МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему Розробка програми для тренування логічного мислення. Процедурна генерація оточення
Program development for logical thinking training. Enviroment procedural generation

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

Гасанов Нурал Тофіг огли
(прізвище, ім'я, по-батькові)

Керівник канд фіз.-мат. наук Антоненко О.С.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент ст. викладач Максимов О. С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2021 р.

Завідувач кафедри

_____ _Є.В. Малахов_
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол № __ від «___» ____ 2021 р.

Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _Н.Ф. Казакова_
(підпис) (прізвище, ініціали)

Одеса-2021

АНОТАЦІЯ

Дипломна робота є складовою частиною проекту тренувальника логічного мислення. Основним призначенням тренувальника логічного мислення є систематичне тренування головного мозку для підтримування мозку в активному стані. Тренувальник дуже корисний для тих, діяльність яких пов’язана з постійною розумовою діяльністю.

Метою роботи є проектування і розробка підсистеми процедурної генерації оточення, з адаптивним інтерфейсом користувача та комфортною механікою пересувань персонажу.

Актуальність використовування підсистеми процедурної генерації в тому, що користувач має безліч можливостей для дій в інтерактивному додатку.

Для реалізації проекту обраний кроссплатформений движок Unity, підтримка системи контролю версій Git, та сервіс 3D моделей та анімацій Mixamo.

Спроектована та реалізована підсистема процедурної генерації, також реалізовані адаптивний інтерфейс користувача та комфортна механіка пересувань персонажу. Було виконано тестування тренувальника логічного мислення.

ABSTRACT

The graduation work is an integral part of the logical thinking trainer project. The main purpose of the logical thinking trainer is to systematically train the brain to keep the brain active. A trainer is very useful for those whose activities are associated with constant mental activity.

The aim of the work is to design and develop a procedural environment generation subsystem, with an adaptive interface and convenient mechanics of character movement.

The relevance of using the procedural generation subsystem is that the user has many opportunities for actions in an interactive application.

For the implementation of the project, the cross-platform Unity engine, support for Git version control system, and the Mixamo 3D models and animations service were chosen.

Designed and implemented subsystem of procedural generation, adaptive interface and convenient mechanics of character movement are also implemented. The logical thinking trainer was tested.

АННОТАЦИЯ

Дипломная работа является составной частью проекта тренажера логического мышления. Основным назначением тренажера логического мышления является систематическая тренировка мозга для поддержания мозга в активном состоянии. Тренажер очень полезен для тех, деятельность которых связана с постоянной умственной деятельностью.

Целью работы является проектирование и разработка подсистемы процедурной генерации окружения, с адаптивным интерфейсом и удобной механикой передвижений персонажа.

Актуальность использования подсистемы процедурной генерации в том, что пользователь имеет множество возможностей для действий в интерактивном приложении.

Для реализации проекта выбран кроссплатформенный движок Unity, поддержка системы контроля версий Git, и сервис 3D моделей и анимаций Mixamo.

Спроектирована и реализована подсистема процедурной генерации, также реализованы адаптивный интерфейс и удобная механика передвижений персонажа. Также выполнено тестирование тренажера логического мышления.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення

ТЛМ – Тренувальник логічного мислення

КР – Кваліфікаційна робота

ПГ – Процедурна генерація

WFC – WaveFunctionCollapse

BSP – Binary space partitioning

ВСТУП

Робота присвячена побудові програмної реалізації підсистеми процедурної генерації тренувальника логічного мислення за допомогою Unity на мові C#.

Тренувальник логічного мислення (ТЛМ) – це деякий тренажер, який сприяє підвищенню активності головного мозку. В результаті тренування скорочується час, необхідний мозку для вирішення різноманітних задач, та дозволяє підвищити цілісність сприяття навколишнього середовища.

Тренування сприяє вибудовуванню моделі окремих процесів та явищь. Володіння такого роду якостями мислення як критичність, доказовість, абстрактність та лаконізм потрібні людині у будь-яких сферах діяльності.

