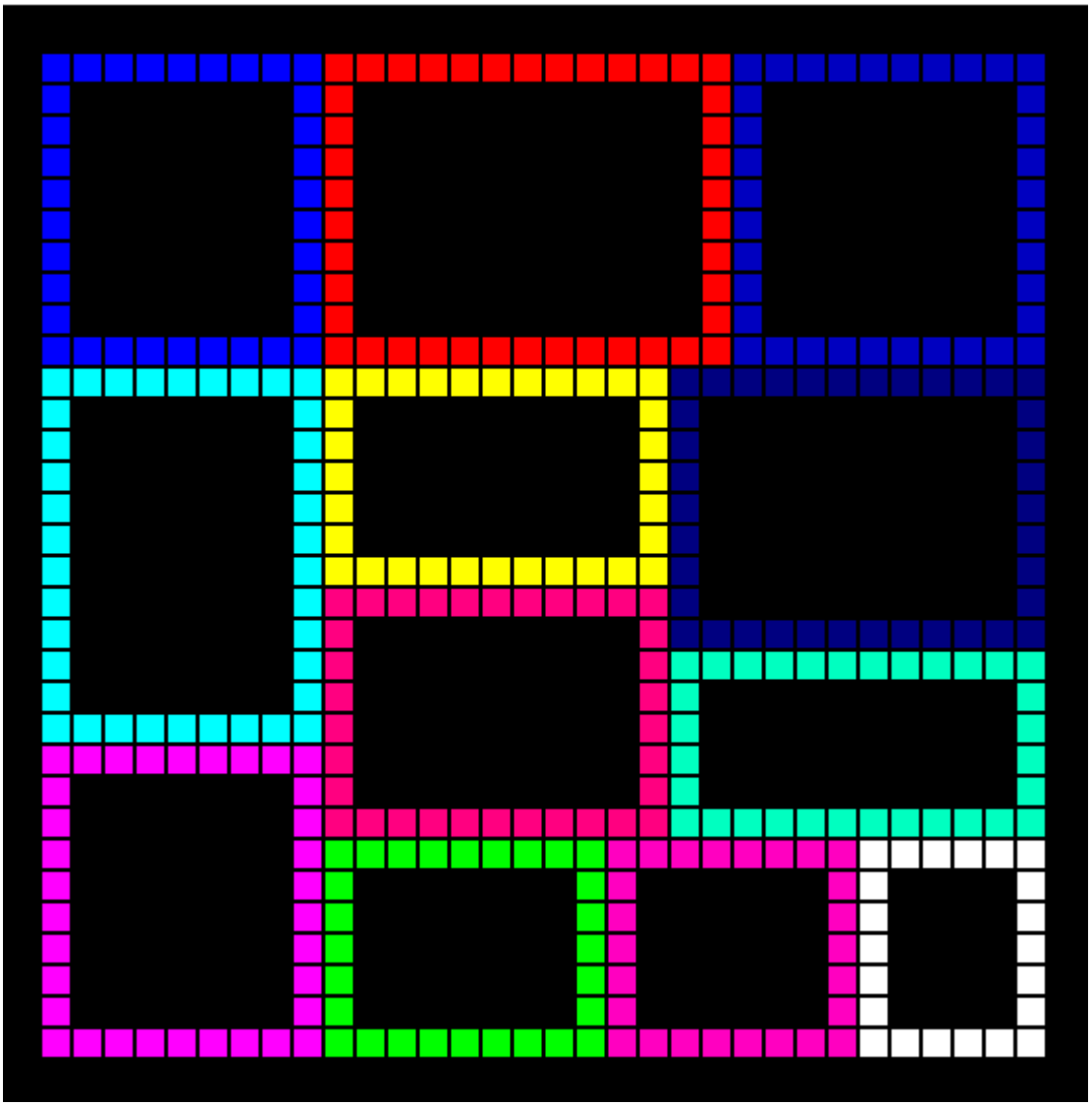


Рисунок 2.4 – Рекурсивно створені прямокутники до певної межі

Потім в кожному прямокутнику вибираємо "кімнату" - прямокутник такого ж розміру як вихідний або меншого (але не менше, ніж 3x3 - більш детально про це буде нижче) (рисунок 2.5).

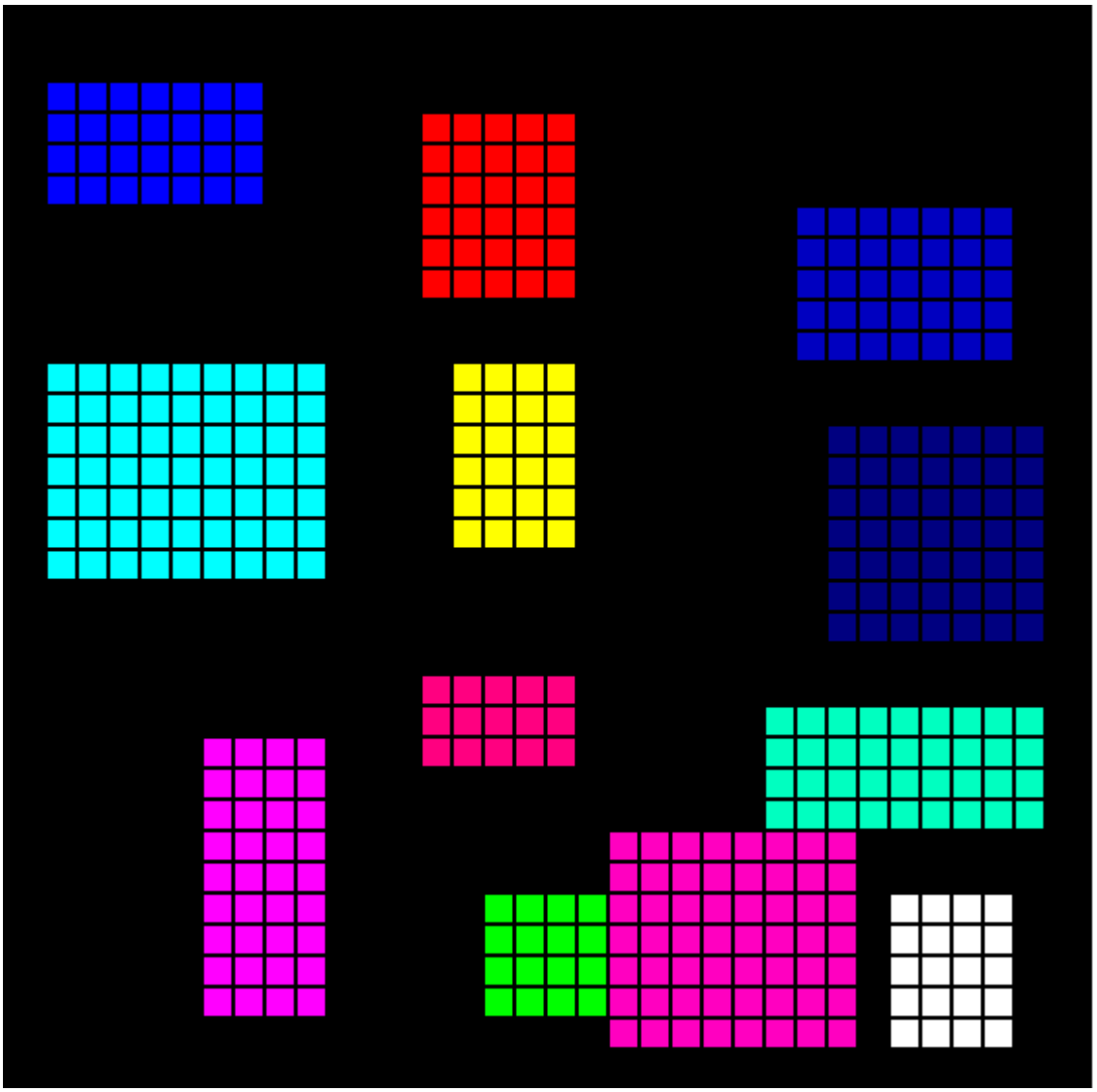


Рисунок 2.5 – Обрані у кожному прямокутнику «кімнати»

Потім кімнати з'єднуються коридорами. Але не кожна з кожною, а особливим методом, через те вони зберігаються в "BSP"-подібниій структурі.

Крім того, якщо кімната (на рисунку 2.6 - бірюзова) перетинає коридор, то вона повинна ділити його на дві різні секції (тому один і той же коридор може бути зображений різними кольорами):

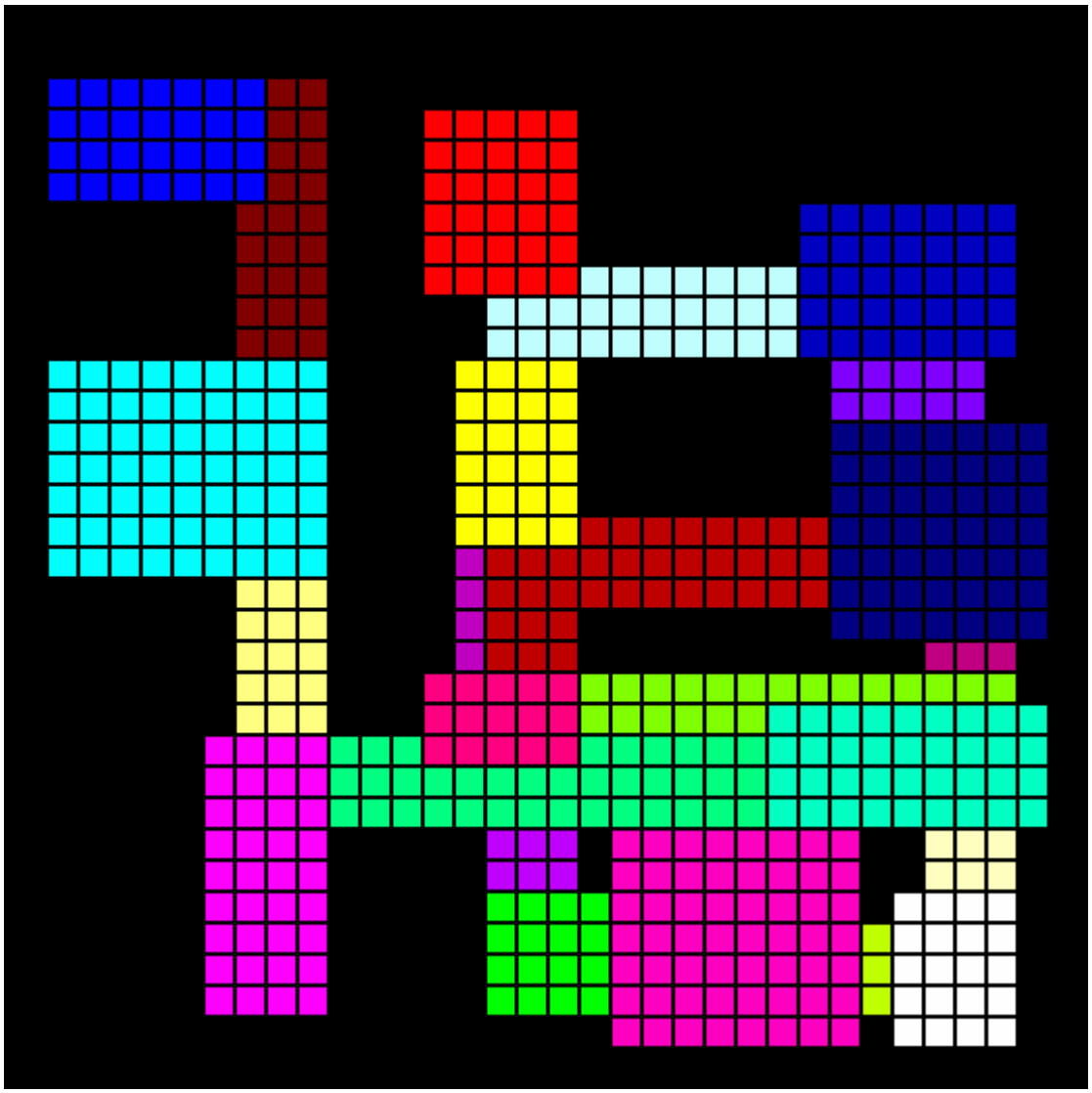


Рисунок 2.6 – Кімнати, з’єднані коридорами

Далі починається фаза позначки сміттєвих клітин (рисунок 2.7):

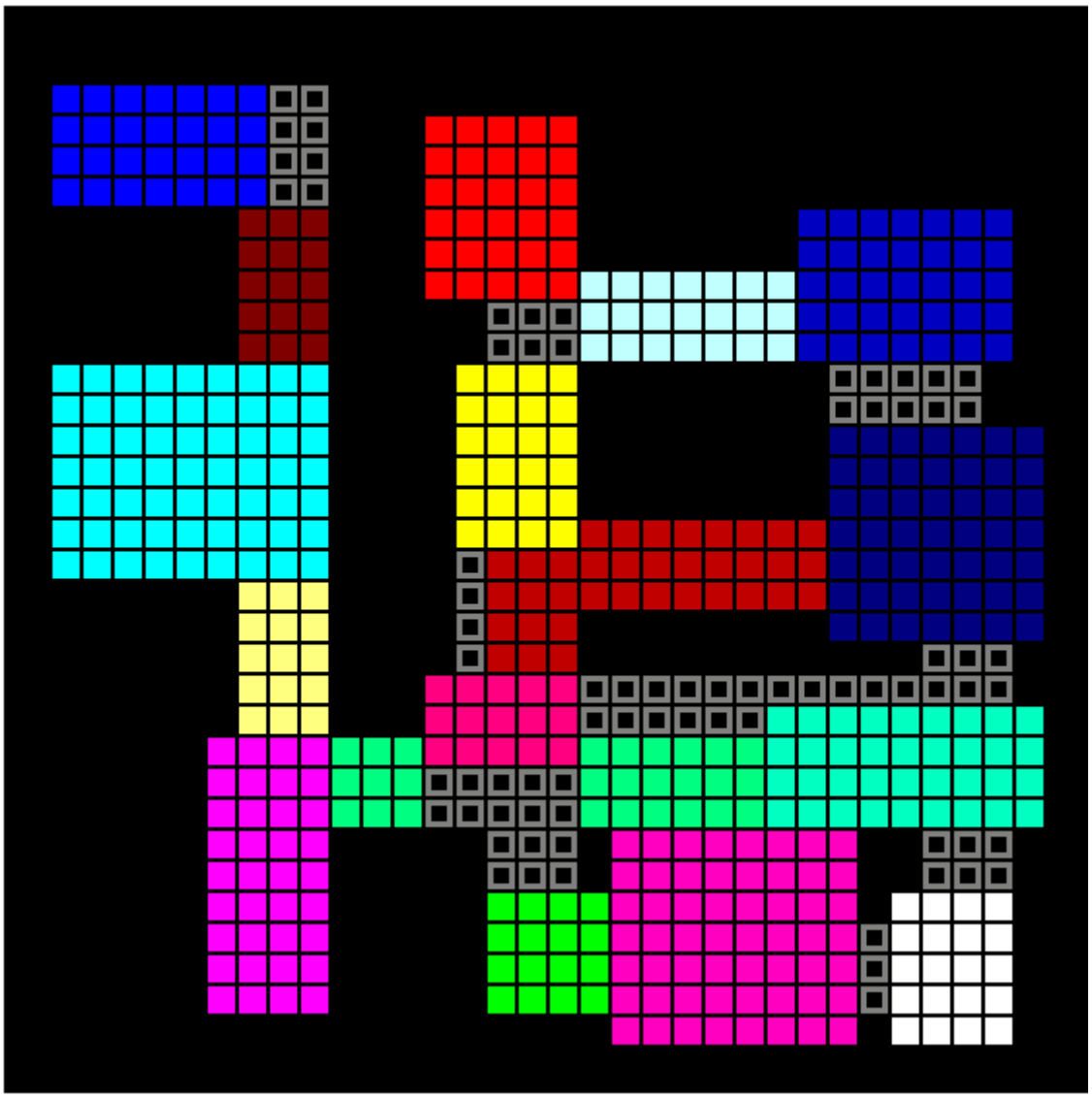


Рисунок 2.7 – Позначення «сміттєвих» клітин

Як було зазначено вище, ніякий сектор не може бути менше, ніж 3x3 клітини. Тому, всі ті клітини, які не підходять під це правило, позначаються. Але просто взяти, і видалити їх не можна - так пропадає багато з'єднань, і рівень виходить дуже обрізаним.

Замість цього, для кожної поміченої клітини шукається той сектор, до якого вона може примкнути (дотримуючись правила 3x3) (рисунок 2.8):

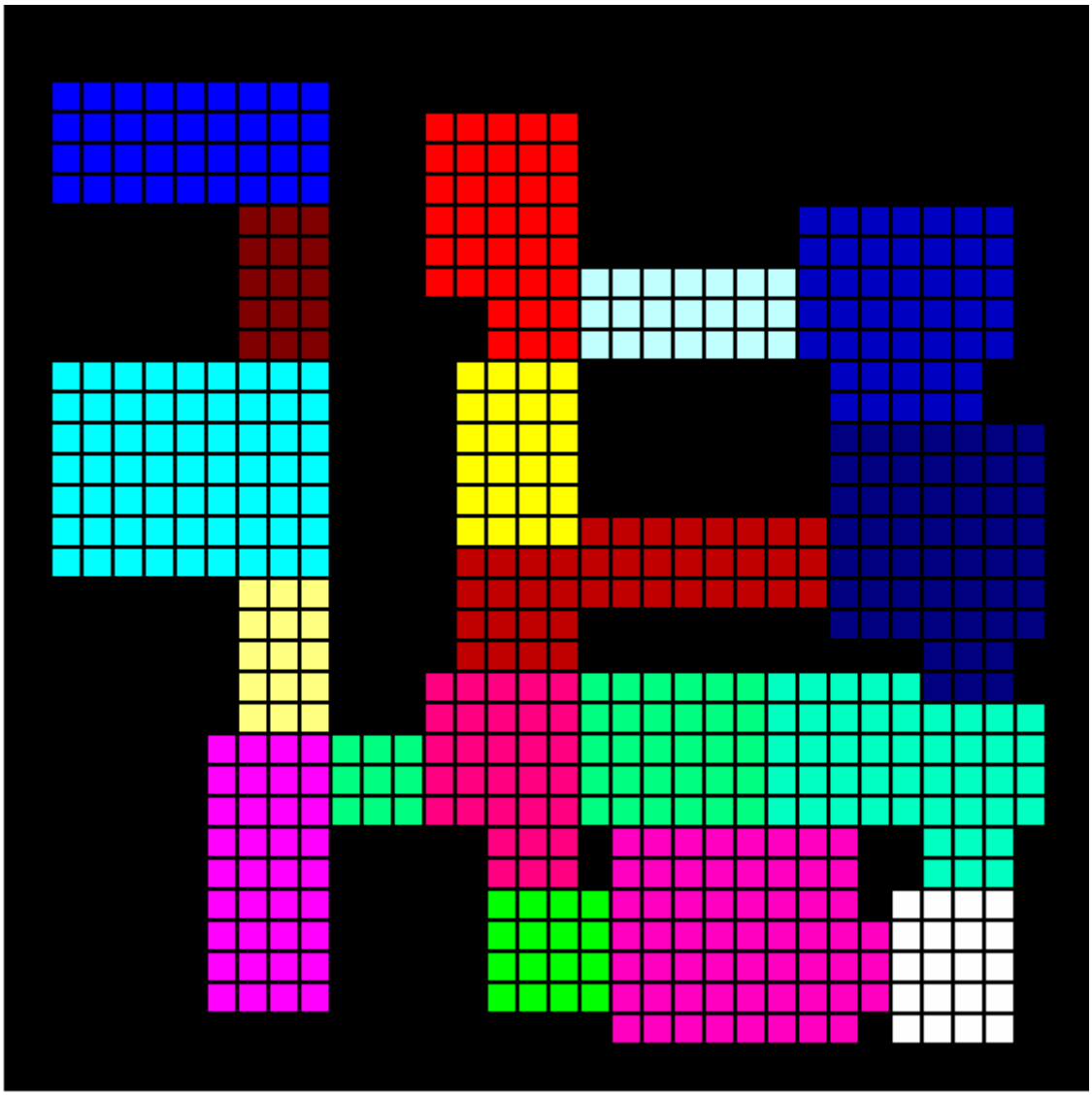


Рисунок 2.8 – «Сміттєві» клітини примкнули до кімнат

Якщо клітину не можна віднести до сектора, вона видаляється.

На останньому етапі ця картинка векторизується, і намальовані сектори перетворюються в такі полігони, у яких кожне ребро разташоване або строго вертикально, або строго горизонтально(рисунок 2.9) [5].

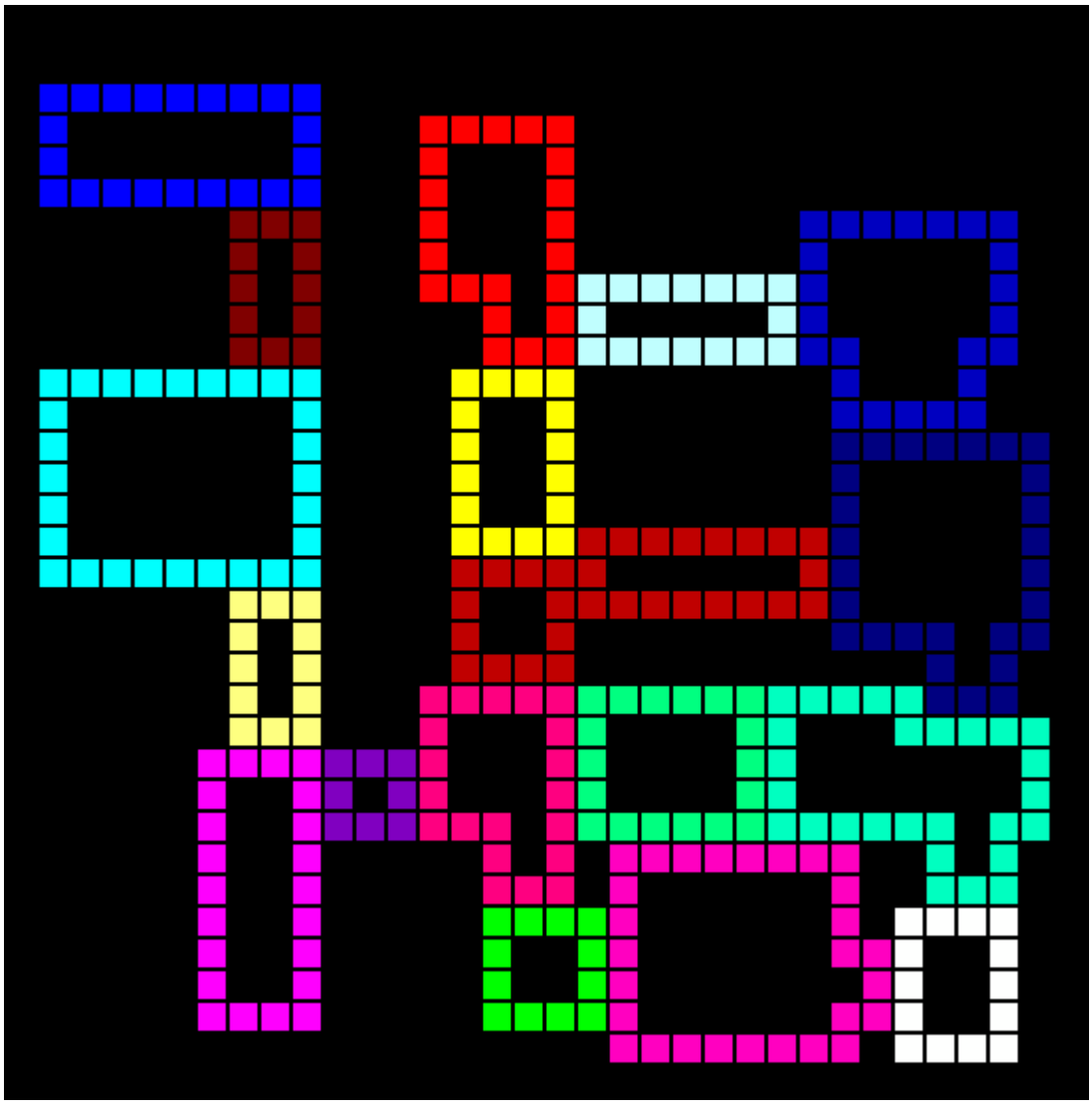


Рисунок 2.9 – Фінальний результат алгоритму генерації початкової геометрії “BSP”

2.2 Генерація початкової геометрії: планування приміщень

Для початку ігрове поле заповнюється таким собі стоп-значенням, а потім випадковим чином на ньому очищується прямокутна область (рисунок 2.10):

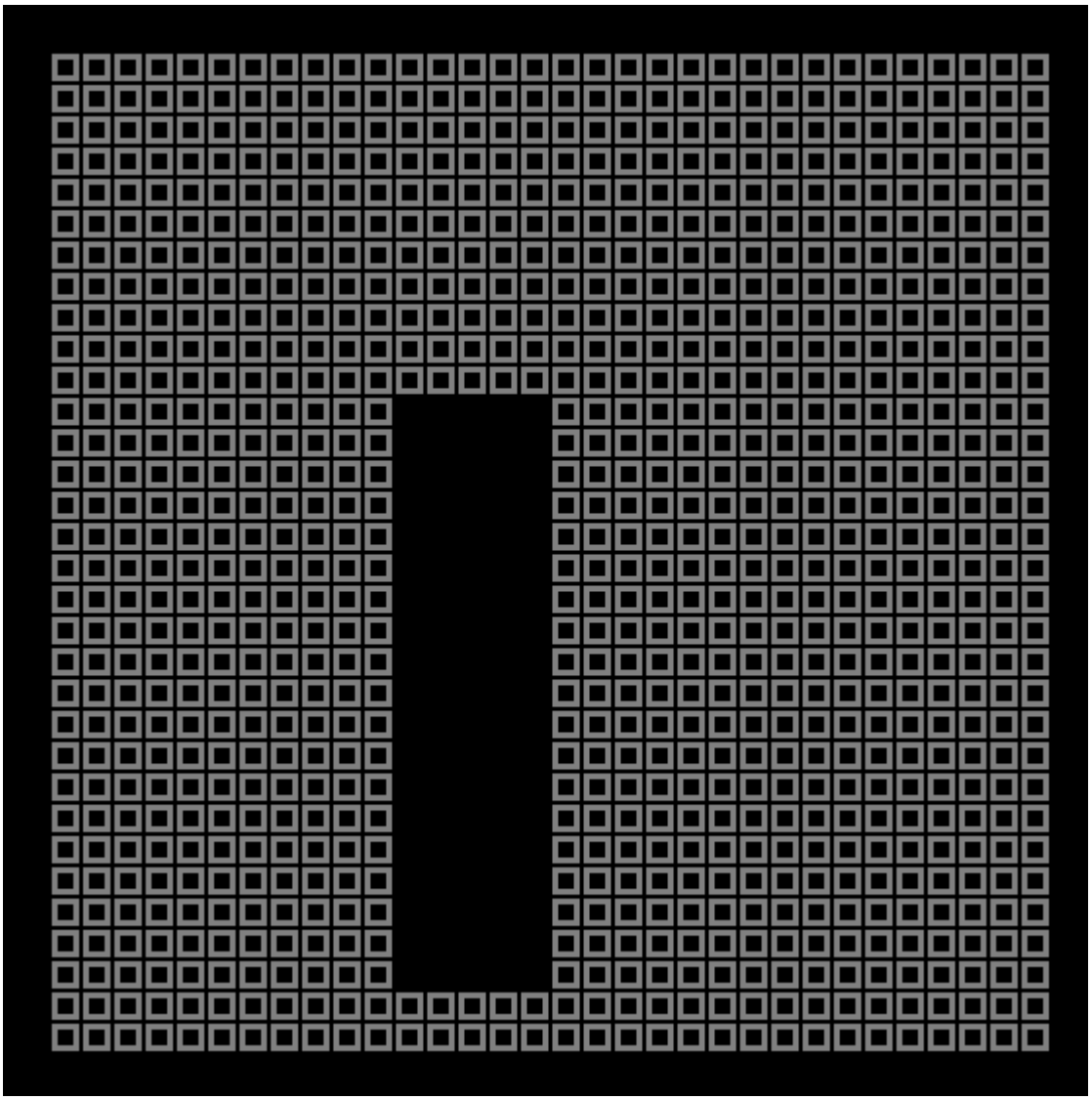


Рисунок 2.10 – Заповнення ігрового поля стоп-значенням

Етап очищення випадкового прямокутника проводиться ще деяку (теж випадкову) кількість разів, і в результаті виходять зовнішні контури рівня (рисунок 2.11):

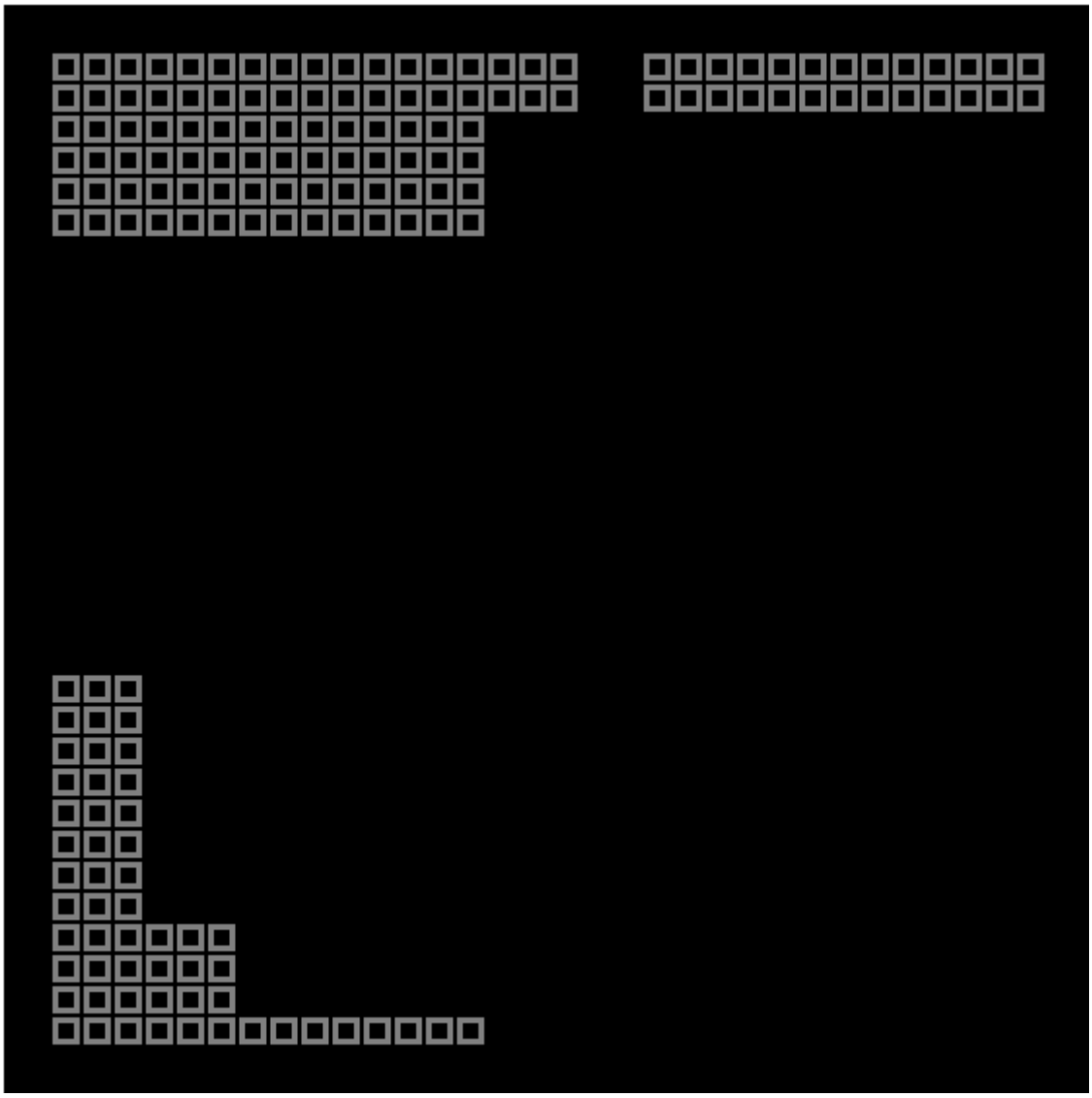


Рисунок 2.11 – Зовнішні контури рівня

У вільному просторі випадково розкидаються точки зростання кімнат (мінімальний розмір кімнати - 3x3) (рисунок 2.12):