Систематичне використання ТЛМ дозволяє більш тривалий час не втомлюючись підтримувати мозок у активному стані.

Користування ТЛМ має велике значення для людини на всіх етапах його життя. Тому це буде корисно для школярів, студентів та працевлаштованих людей, особливо для людей, діяльність яких пов’язана з постійною розумовою діяльністю.

В процесі використання ТЛМ користувач буде вирішувати різноманітні завдання, що розташовані у спеціальних кімнатах. Кімнати з завданням будуть розташовані в деякому оточені, та користувачу потрібно їх знайти, потім їх вирішити.

Оточення представляє собою велику тривимірну локацію складної структури, яка складається із 3D блоків. Вручну таку модель створювати досить довго, це вимагає рутинної роботи, тому було прийнято рішення використовувати метод процедурної генерації для створення оточення.

Тому ТЛМ буде складатися з двох підсистем:

1) підсистеми процедурної генерації оточення;
2) підсистеми кімнат з логічними завданнями.

Актуальність використання процедурної генерації в інтерактивних проектах в тому, що воно дає користувачеві відчуття "нового досвіду", відповідно, з'являється безліч можливостей діяти в інтерактивному додатку.

Потрібність процедурної генерації саме в ТЛМ в тому, що користувач може кожен раз проходити задачі тренувальника логічного мислення по різному, тобто користувач кожен раз досліджує оточення заново. Також процес знаходження кімнати з логічним завданням тренує просторове мислення.

Метою даної дипломної роботи є проектування та розробка підсистеми процедурної генерації оточення програмного забезпечення для тренування логічного мислення.

Для досягнення наведеної мети необхідно розв’язати наступні задачі:

1) виконати аналіз предметної області, пов’язаної із процедурною генерацією оточення;
2) спроектувати підсистему, пов’язану із процедурною генерацією оточення;
3) розробити програмну реалізацію процедурної генерації оточення;
4) виконати тестування отриманої підсистеми процедурної генерації оточення ТЛМ.

1 ОГЛЯД ОБЛАСТІ ДОСЛІДЖЕННЯ

Кваліфікаційна робота має базуватися на існуючих методах та алгоритмах процедурної генерації оточення, тому потрібно провести їх аналіз та обрати метод для реалізації.

Як і здебільшого процедурна генерація може знадобитися для створення оточення в двомірній/тривимірній графіці.

Відповідно, область застосування впирається в індустрію розробки комп'ютерних ігор, чи, як приклад: додатків симуляції створення цілих світів (створення планет з сузір'ями, галактик, тощо).

Для розвитку якостей логічного мислення людини можна розробити процедурно-генеруєме оточення та кімнати з логічними та математичними задачами. У рамках цієї КР дослідимо побудову процедурної генерації оточення.

1.1 Етап формування поняття про процедурне оточення

По-перше, розглянемо, що саме з себе представляє “Процедурна генерація”. Процедурна генерація (ПГ) – це автоматичне створення просторового контенту за допомогою алгоритмів. Тобто, процедурна генерація представляє собою програмне забезпечення, яке може створювати просторовий контент самостійно, чи спільно при взаємодії з гравцями чи геймдізайнерами. [1]

ПГ включає в себе множину генеративних алгоритмів, принцип роботи яких полягає в створенні даних не вручну, а алгоритмічно: замість ручного створення того, що ми хочемо створити (карти, музику, рельєф, тощо), пишеться алгоритм, який успішно може створити різноманітні приклади без багаторазового виконання того самого процесу. Особливо корисний такий підхід в інтерактивних ПЗ, де випадковим чином може генеруватися ціла карта чи рівень (наприклад, карти з відеоігор: Minecraft, Terraria чи Factorio).

Прикладом генерації процедурного оточення у відеогрі Terraria, де застосовується генерація шуму для генерації 2D-рельєфу. Для цього застосовується функція Gradient, в яку передається відрізок прямої в N-вимірному просторі (тобто в будь-якому просторі координат, будь-то 2D, 3D, тощо). Вона обчислює поле градієнту уздовж цього відрізку. Вхідні координати проектуються на цей відрізок і їх значення градієнту обчислюється в залежності від того, де вони лежать щодо кінцевих точок відрізка. Спроектованим точкам призначаються значення в інтервалі (-1, 1). Тобто можна поставити функцію Gradient уздовж осі Y. У верхній частині інтервалу зіставити поле градієнту з -1 (повітря), а в нижній – з 1 (земля). [2]

На рисунку 1.1 чітко видно, як виглядає ландшафт для 2D оточення з застосуванням шуму.

Рисунок 1.1 – Застосування шуму для генерації 2D оточення

За цим же принципом вибудовуються тривимірні блоки в відеогрі Minecraft, приклад наведений на рисунку 1.2.

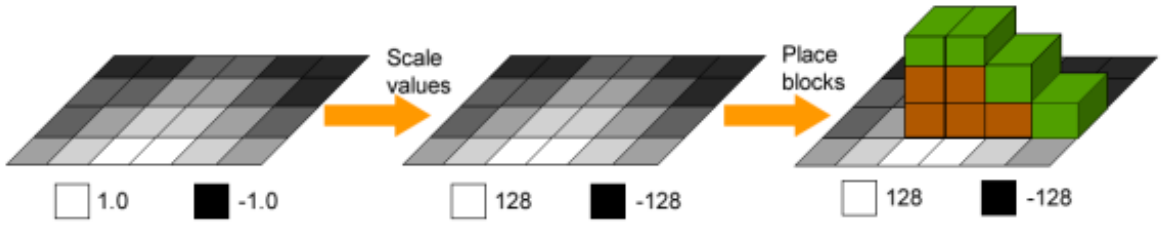

Рисунок 1.2 – Принцип розташування тривимірних блоків в Minecraft

1.2 Методи ПГ оточення

Існують декілька актуальних методів процедурної генерації оточення: метод двійкового розбиття простору, метод “хода п’яниці”, метод клітинних автоматів та алгоритм колапсу хвильової функції WFC.

1.3 Метод двійкового розбиття простору BSP

Класичний метод двійкового розбиття простору BSP, являє собою метод рекурсивного розбиття євклидового простору на опуклі множини та гіперплощини. В результаті об’єкти отримають уявлення в виді структури даних, так званим BSP-деревом.

Беремо область, так звану листом (Leaf), ділимо її, вертикально чи горизонтально, на два менших листа, потім повторяємо процес з меншими областями, знову і знову, доки кожна область не стане менше чи рівній заданому максимальному значенню. Після завершення процесу у нас виходить ієрархія розбитих Leaf, з якими можна робити все, що завгодно. В тривимірній графіці BSP можна використовувати для сортування видимих для гравця об’єктів чи для розпізнавання колізій в ще менших частинах. [3]

Приклад схеми того, як виконується алгоритм двійкового розбиття простору на рисунку 1.3.