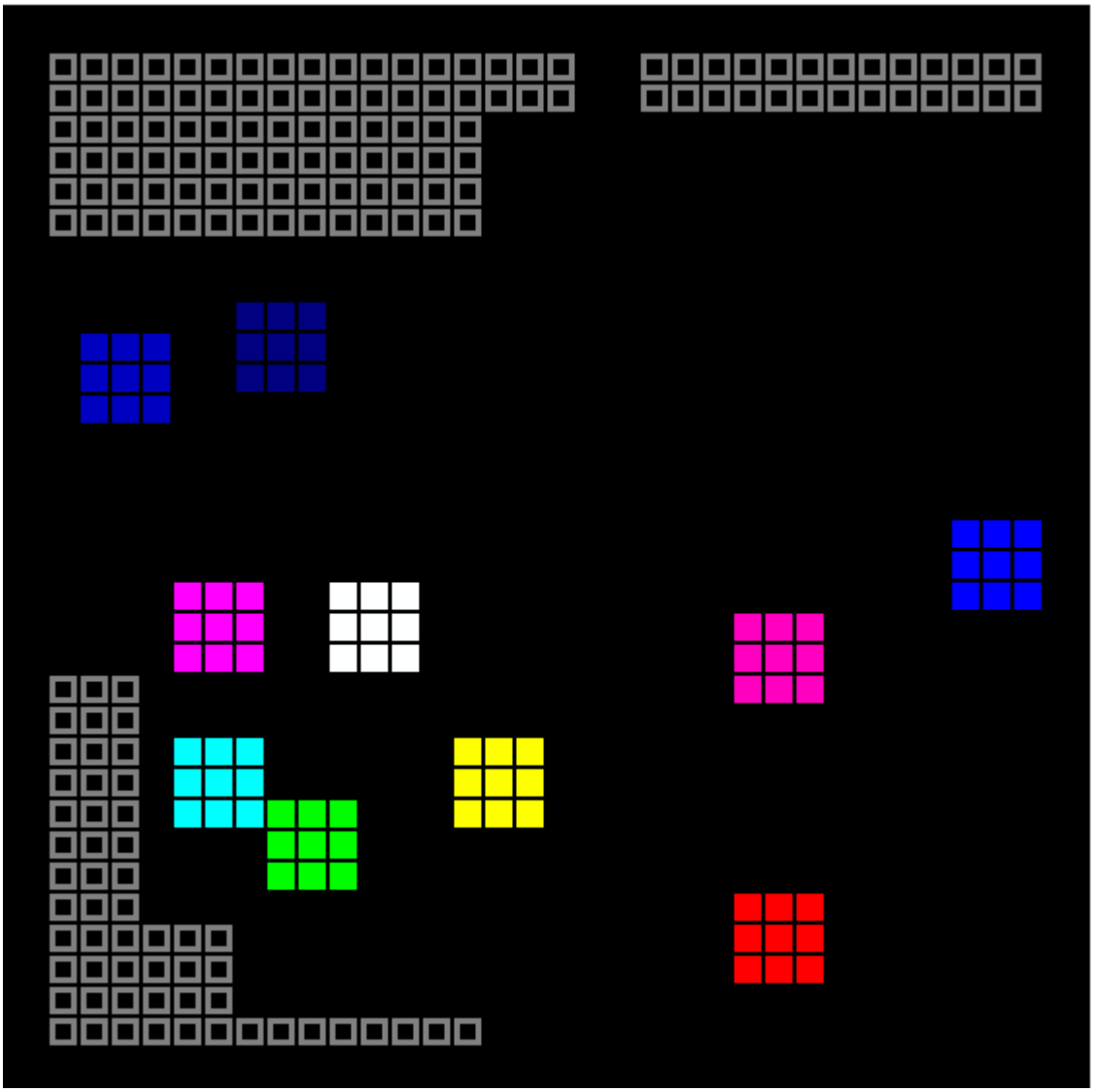
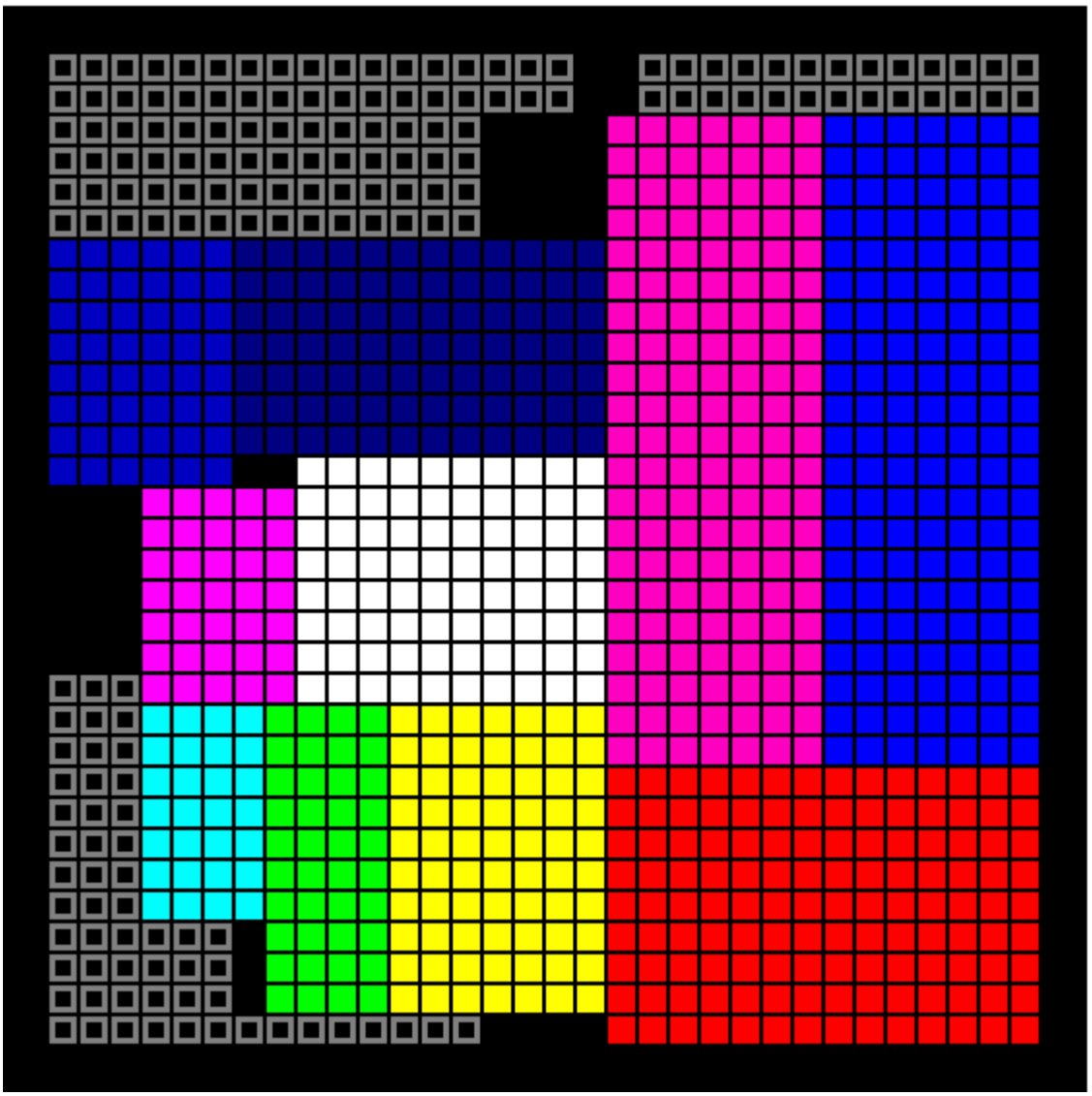


Рисунок 2.12 – Точки зростання кімнат

Починається перший етап зростання кімнат - для кожної кімнати вибирається найбільша сторона, яка ще може рости, і вона росте на 1 клітину (якщо є кілька сторін з однаковою довжиною - випадкова з них). Кімнати перебираються по черзі, щоб не було перетинів (рисунок 2.13).



Риснок 2.13 – Перший етап зростання кімнат

Потім кімнати перетворюються в полібокси (рисунок 2.14):

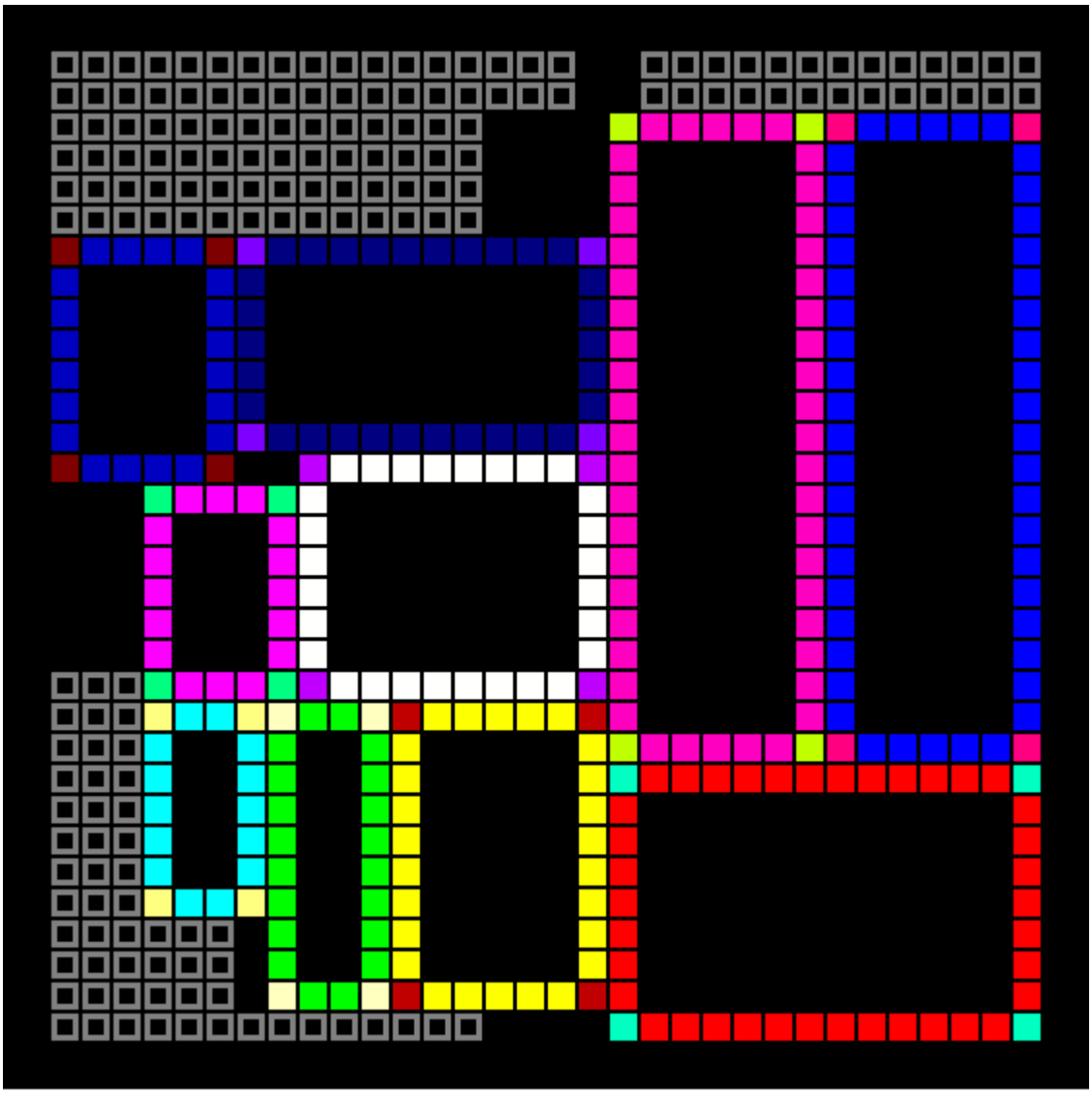


Рисунок 2.14 – Перетворення кімнат в полібокси

Починається другий етап зростання – на цьому етапі сторона може поділятися на кілька частин. На відміну від першого етапу, вона росте не по одній клітці за раз, а відразу до максимального упору - це дозволяє уникнути "драбинок" на стиках кімнат. Якщо після проходу по всіх кімнатах все ще залишилися порожні клітини, цикл повторюється.

В результаті виходить повністю заповнений простір (рисунок 2.15):

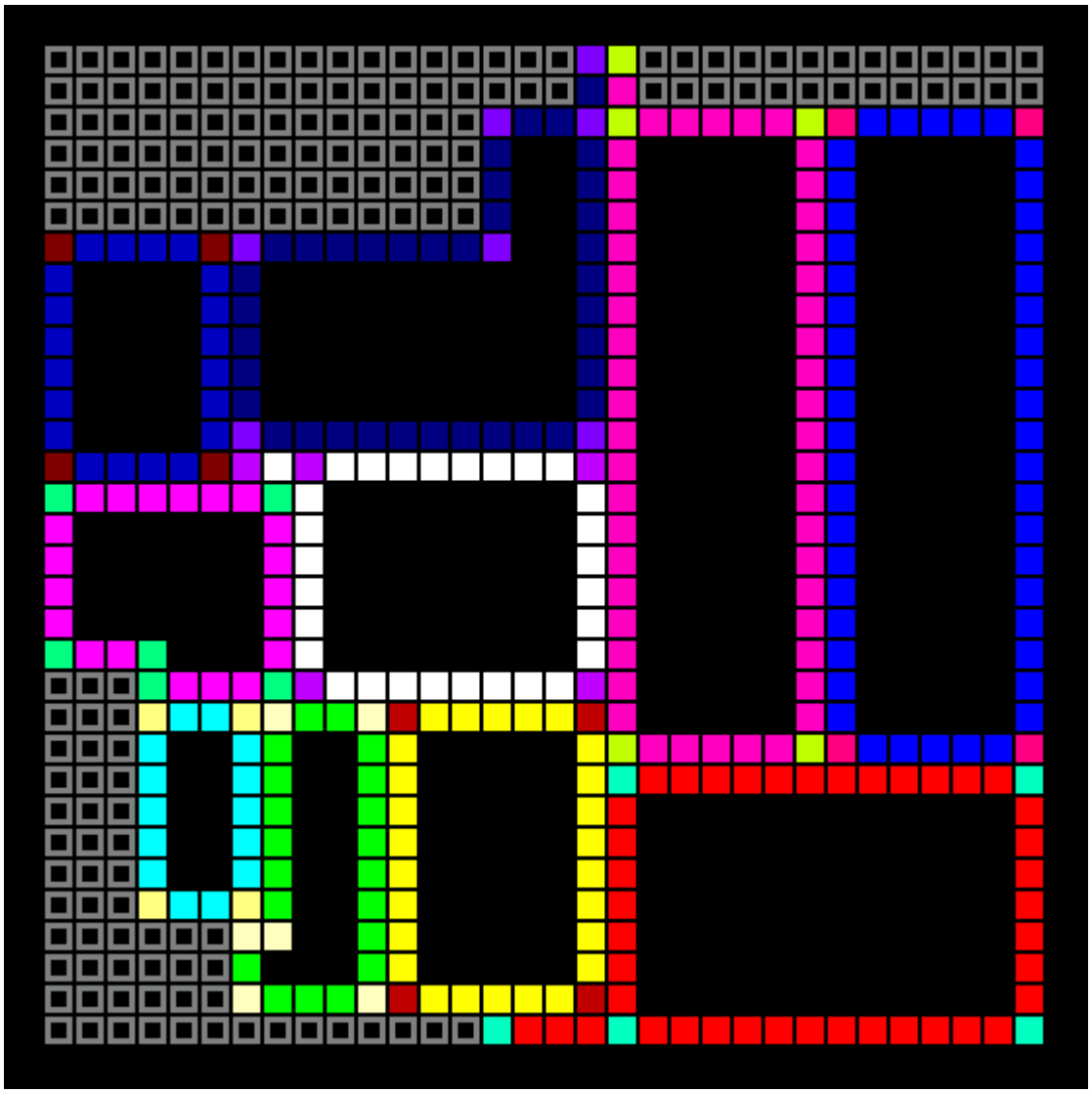


Рисунок 2.15 – Другий етап зростання кімнат

Далі полібокси представляються у вигляді растра, і (як і в разі "BSP" - генераторів) починається етап позначки "сміттєвих" клітин (рисунок 2.16):

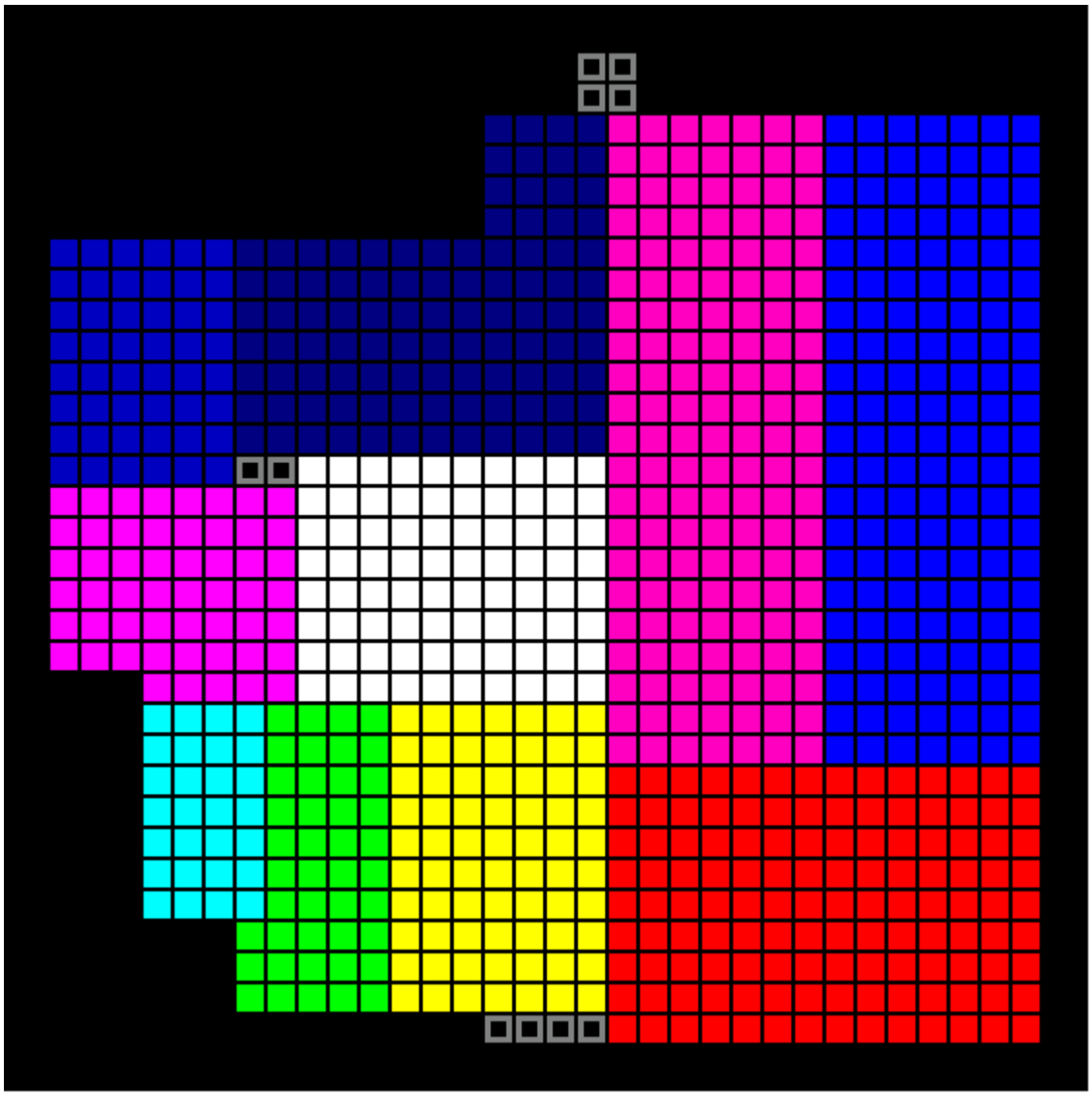


Рисунок 2.16 – Полібокси у вигляді растра

І приєднання їх до існуючих сектораів (рисунок 2.17):

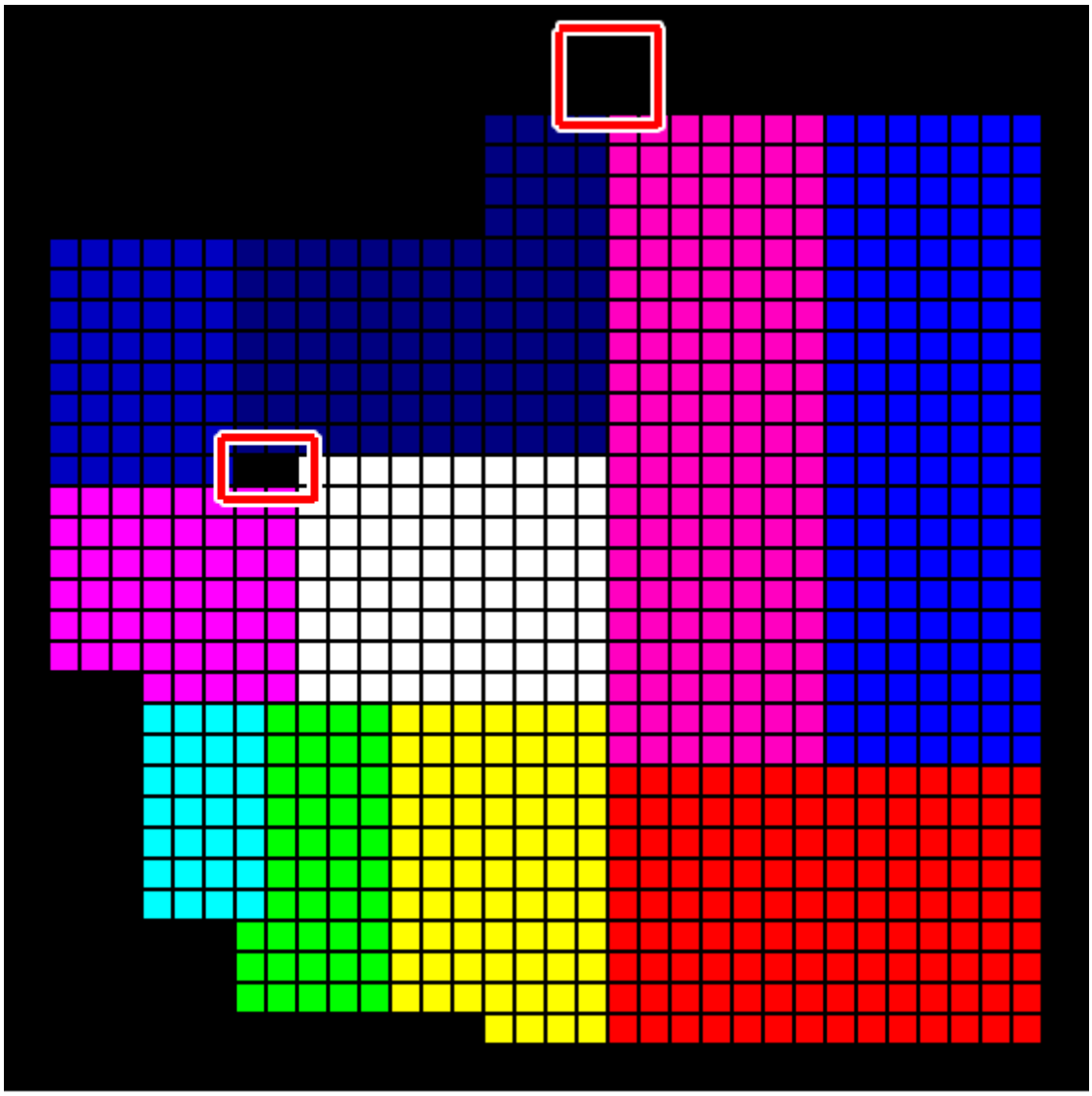


Рисунок 2.17 – Приєднання до існуючих секторів

Результат знову перетворюється в полібокси (рисунок 2.18):

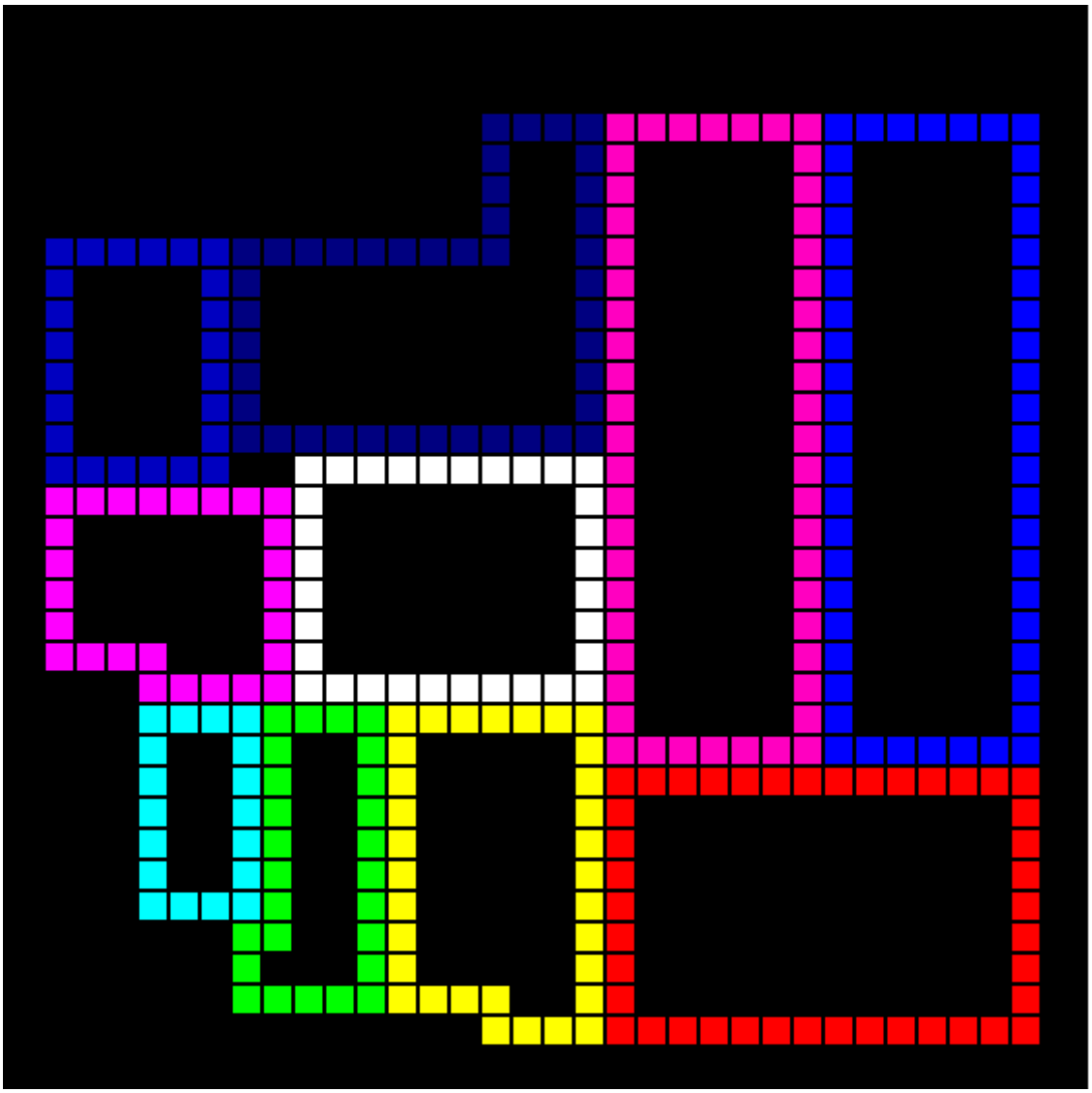


Рисунок 2.18 – Результат перетворений в полібокс

Іноді виникають такі секції, всередині яких не дотримується правило 3x3 (рисунок 2.19):

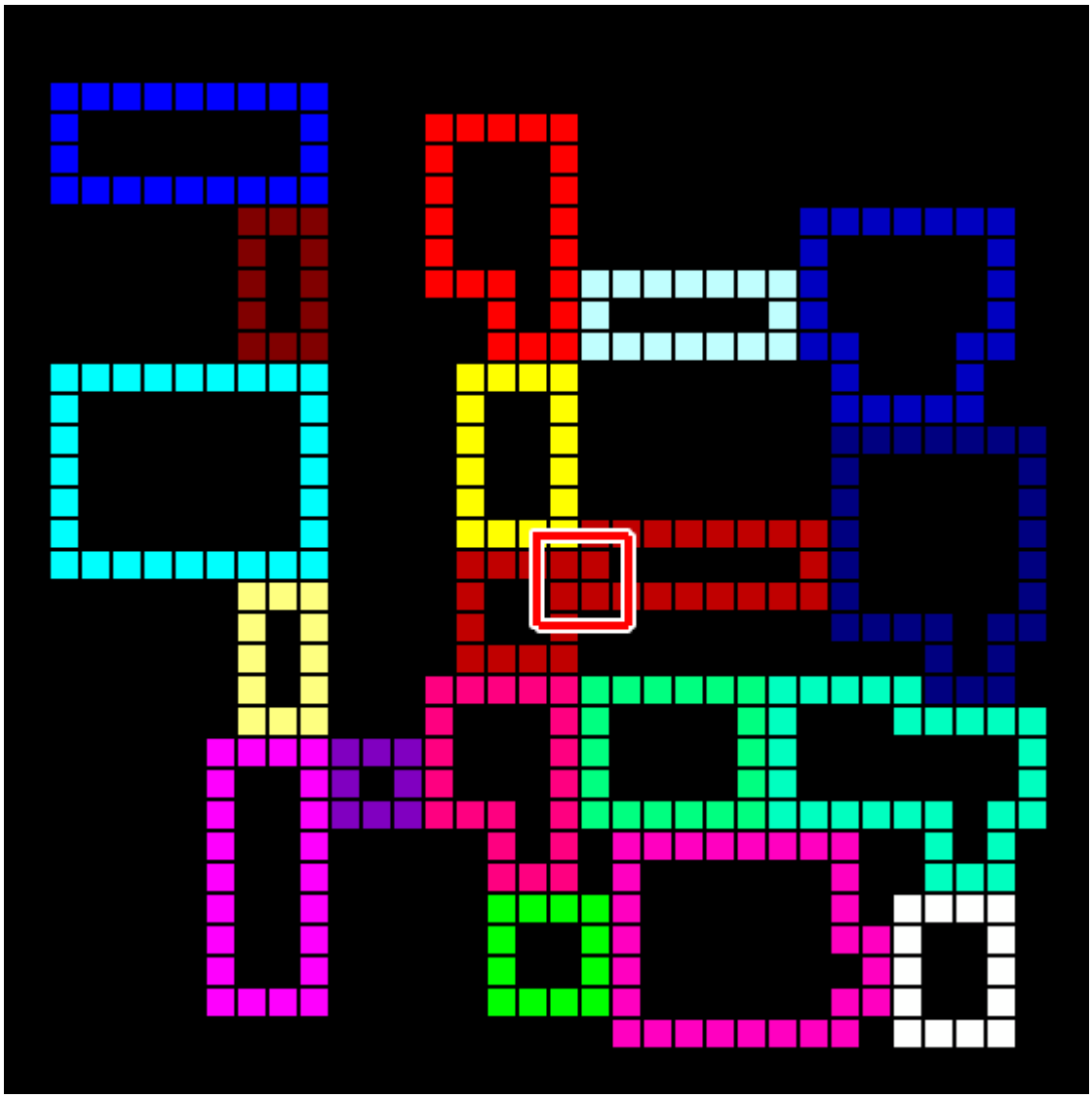


Рисунок 2.19 – Секції, що порушують правило 3x3

Можна спробувати такі секції "відновити", або піти по більш простому шляху, і просто їх видалити (рисунок 2.20):

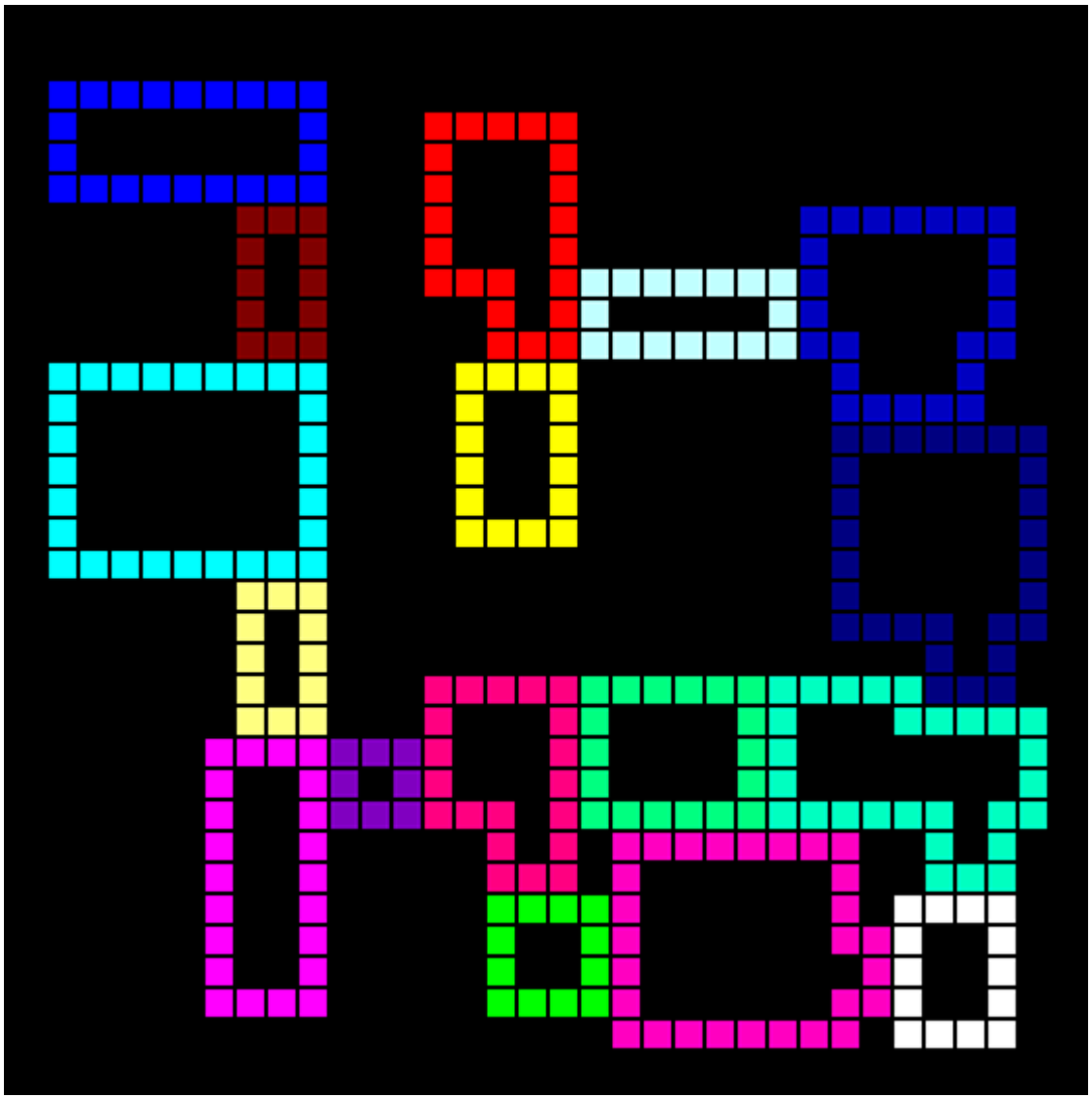


Рисунок 2.20 – Видалення невідповідних секцій

Для кожної секції шукаються її сусіди, і в місцях зіткнення таких секцій створюються з'єднання. З'єднання створюються з обох сторін - якщо секція А стикається з секцією В, то буде з'єднання з А в В і з В в А. В результаті виходить двонаправлений граф.