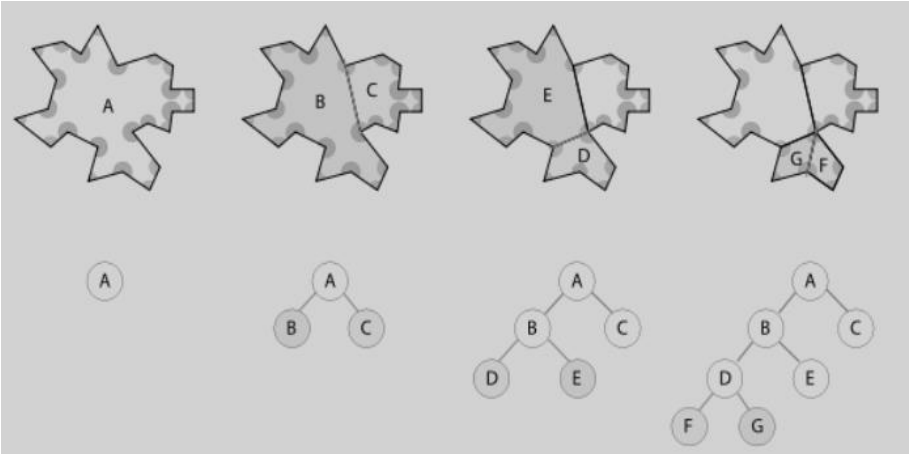

Рисунок 1.3 – Схема виконання алгоритму двійкового розбиття простору

Також, приклад того, який результат ми отримаємо при методі двійкового розбиття простору на рисунку 1.4.

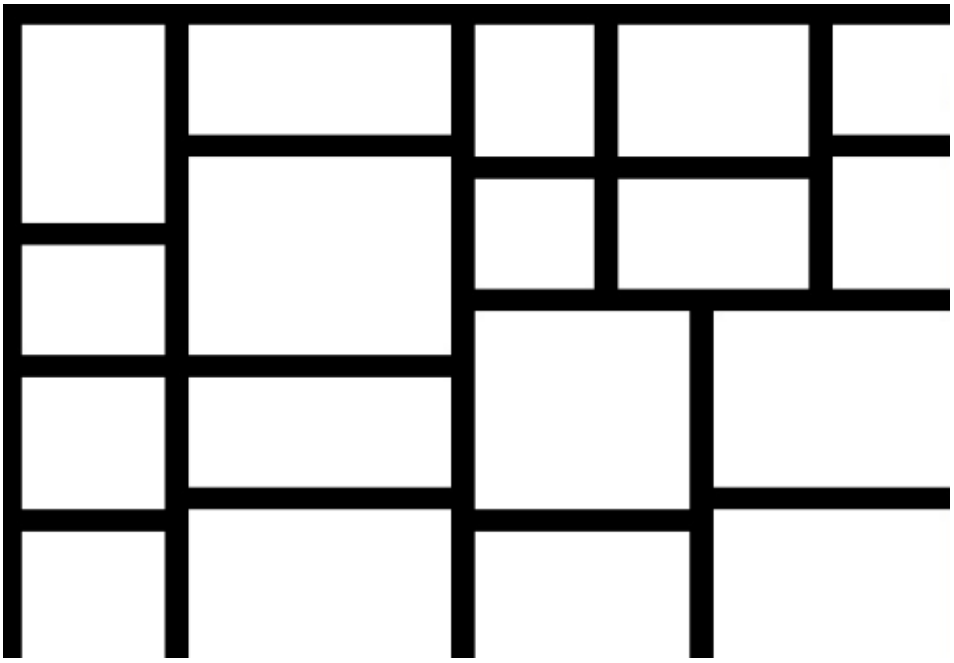

Рисунок 1.4 – Приклад двійкового розбиття простору

1.4 Метод “ходу п’яниці”

Метод “ходом п’яниці” – отримав свою назву за відповідний хаотичний візерунок, який утворюється в результаті його роботи.

Метод основується на переміщенні курсору, який зафарбовує область кімнати, рухаючись в випадковому напрямку. Він описується таким чином:

1) кожна точка рівню оголошується “стіною” – непрохідною областю;
2) на площі рівня обирається випадкова точка – точка початку. Вона відзначається як “підлога” – прохідна область;
3) обирається випадковий напрямок руху;
4) курсор робить крок в обраному напрямку, його нова точка розташування відзначається як “підлога”;
5) кроки 3-4 повторюються до тих пір, поки не будуть зафарбована задана кількість точок.

Цей метод гарантує відсутність ізольованих пустих областей - з кімнати можна дістатися до будь-якої точки на карті. [4]

Прикладом демонстрації результату роботи цього алгоритму продемонстровано на рисунку 1.5.