Іноді, в результаті очищення від сміттєвих секцій, виходить не один граф, а кілька незалежних, як в цьому прикладі (рисунок 2.21):

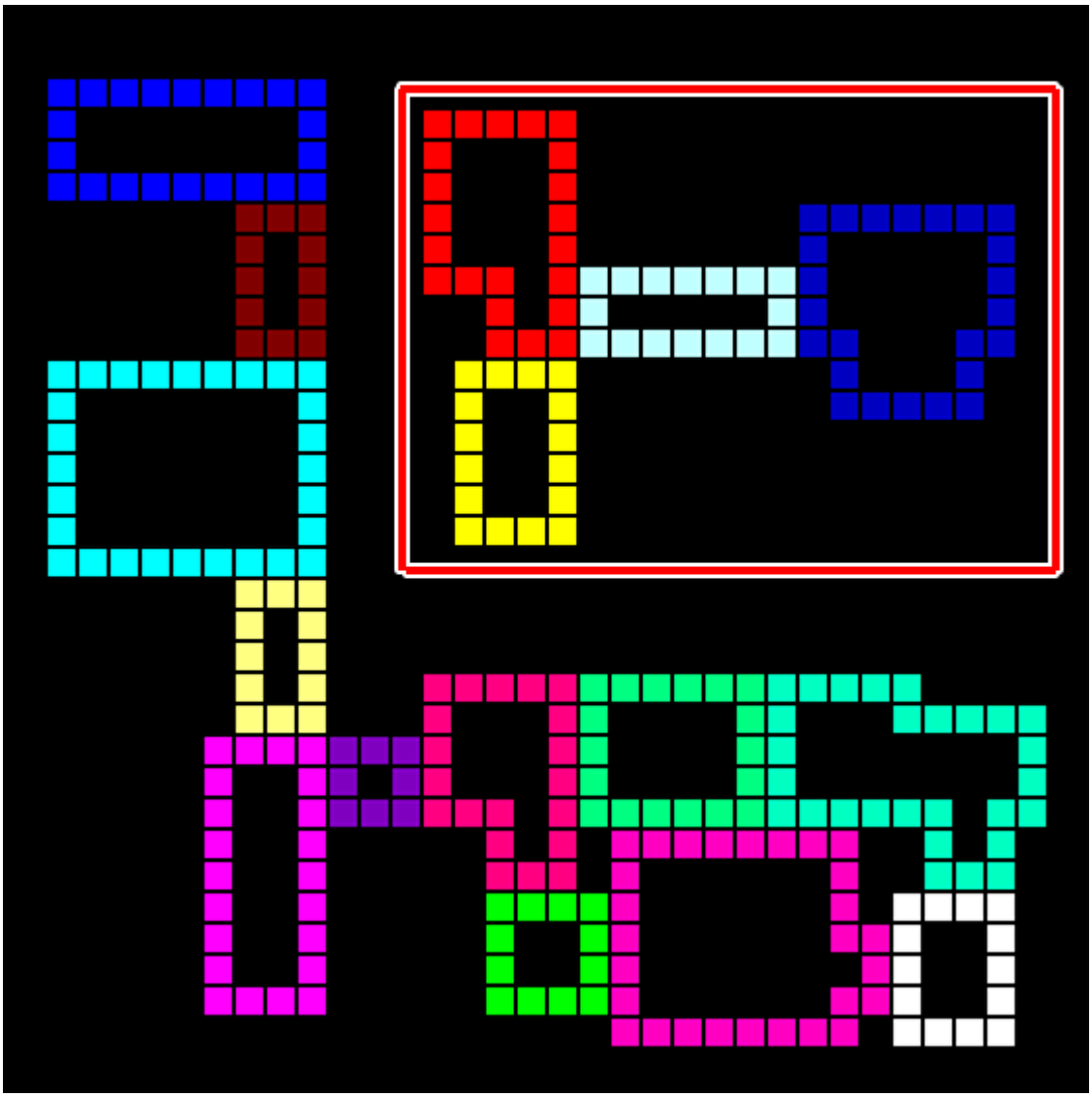


Рисунок 2.21 – Приклад виникнення декількох незалежних графів

В такому випадку в якості основного вибирається підграф, "площа" секцій якого максимальна, а решта видаляються ("площа" взята в лапки, тому що хоч і є можливість порахувати реальну площу полібоксів, я спростив завдання, і вважаю площа прямокутника - це неправильно, але для генератора підходить).

Якщо з кожного сектора буде прохід в кожен, з яким він з'єднується, то на рівні буде занадто багато дверей, і буде сильніше відчуватися що він згенерований. Тому, на цьому етапі зайві сполучення видаляються (рисунок 2.22 і 2.23):

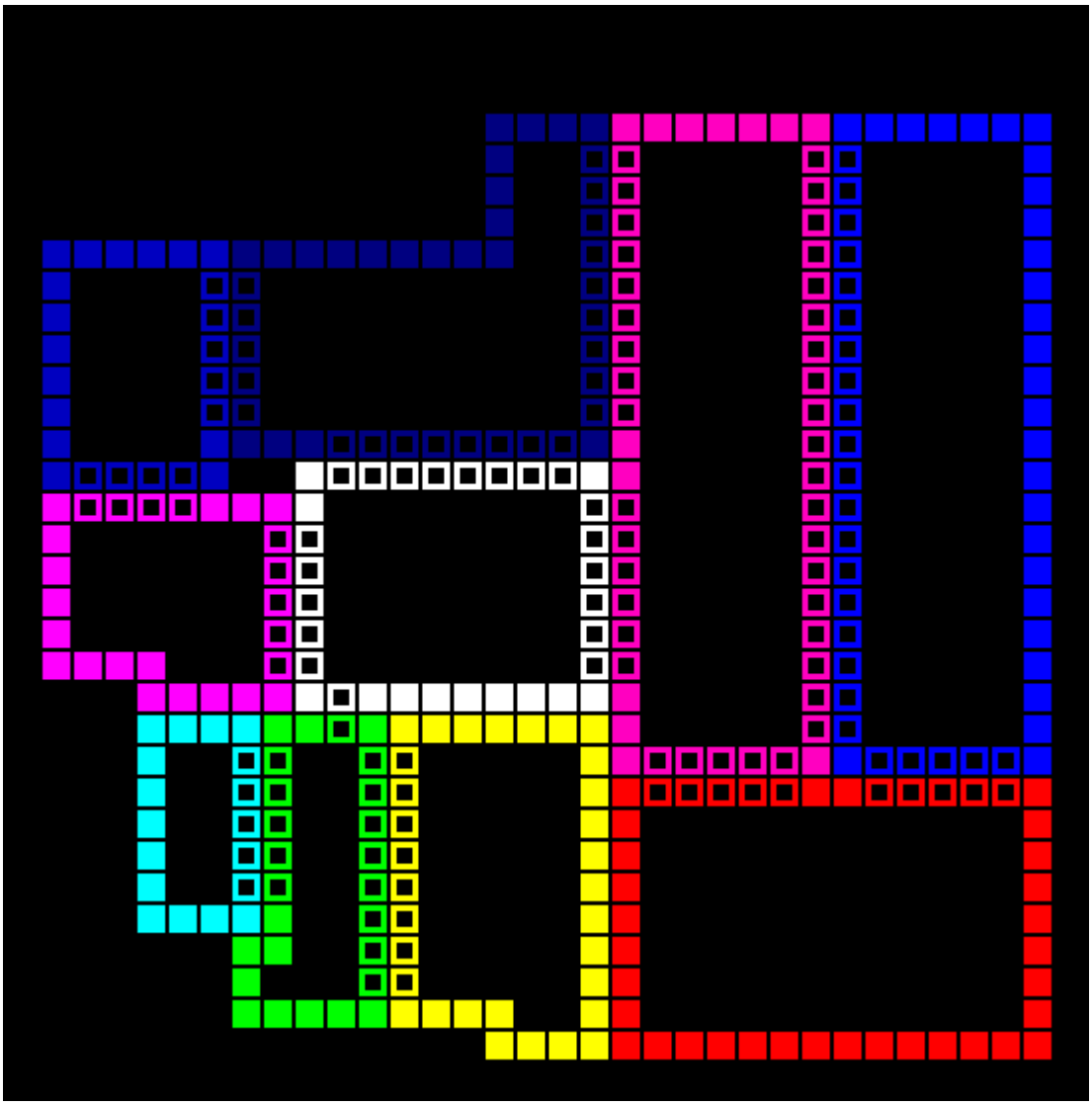


Рисунок 2.22 – Рівень з зайвими сполученнями

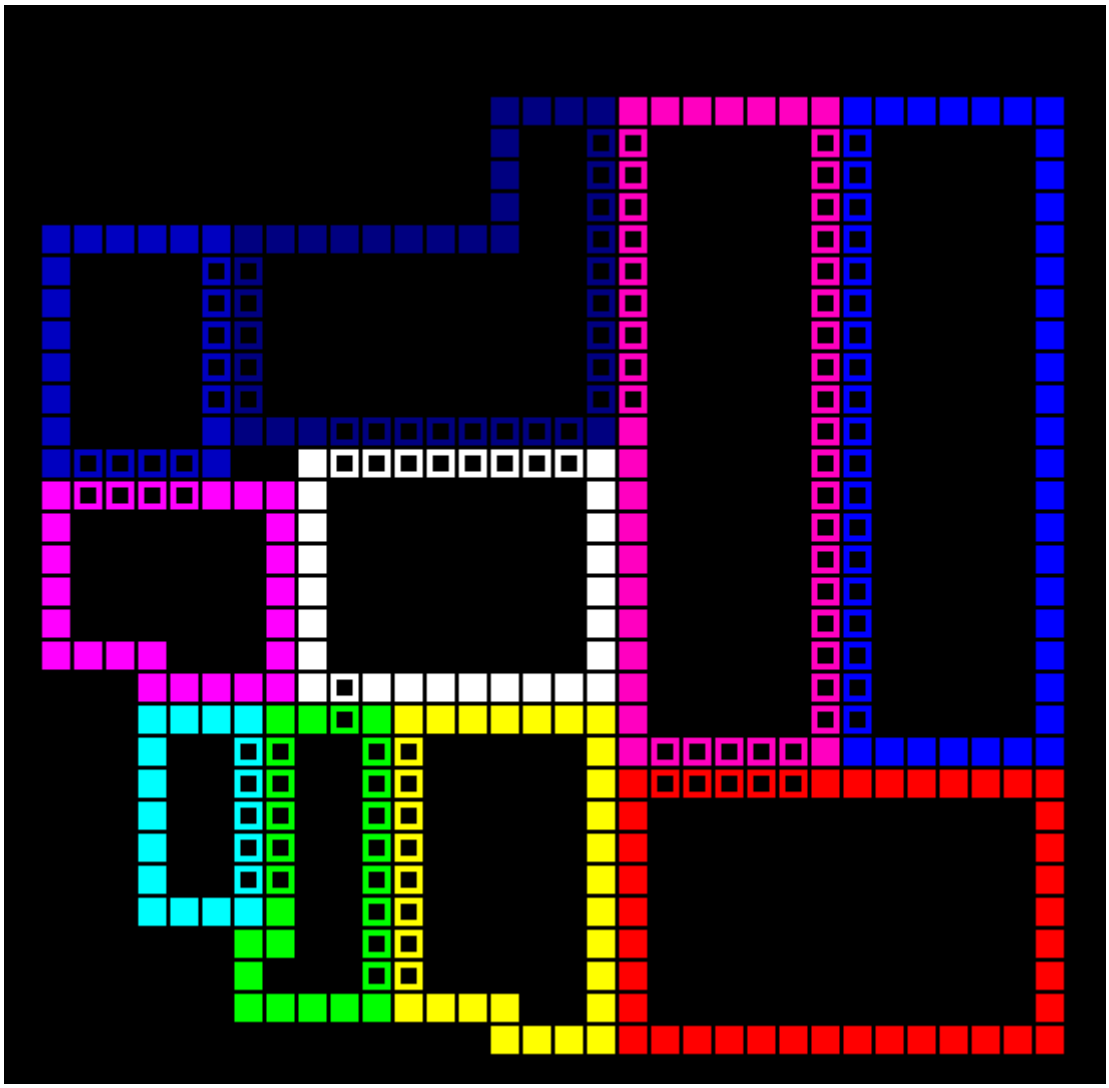


Рисунок 2.23 – Видалено зайві сполучення

Геренація секретних кімнат. На графі вибираються такі сектори, у яких є тільки одне з’єднання (рисунок 2.24):

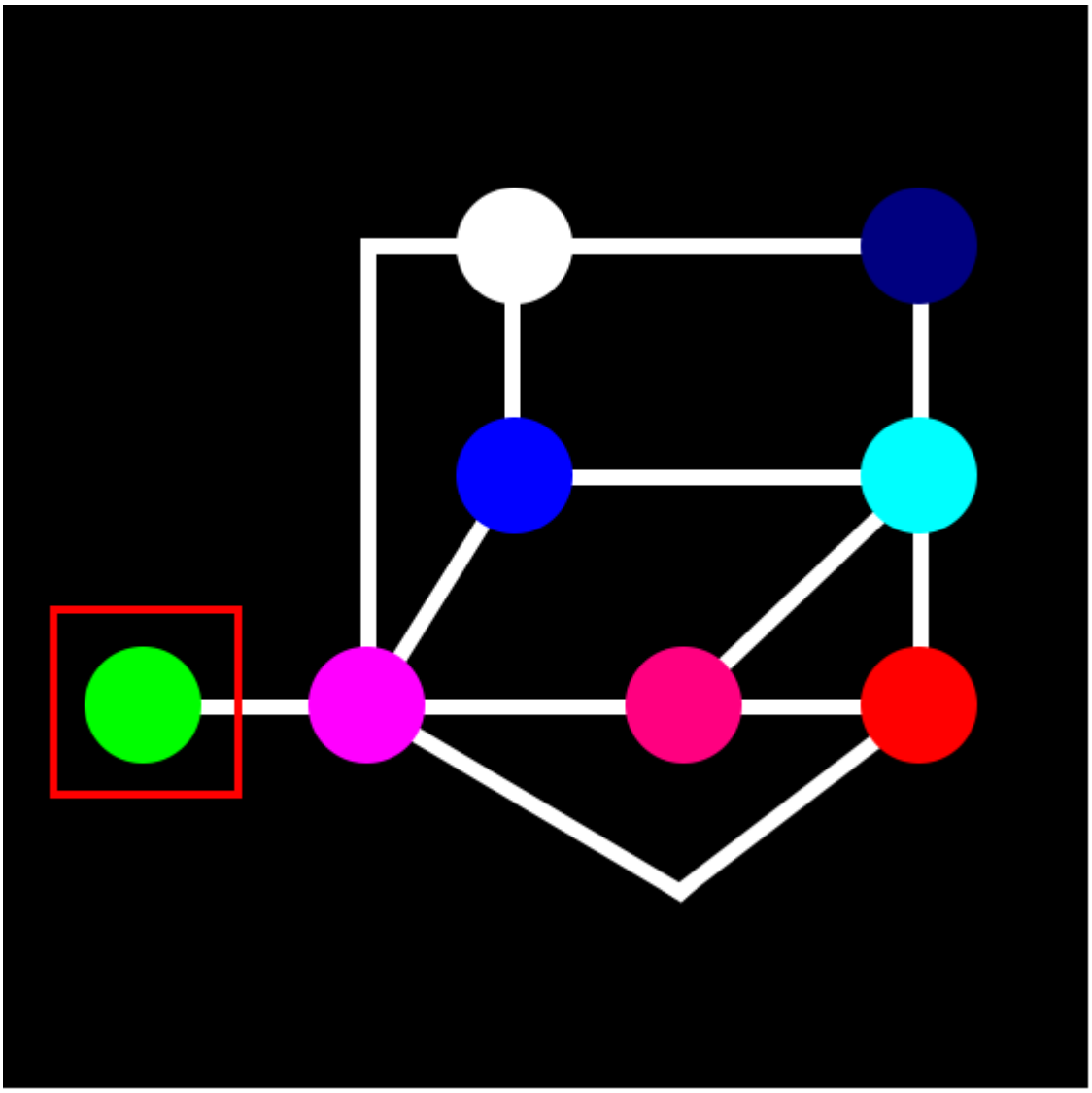


Рисунок 2.24 – Сектор з одним з’єднанням

Якщо таких секторів буде кілька, то вони всі збираються в масив, і упорядковуються відповідно до "площі". Потім цей масив обрізається випадковим чином (але так, щоб в ньому залишився хоч один елемент). Ці сектори і стануть секретними кімнатами (рисунок 2.25):