Рисунок 1.5 – Результат роботи алгоритму “Хід п’яниці”

1.5 Метод клітинних автоматів

Клітинний автомат – це набір клітин, утворюючих деяку періодичну решітку з заданими правилами переходу, визначаючими стан клітини в наступний момент часу через стан клітин, що знаходяться від неї на відстані не більше деякого, в теперішній момент часу. [5]

Алгоритм, яких базується на клітинним автоматі, починає свою роботу з того, що випадковим образом об’являє кожну точку рівня “стіною” чи “підлогою”. Таким чином отримається стартовий стан клітинного автомату.

Сусідами кожної точки являються 8 сміжних с нею точок. Після цього починається його ітераціоний процес, який визначається наступними правилами:

1) точка залишається стіною, якщо вона була стіною та інші сусідні точки це стіни;
2) точка стає стіною, якщо вона була підлогою, а інших 5 її сусідів це стіни.

Прикладом демонстрації результату роботи цього алгоритму продемонстровано на рисунку 1.6.

Рисунок 1.6 – Результат роботи клітинного автомату

1.6 Коллапс хвильової функції WFC

В рамках даної КР вибраний алгоритм WaveFunctionCollapse, WFC, який був запропонований Максимом Гуміним для процедурної генерації оточення ТЛМ.

В основі цього алгоритму лежить ціла ідея покрокового створення готового зображення з відстеженням того, які тайли (готове зображення, для використовування в інтерактивному додатку у 2D графіці) ідентичні вже

частково створеному зображенню. Також можна проглянути приклади процедурно згенерованих з seed зображень по методу WaveFunctionCollapse на рисунку 1.7. Ліворуч в нас йде seed зображення, з якого нам потрібно отримати декілька тайлів, які знаходяться праворуч.

Окрім створення між собою схожих зображень ще одною областю застосування алгоритму WFC являється генерація цілих тайлових 2D мап та 3D оточень. [6, 7]

Приклади генерації тайлових 2D мап на рисунку 1.8.

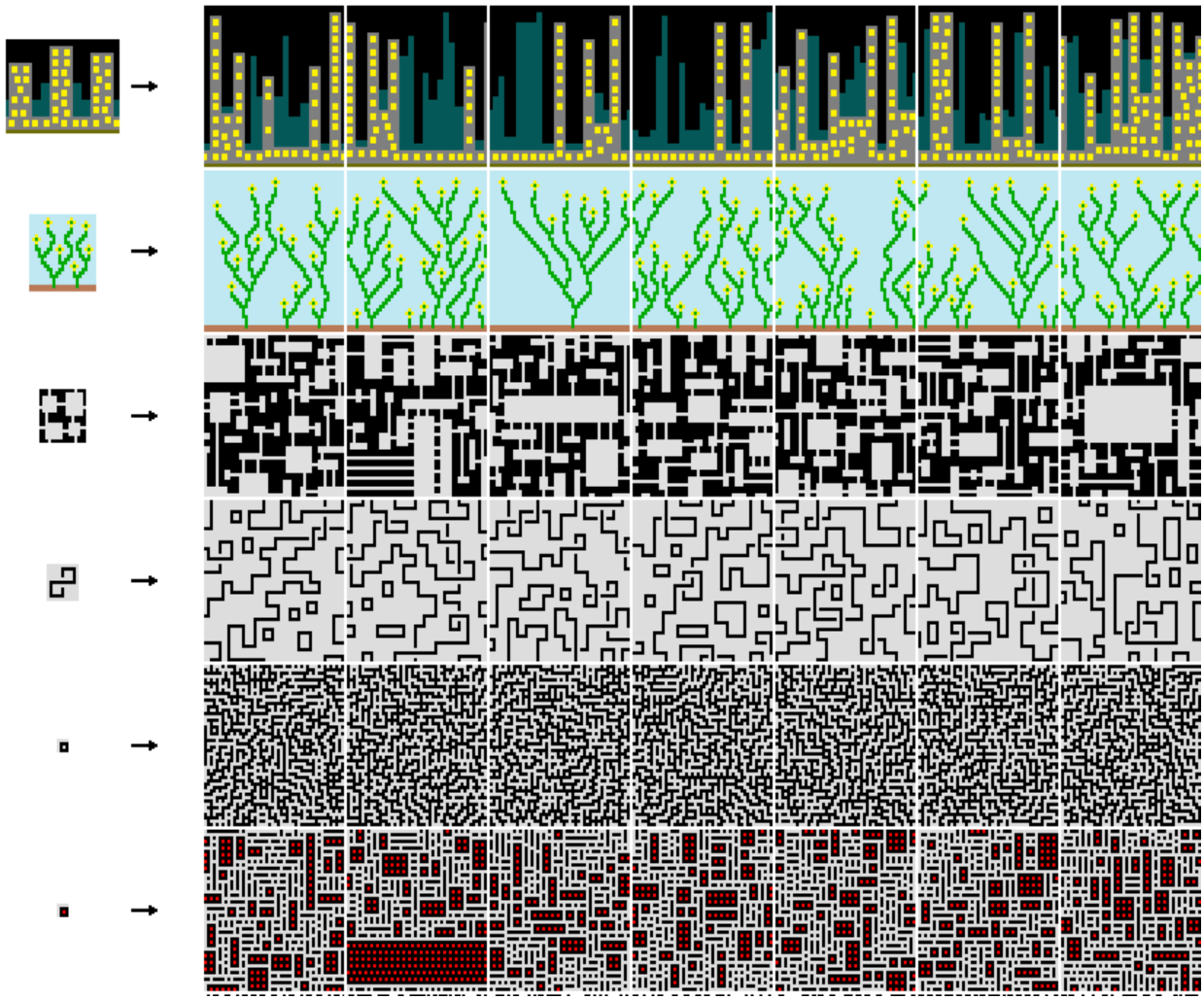

Рисунок 1.7 – Приклади процедурно згенерованих з seed зображень

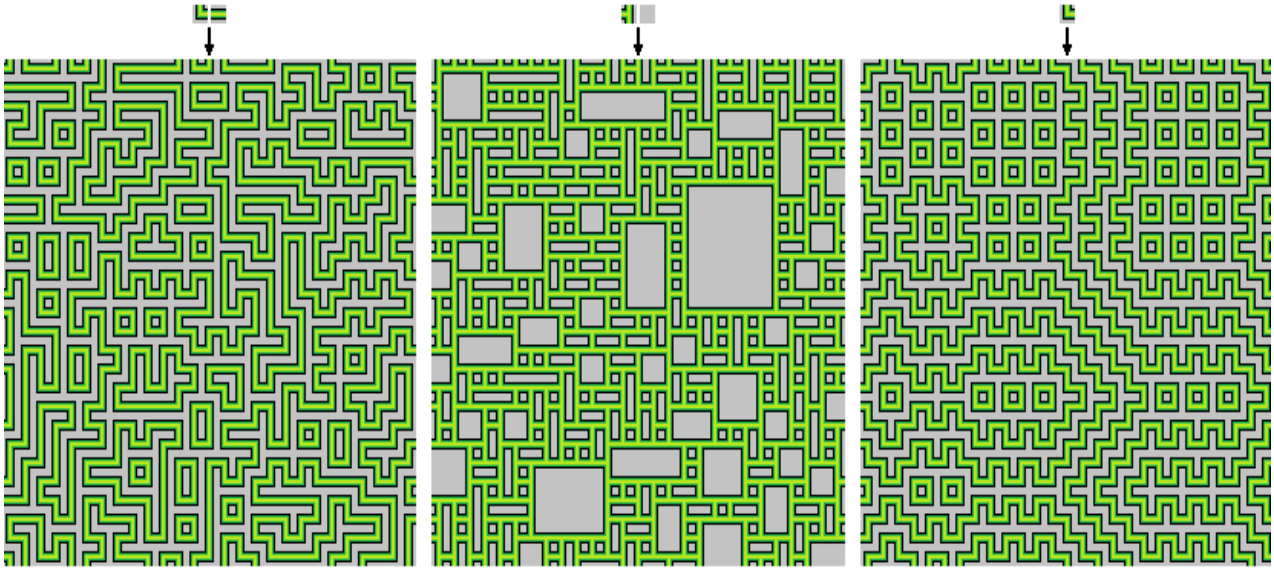

Рисунок 1.8 – Приклади генерації тайлових 2D мап завдяки WFC

1.7 Вибір методу процедурної генерації оточення

Для задач КР проекту ТЛМ обраний метод WFC, тому що використовуючи цей метод процедурної генерації можна згенерувати локації складної структури у 3D графіці, які складаються із тривимірних блоків, параметри яких можна налаштувати необхідним чином для генерації кімнат з логічними завданнями. Також існують адаптації цього методу для використання в різних движках.

2 ПРОЕКТУВАННЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

Для відображення тривимірної графіки треба використовувати деякий движок чи бібліотеку. Один із популярних виборів для вирішення цієї задачі це Unity – кроссплатформне середовище розробки комп'ютерних ігор, розроблена американською компанією Unity Technologies.

Основними перевагами Unity є наявність візуальної середовища розробки, кроссплатформної підтримки і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами і труднощі при підключенні зовнішніх бібліотек. [8]

Можливості та переваги Unity:

1) простий Drag&Drop інтерфейс;
2) легке налаштування плагінів;
3) налагодження додатка у редакторі;
4) підтримка мови програмування C#;
5) розрахунки фізики виконуються за допомогою PhysX від NVIDIA;
6) підтримка нових версій DirectX (на цей час підтримується DX12);
7) проект ділиться на окремі сцени. Кожна сцена має свій набір об'єктів;
8) безкоштовний;
9) систематичні оновлення;
10) велика кількість асетів для розробників.