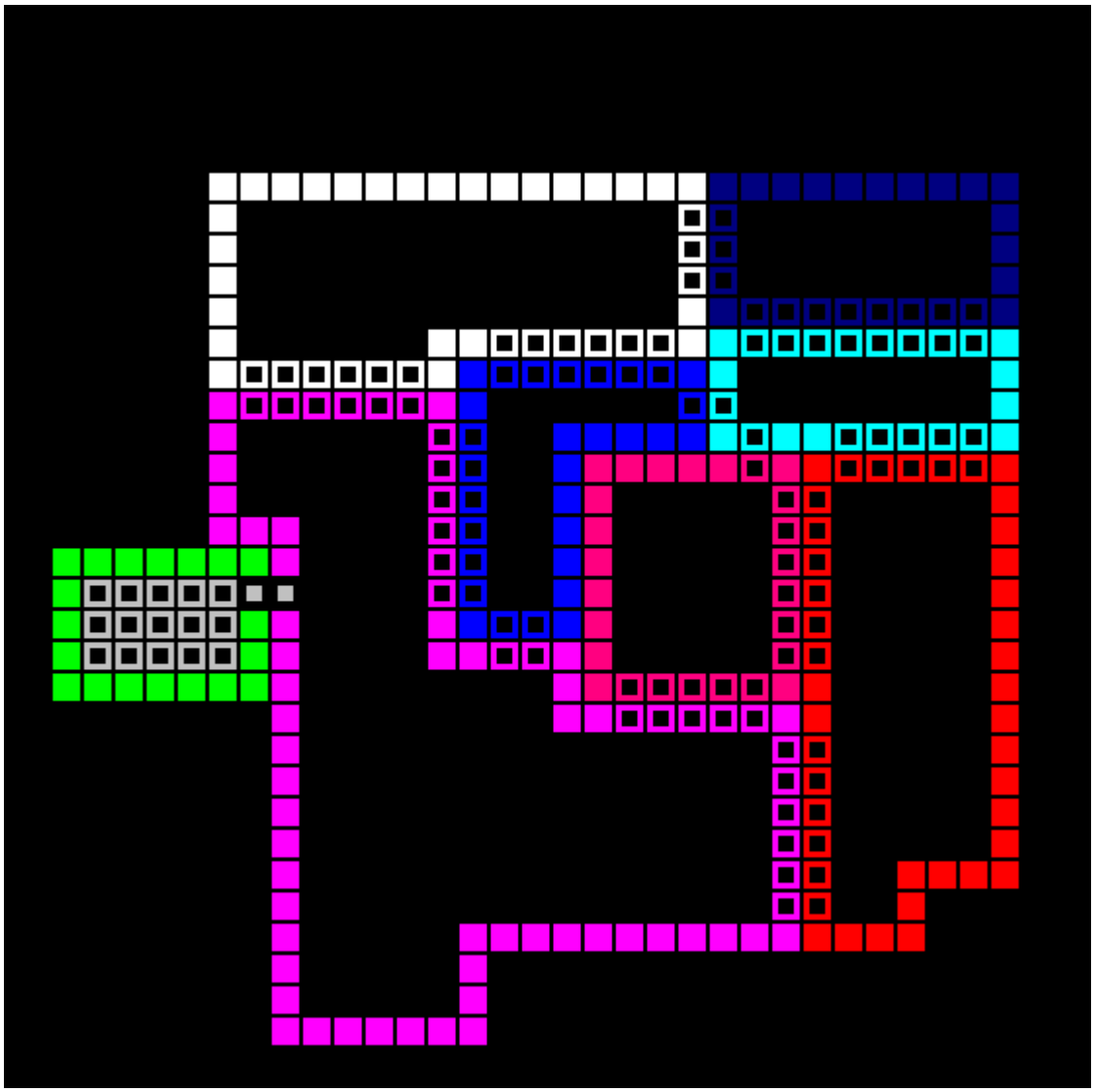


Рисунок 2.25 – Вибір секретної кімнати

2.3 Генерація рівнів за допомогою підготовлених частин

Даний вид генерації досить простий. Його суть полягає в тому, що рівень випадковим чином генерується із частин які були підготовлені заздалегідь. Основними перевагами даного способу є можливість генерації нескінченного рівня та його простота. Недоліками можна вважати необхідність підготовки частин рівня та залежність кількості підготовлених участків на випадковість згенерованого рівня.

Розглянемо даний спосіб на прикладі генерації рівня для Hyper Casual гри HelixJump.

На першому єтапі нам необхідно підготувати декілька різних частин рівня. В нашому випадку рівень складається з платформ(рисунок 2.26 і 2.27).

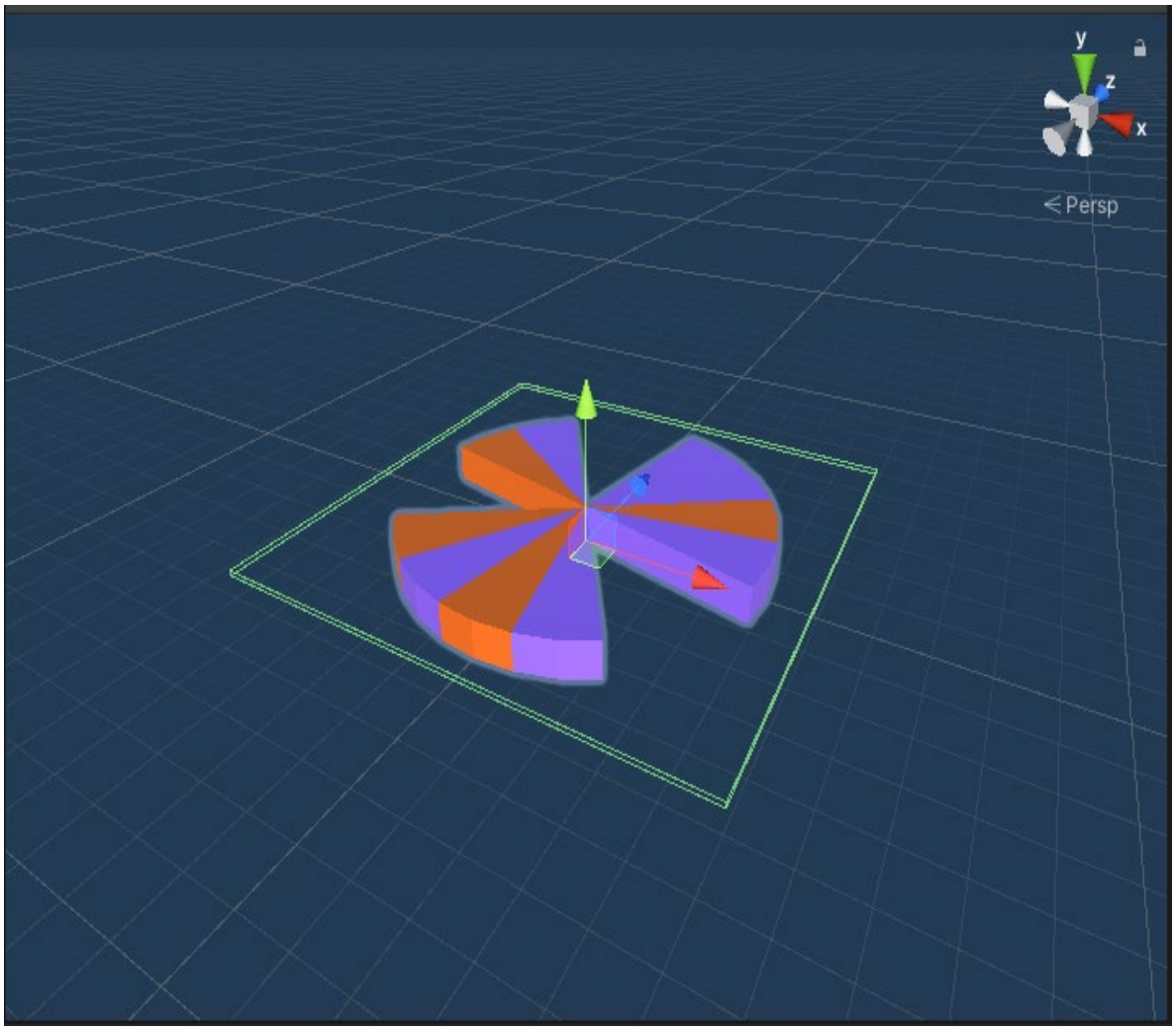


Рисунок 2.26 – Підготовлена платформа

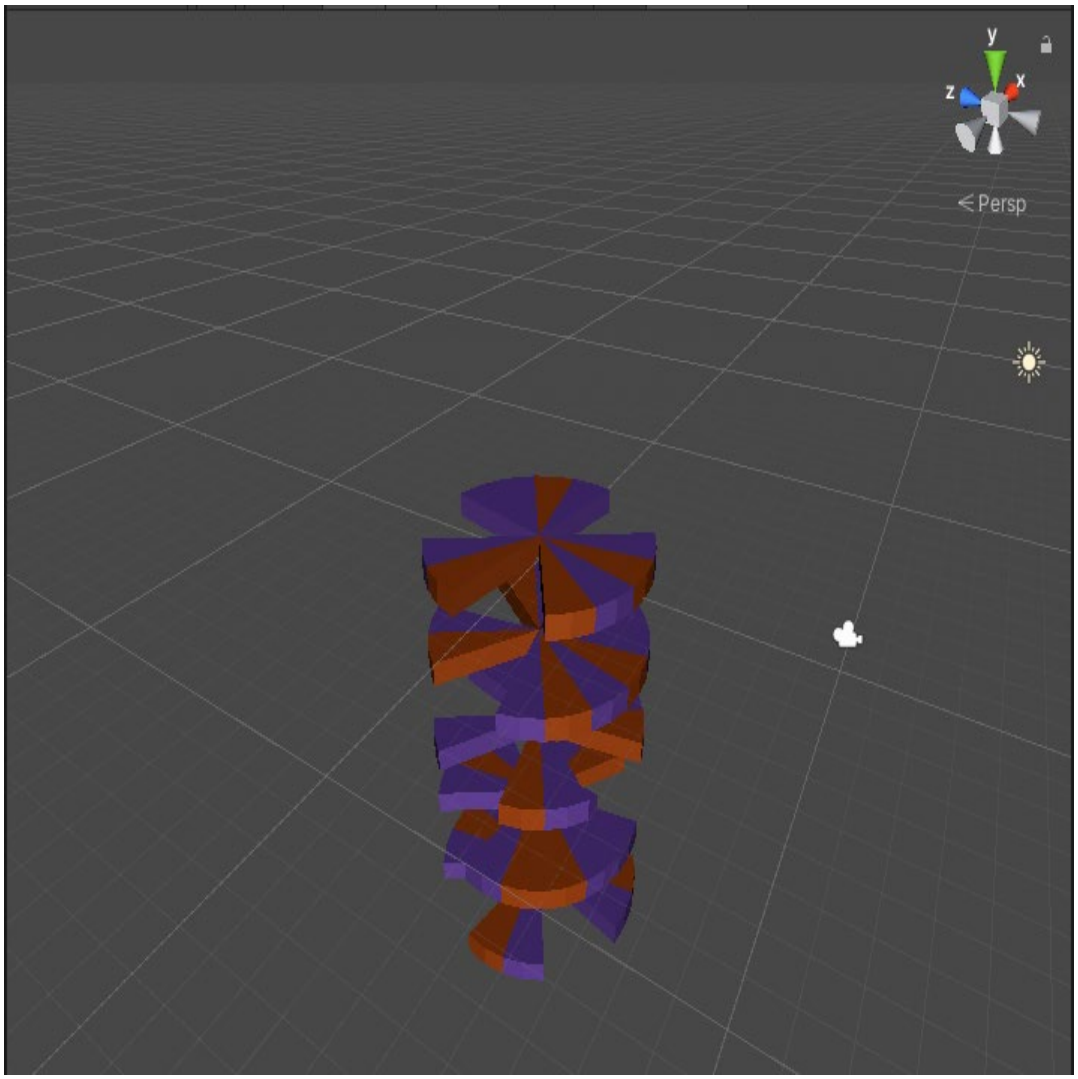


Рисунок 2.27 – Усі підготовлені платформи

На наступному етапі відбувається сама генерація рівня. Випадковим чином обирається одна із заготовлених платформ, створюється її екземпляр, повертається випадковим чином по осі *y* і поміщається на сцену (рисунок 2.28).

```
        for (int i = 50; i > 0; i--)
                {
                    GameObject go = Instantiate(prefabs[Random.Range(1,
5)], new Vector3(0, 0, 0), Quaternion.identity);
                    go.transform.parent = parent;
                    go.transform.position = new Vector3(0, i *
distanseBetweenPlatforms, 0);
```

```
                go.transform.Rotate(new Vector3(0, Random.Range(-
180f, 180f), 0));
            }
```

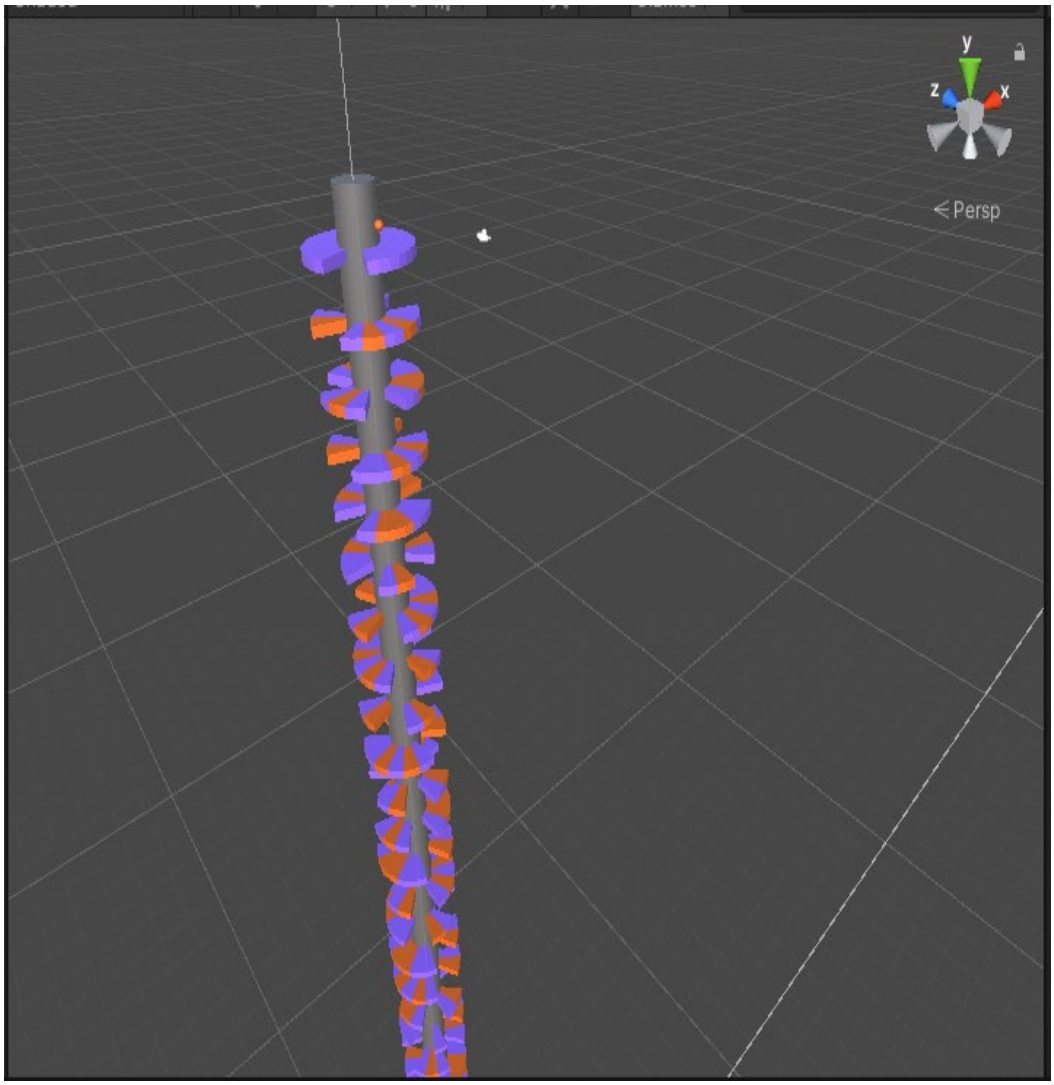


Рисунок 2.28 – Згенерований рівень

Данний спосіб дозволяє нам кожного разу отримувати унікальний рівень. Також він досить зручний для розглянутого випадку, адже всі платформи мають однакову геометрію і їх легко помістити на потрібну позицію. Достатньо задати для генератора відстань між платформами. Для більш складних випадків кожна підготовлена частина повинна буде зберігати данні про координати «входів» та «виходів» та передавати їх генератору для правильного розміщення. Але навіть для таких випадків спосіб залишається відносно простим та швидким у використанні.

3 РОЗРОБКА МОДИФІКОВАНОГО МЕТОДУ ГЕНЕРАЦІЇ РІВНІВ

3.1 Пул об’єктів

Алгоритми розглянуті в пункатах 2.1.1 і 2.1.2 дають змогу генерувати кожного разу випадковий рівень. Але обидва мають спільнний недолік. Для генерації досить великого рівня вони є ресурсозатратними, адже весь рівень генерується в один єтап на початку роботи програми. Також незважаючи на можливість генерації нескінченного рівня, данні способи також не будуть оптимізованими, адже виникне необхідність видаляти старі та створювати нові частини рівня прямо під час гри. В залежності від складності рівня це може спричинити стрімке падіння частоти кадрів під час «додавання» нової частини рівня.

Для вирішення даної проблеми я пропоную скористатися таким шаблоном проектування як Пул об'єктів (Object pool).

Пул об'єктів призначений для зберігання готових до використання об'єктів. Коли системі потрібен новий об'єкт, він запитує з Пула, минаючи процес породження. А після використання повертається назад в Пул замість знищення.

Шаблон застосовується для підвищення продуктивності, якщо:

- об'єкти часто створюються і знищуються;в системі;
- існує обмежена кількість об'єктів типу, що зберігається в пулі;
- створення і / або знищення об'єкта є дуже витратними операціями.

Пул об'єктів може працювати як з інтерфейсами, так і з конкретними реалізаціями. Все залежить від архітектури розробки і вирішуваних завдань.

Можна зустріти спільне використання Пула об'єктів та інших породжуючих шаблонів. Наприклад, для створення об'єктів в певному стані може застосувати Прототип. А за допомогою Singleton - створити єдиний екземпляр Пула в системі.

Поганою практикою є приховування Пула за іншими породжуючими шаблонами. Розробник, який використовує такий "гібрид", не очікує вимоги повернення об'єктів від, наприклад, Фабричного методу. А без повернення об'єктів сам Пул стає марним. В такому випадку, правильним рішенням буде відокремити реалізацію класів, що створюють об'єкти.

Особливості застосування:

- Пул нічого не знає про реалізацію збережених об'єктів. Тому повернутий об'єкт вважається які у невизначеному стані. Для подальшого використання його необхідно перевести в початковий стан (скинути). Наявність об'єктів в невизначеному стані перетворює Пул в "об'єктну клоаку" (object cesspool);
- Повторне використання може стати причиною витоку конфіденційної інформації. Тому необхідно обов'язково очищати поля з секретними даними при скиданні, а самі дані - затирати або знищувати;
- Можлива ситуація, коли в Пулі не залишиться вільних об'єктів. У цьому випадку реакція на запит може бути наступна:
  - збільшення розміру пулу;
  - відмова в видачі об'єкта;
  - постановка в чергу і очікування звільнення об'єкта.

Реалізація шаблона в загальному вигляді:

- визначаємо інтерфейс IPoolable, який повинні реалізовувати об'єкти для взаємодії з Пулом;
- розробляємо архітектуру роботи Пула з об'єктами, включаючи: їх створення, зберігання та видалення;
- скидання в початковий стан при поверненні, використовуючи інтерфейс IPoolable;
- реакцію на відсутність вільних об'єктів;
- реалізуємо Пул;
- в клієнтському коді для отримання об'єкта звертаємося до Пулу, а після використання обов'язково повертаємо об'єкт назад.

Особливості реалізації в C#:

Для початку згадаємо деякі факти про роботу з пам'яттю в .NET:

Прибирання сміття в .NET відбувається по поколінням. І чим старше покоління, тим рідше в ньому відбувається прибирання сміття.

Об'єкти, за винятком великих, створюються в нульовому поколінні. Вони переходять в наступне, якщо виживають після чергової збірки сміття.

Чим більше буде інтенсивність створення об'єктів, тим частіше будуть відбуватися збірки сміття.

Повернемося до Пулу. Він як раз використовується у випадках, коли об'єкти створюються часто. Це не обов'язково відбувається в тілі циклу, як в прикладі вище. Породження можуть відбуватися в різних методах класу або навіть різних класів, виконуватися в різних потоках і т.д. Важливо тільки те, що вони відбуваються досить часто.

Якщо в такій ситуації використовувати звичайне створення об'єктів, то молодше покоління буде накопичувати велику кількість сміття. Це призведе до збільшення числа запусків збирача. Крім того, частина об'єктів встигнуть перейти в старші покоління. У підсумку - нехай невелике, але зменшення продуктивності і збільшення витрат пам'яті.

При коректній роботі з Пулом зменшується число породжуваних об'єктів. Крім того, вони самі можуть бути спроектовані так, щоб при скиданні стану уникнути повторного створення примірників потрібних їм об'єктів. Як наслідок - зменшення витрати пам'яті, сміття, періодичності потрібно збирача [6].

3.2 Вибір інструмента реалізації

На данний момент існує дуже багато варіантів вибору інструмента для реалізації поставленної задачі. Основним критерієм має бути відностна простота застосування і кросплатформність.

Unity3D - кросплатформенний ігровий движок від компанії Unity Technologies. Історія створення движка досить цікава і повчальна. Цікава, бо двоє хлопців захотіли зробити гру, але для цього їм не підходили існуючі

інструменти. І вони вирішили зробити свій движок, а потім вже робити на ньому гру. І після того, як вони зробили движок, вони зрозуміли, що їм не так-то й цікаво робити ігри, а більше подобається займатися безпосередньо движком. Так і почалася історія одного з найвідоміших і потужних движків. А повчальна ця історія тому, що ніколи не знаєш, чим обернеться те чи інше починання.

На рисунку 3.1 зображено логотип Unity.

Рисунок 3.1 – Логотип Unity

Unity - безкоштовний движок. Обмеження - при запуску гри показується логотип Unity. Купивши розширену версію, ви позбудетеся і від логотипу. Безкоштовність движка - це те, що привернуло багатьох до розробки ігор на ньому.

Unity використовує компонентно-орієнтований підхід. Все в грі - це об'єкт, куди додані різні компоненти. Наприклад, якщо ми робимо платформер, ми додаємо GameObject, і до цього GameObject додаємо графічний компонент (для відтворення гравця) і компонент управління (щоб можна було управляти гравцем клавіатурою або мишкою). Таких різних компонентів можна додати будь-яку кількість до будь-якого GameObject.

Тобто, створення гри в Unity - це додавання GameObject-ів, і додавання їм корисних компонентів.

Нехай вас не зводить уявна простота цього процесу. Щоб зробити щось нетривіальне, вам доведеться писати свої компоненти. У термінах Юніті вони називаються скриптами. Пишуться вони на мові C #. Написання своїх компонентів - це досить складне заняття. Фактично, це звичайне програмування. Так що без знання будь-якої мови програмування вам доведеться туго.

Unity - хороший вибір для створення середніх по складності проектів як для ПК, так і для мобільних пристроїв. Велика кількість готових Ассет, включаючи скрипти, дуже цьому допомагає. Ну і велика спільнота - це теж добре, вам допоможуть вирішити будь-якої затикаючи, якщо він виникне.

Якщо ж ви робите маленький проект - будь-якої клікер або щось на зразок цього - задумайтеся, можливо, Unity буде занадто великим монстром для цього.

Ну і якщо ви робите величезний проект AAA-класу, Unity теж може бути проблемою. Сама по собі ідея зі скриптами хороша, але досить повільна. Ну і мову C # - інтерпретована. Незважаючи на всі JIT-оптимізації, він повільніше за який-небудь C ++. Сотні об'єктів на сцені з складними компонентами можуть вбити продуктивність. Тому для величезних проектів хорошим вибором може стати CryEngine, наприклад.

Втім, все залежить від програміста. Тямущі люди створювали на Unity великі проекти з хорошою продуктивністю. Правда, їм доводилося багато чого прелаштовувати в движку під себе. Так що, роблячи щось грандіозне, будьте готові загрузнути в деталях движка. Дійсно великим командам за окремі гроші Unity Techologies надає вихідний код Unity, так що на крайній випадок можна попорпатися в вихідному коді, і щось там поправити. Але знову ж таки - все це має сенс лише в разі величезних проектів.

Один з козирів Юніті - це список підтримуваних платформ, де можуть запускатися програми. Unity працює майже всюди - на ПК (всі операційні

системи), на Андроїд, на iOS, на SmartTV, в браузері, на різних екзотичних системах - наприклад, Tizen OS. Правда, не обійшлося без ложки дьогтю. Якщо ви працюєте з чимось специфічним, наприклад, низькорівневий доступ до заліза в тому ж Андроїд - будьте готові писати частину коду на Java, потім компонувати все це з Юніті. Аналогічно з iOS. Також, зібрати додаток під iOS можна лише з-під MacOS X. Тобто, не маючи макбуков або чогось схожого, гру на iOS ви так просто не випустіть. Це не недолік Юніті, це обмеження Apple. Але ситуація саме така. Так що в разі, якщо ви орієнтуєтеся на iOS - подбайте про середовищі збірки вашої гри.

Потужний плюс Юніті - це Ассети. Все в грі, включаючи код, картинки, представляється Ассетом (Asset). Ассет можна експортувати, імпортувати. Таким чином, сторонні розробники можуть робити цілі заготовки для ігор. Все, що вам залишиться - це замінити картинки, підправити скрипти - і можна релізі гру. Знову-таки, все не так просто, диявол криється в деталях. Різні Ассет можуть бути несумісні між собою як в прямому сенсі, так і не підходити по стилю. Але це вже деталі.

Є спеціальний онлайн-магазин - Unity Asset Store. Там продаються готові Ассет від сторонніх розробників. Будь-який бажаючий може зробити свій Ассет, і викласти його в продаж в цьому магазині. Деякі люди зробили на цьому цілий бізнес, завдяки великому ринку Unity-користувачів. Також важливий момент, що магазин доступний прямо з редактора Unity. Тобто, додавання нових Ассет максімлально спрощується. Ви заходите в магазин, клікаєте на потрібний Ассет, і він відразу закачується і додається в ваш поточний проект. Швидко і зручно.

Наступною вагомою особливістю є ком'юніті. Воно величезне. Якщо у вас є якесь питання, скоріше за все, воно вже багато разів ставилося, і стільки ж раз вже було вирішене. Пошукайте на профільних форумах, на StackOverflow. Почитайте приватні блоги людей, що пишуть ігри на цьому движку. Інформації просто море. А якщо ж знайшовся питання, на який ви не знайшли відповіді - задайте його на офіційному форумі Юніті, і з великою

ймовірністю ви отримаєте відповідь в той же день. Це величезний плюс движка в порівнянні з іншими[7].

Враховуючи все вище перечислене для реалізації було обрано ігровий движок Unity 3D.

3.3 Реалізація модифікованого методу

Реалізацію я почав зі створення об’єкту Object Pooler, який буде управляти Пулами.

На рисунку 3.2 зображено створенний мною Object Pooler який буде оперувати п’ятьма пулами платформ. Для кожного пула вказано тип платформи за яку він буде відповідати. Кожен пул на початковому єтапі буде містити п’ять екземплярів кожної платформи.

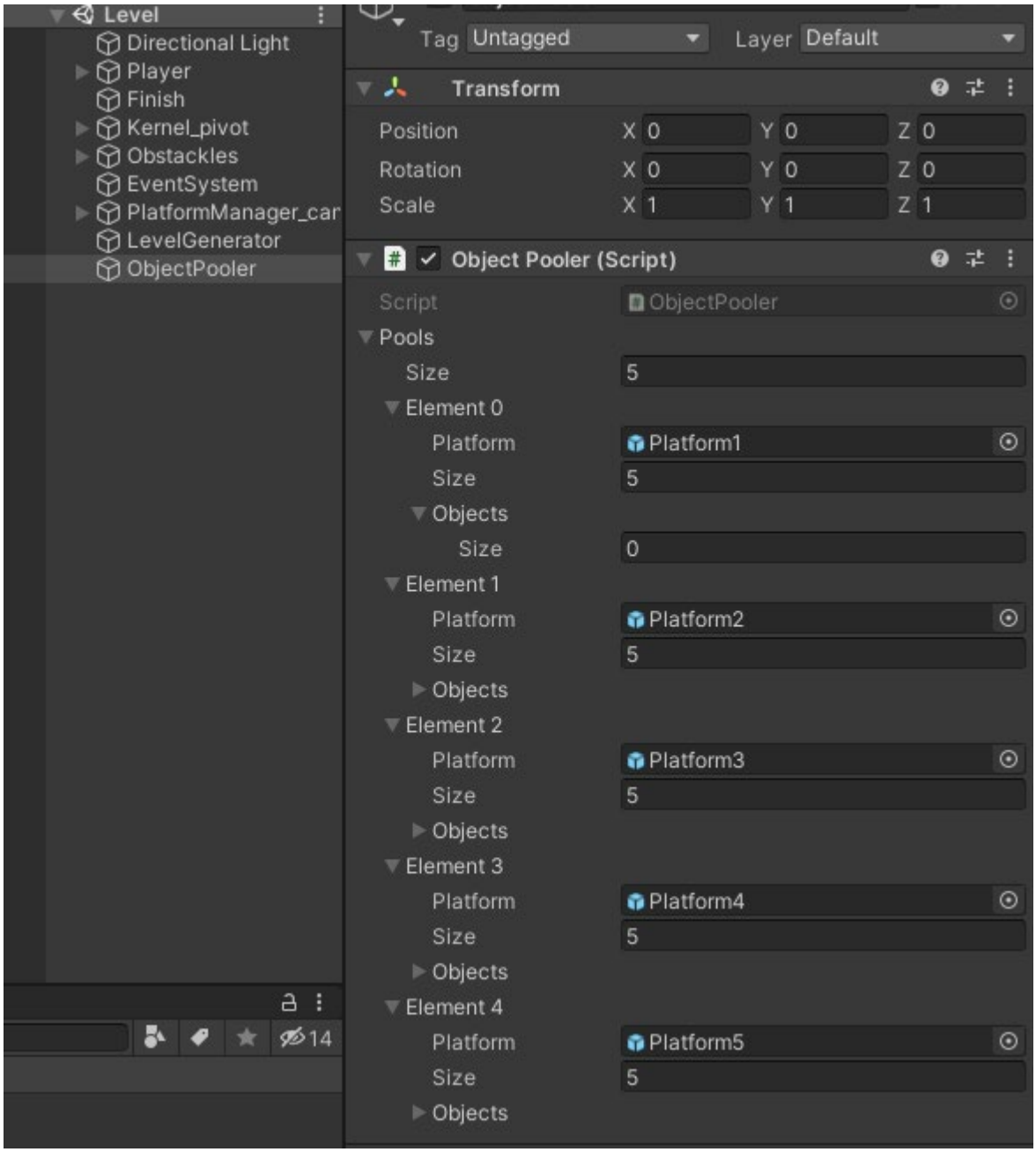


Рисунок 3.2 – Пул об’єктів на сцені в Unity

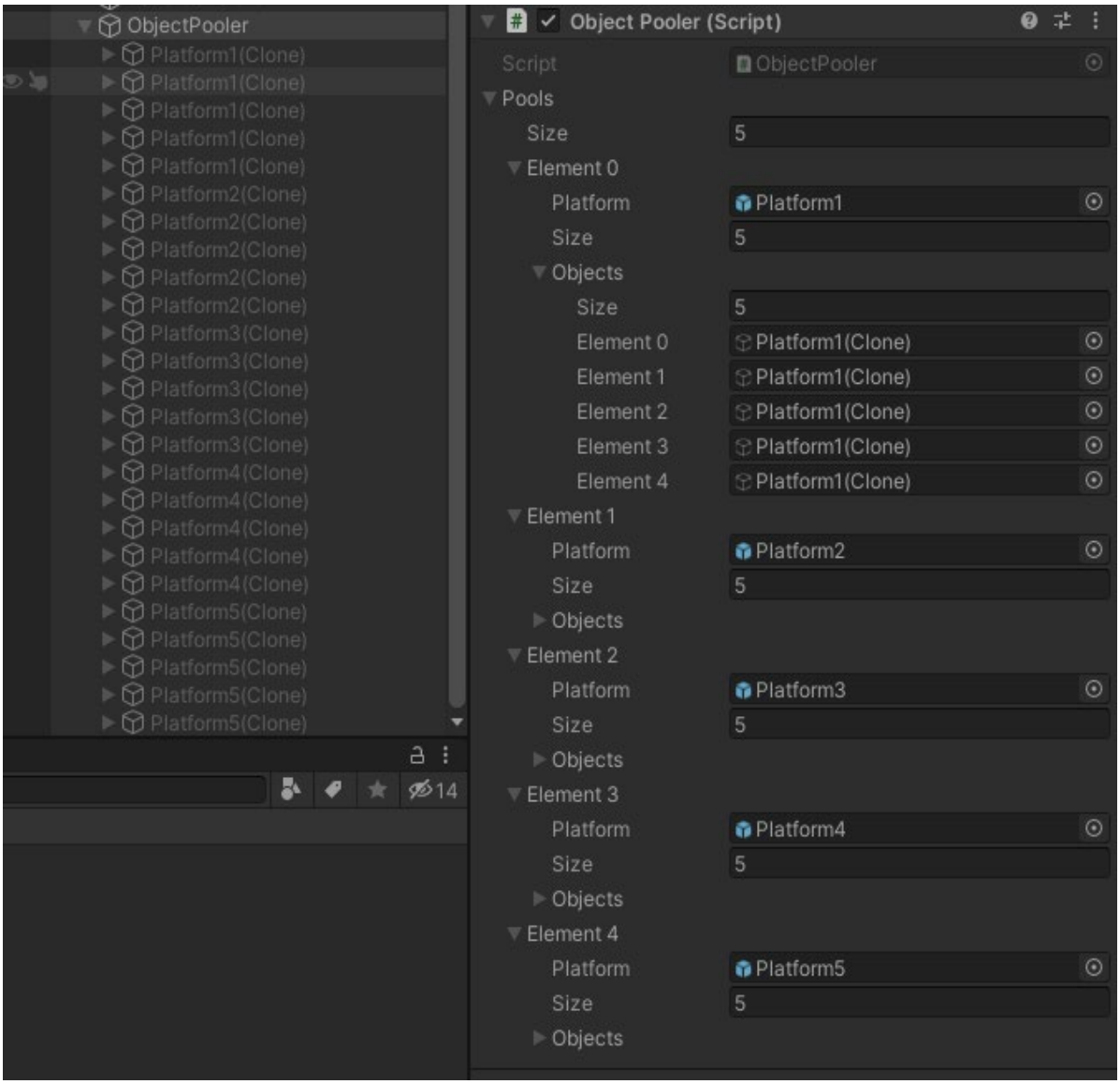


Рисунок 3.3 – Ініціалізація Пула об’єктів

На рисунку 3.3 показано інізіалізацію Пулів в момент запуску сцени. Для кожного пула створена потрібна кількість екземплярів, які потім будуть використовуватися для генерації рівня.

```
private void Initialize()
    {
        for (int i = 0; i < pools.Count; i++)
        {
            for (int j = 0; j < pools[i].size; j++)
            {
                var tmp = Instantiate(pools[i].platform);
                tmp.transform.parent = transform;
                tmp.transform.localScale = transform.localScale;
                tmp.transform.rotation = transform.rotation;
```

```
                tmp.gameObject.SetActive(false);
                pools[i].objects.Add(tmp);
            }
        }
          }
```

На даному етапі просто створюються екземпляри, задається їх компонент transform, що відповідає за розміщення об’єкту на сцені. Потім кожен екземпляр відключається і стає доступним для виклику в будь-який момент.

Наступний участок коду відповідає за надання Пулом потрібного об’єкту:

```
public GameObject GetObjectFromPool(int poolNumber)
    {
        var freeObject = pools[poolNumber].objects.Where(item =>
item.activeSelf == false).FirstOrDefault();
        if (freeObject == null)
        {
            var tmp = Instantiate(pools[poolNumber].platform);
            tmp.transform.parent = transform;
            tmp.transform.localScale = transform.localScale;
            tmp.transform.rotation = transform.rotation;
            tmp.gameObject.SetActive(false);
            pools[poolNumber].objects.Add(tmp);
            return tmp;
        }
        return freeObject;
          }
```

Таким чином при недостачі об’єктів в конкретному пулі буде створюватись новий екземпляр і функція буде повертати саме його.

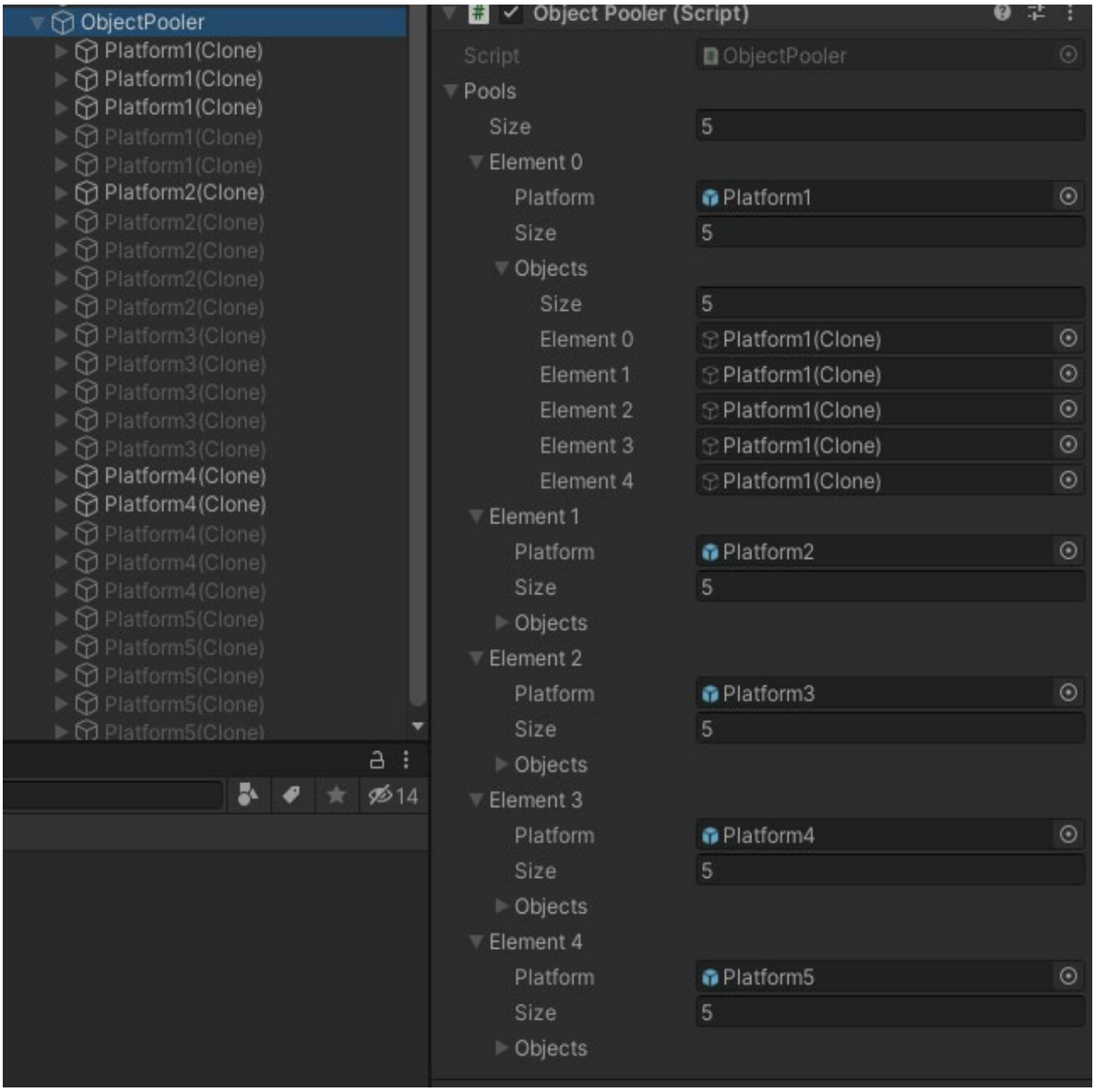


Рисунок 3.4 – Object Pooler під час генерації рівня

На рисунку 3.4 бачимо, що під час генерації рівня використовуються лише обрані випадковим чином платформи, інші зннаходяться у вимкненому стані і є доступними для використання генератором рівня.

На рисунку 3.5 показано як виглядає сцена під час динамічної генерації рівня.

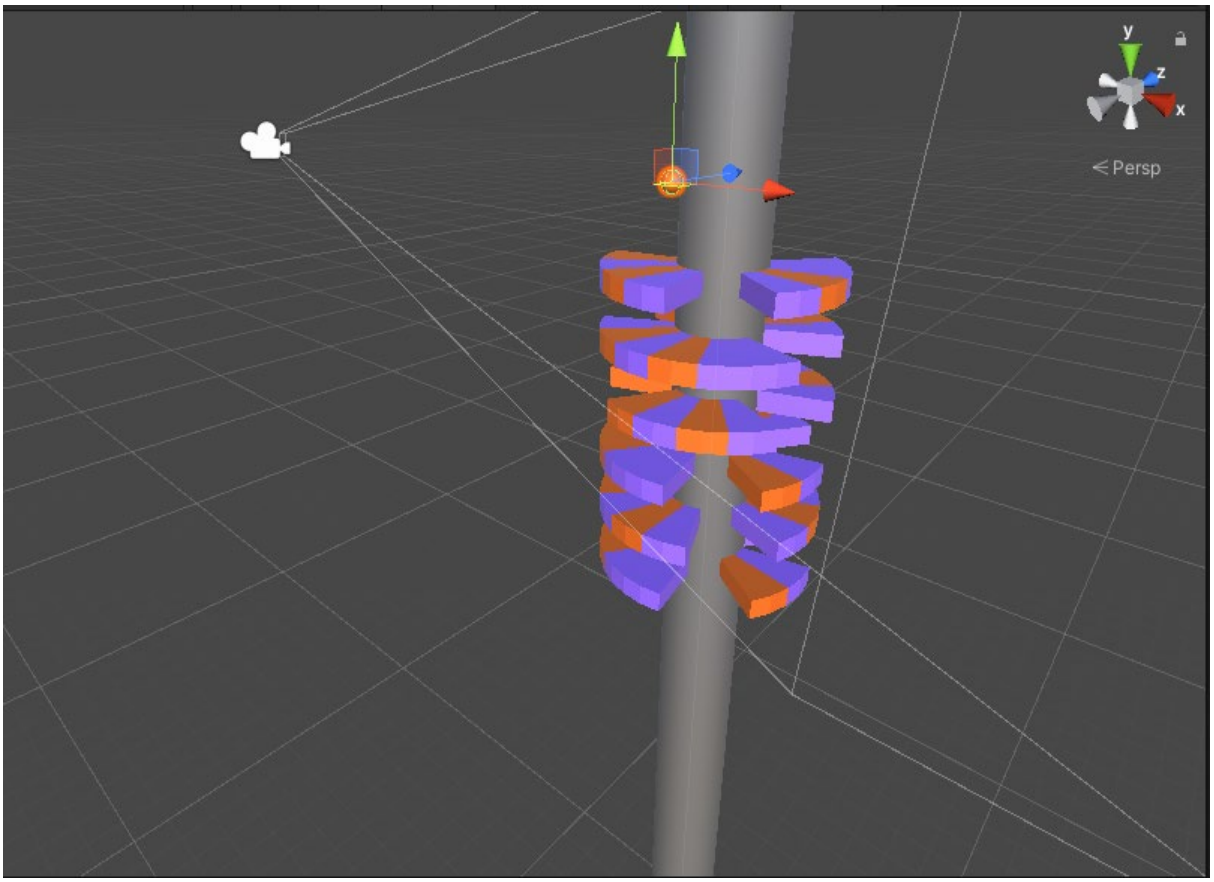


Рисунок 3.5 – Генерація рівня з використанням Object Pooler

В данному випадку платформа повертається в Пул кожен раз коли шар опиняється нижче неї. При її поверненні в наступну позицію одразу ставиться нова. Повернення в Пул означає не видалення конкретної платформи, а її вимкнення на сцені, що робить платформу знову доступною для використання. Таким чином зникає необхідність постійного створення нових платформ, замість цього генератор буде використовувати вже створені, доступні із Пула платформи.

Даний спосіб дозволяє сильно підняти продуктивність гри, адже відпала потреба в ресурсозатратних операціях динамічного створення та видалення платформ. Особливо ефективним даний модифікований метод буде при потребі генерації відносно довгих або нескінченних рівнів.

ВИСНОВОК

В данній атестаційній роботі було досліджено основні існуючі алгоритми процедурної генераццїї рівнів. Була проаналізована предметна область і поставлена задача даної роботи. В існуючих методах генерації рівнів було виявлено недолік і в результаті розробки модифікованого алгоритму він був виправлений. Розроблений модифікований метод базується на існуючих алгоритмах, але дозволяє виконувати задачу генерації рівня більш ефективно завдяки своїй оптимізації використання ресурсів. Даний метод не є універсальним рішенням через можливі складності в реалізації для більш складних рівнів, але добре себе показує в ситуаціях, коли частини рівня мають відносно однакову геометрію.

ПЕРЕЛІК ПОСИЛАНЬ

1. App2Top. Этапы разработки игры для мобильных платформ. URL: https://app2top.ru/columns/e-tapy-razrabotki-igry-dlya-mobil-ny-h-p-40118.html (дата звернення 29.10.2020).

2. Сахнов К. Семь этапов создания игры: от концепта до релиза. URL: https://habrahabr.ru/company/miip/blog/308286/ (дата звернення 29.10.2020).

3. Habr. Краткая история игровой индустрии в разбивке по платформам. URL: https://habr.com/ru/company/miip/blog/312884/ (дата звернення 29.10.2020).

4. Wikipedia. Жанри відеоігр. URL: https://uk.wikipedia.org/wiki/%D0%96%D0%B0%D0%BD%D1%80%D0%B8_%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D1%96%D0%B3%D0%BE%D1%80 (дата звернення 29.10.2020).

5. Habr. Процедурная генерация уровней. URL: https://habr.com/ru/post/418685/ (дата звернення 29.10.2020).

6. Порождающие шаблоны: Пул объектов (Object pool). URL: https://andrey.moveax.ru/post/patterns-oop-creational-object-pool (дата звернення 29.10.2020).

7. Обзор игрового движка Unity3d. URL: https://gamedevmania.ru/engines/unity3d (дата звернення 29.10.2020).

8. Will Goldstone. «Unity Game Development Essentials» – October 2009 – 316 p.

9. Кристиан Нейгел. «C# 5.0 и платформа .NET 4.5 для профессионалов – Professional C# 5.0 and .NET 4.5.» – М.: «Диалектика», 2013. – 1440 с. – ISBN 978-5-8459-1850-5.

10. Хокинг Дж. «Unity в действии. Мультиплатформенная разработка на C#» – СПб.: Питер, 2016. – 336 с.: ил. – (Серия «Для профессионалов»).

11. Басараб М.А., Домрачева А.Б., Купляков В.М. Алгоритмы решения задачи быстрого поиска пути на географических картах. Инженерный журнал: наука и инновации, 2013, вып. № 11. URL: http://engjournal.ru/catalog/it/hidden/1054.html.

12. Habr. Генерация и решение лабиринта с помощью метода поиска в глубину по графу. URL: https://habr.com/ru/post/262345/ (дата звернення: 29.10.2020).

13. Studfiles. Поиск в ширину. URL: https://studfile.net/preview/1399243/ (дата звернення: 20.03.2020).

14. Князев Б.А., Черкасский В.С. Начала обработки экспериментальных данных. Измерительный практикум: методическое пособие. Новосибирск: Изд-во Новосиб. ун-та, 2011.

15. Imangulova, Z. An algorithm for building a project team considering interpersonal relations of employees" / Imangulova, Z., Kolesnyk, L. // Eastern-European Journal of Enterprise Technologies. – 2016. – Vol. 6. – P. 19–25.