Також на цьому движку є можливість розробки Augmented Reality (AR) та Virtual Reality (VR), що є одною з величезних переваг даного движку, та має величезне співтовариство на офіційному форумі, де кожен може отримати допомогу при потребі.

Наступним кроком розглянемо архітектуру проекту та етапи проектування ТЛМ.

2.1 Етапи проектування процедурної генерації оточення

Наступним кроком буде поділ на етапи проектування частини процедурної генерації проекту ТЛМ.

Поділ складається з таких етапів:

1) опис User Story із Use-Case діаграмою;
2) опис архітектури частини процедурної генерації оточення ТЛМ із Sequence діаграмою.

2.2 Опис User Story та UML діаграма процедурного оточення

В даному проекті після запуску додатку ТЛМ користувач опиняється у головному меню, де він може одразу запустити тренувальник, чи вийти (за його бажанням). Після запуску тренувальника може пройти деякий час, доки не згенерується ПГ оточення, після цього користувач може користуватися тренувальником, досліджувати ПГ оточення, та також за його бажанням, він може вийти з самого додатку.

Якщо розділити все це покроково, то це виглядає так:

1) крок 1 – Користувач відкриває додаток та відкривається головне меню;
2) крок 2:
   a) користувач натискає кнопку “START” та переходить до кроку 3;
   b) користувач натискає кнопку “EXIT” та виходить із додатку;
3) крок 3 – Починається процес генерації ПГ оточення;
4) крок 4 – Користувач може спокійно досліджувати ПГ оточення;
5) крок 5 – Користувач, по бажанню, може закрити додаток.

Також усі ці кроки візуально відображені в якості Use Case діаграми на рисунку 2.1.

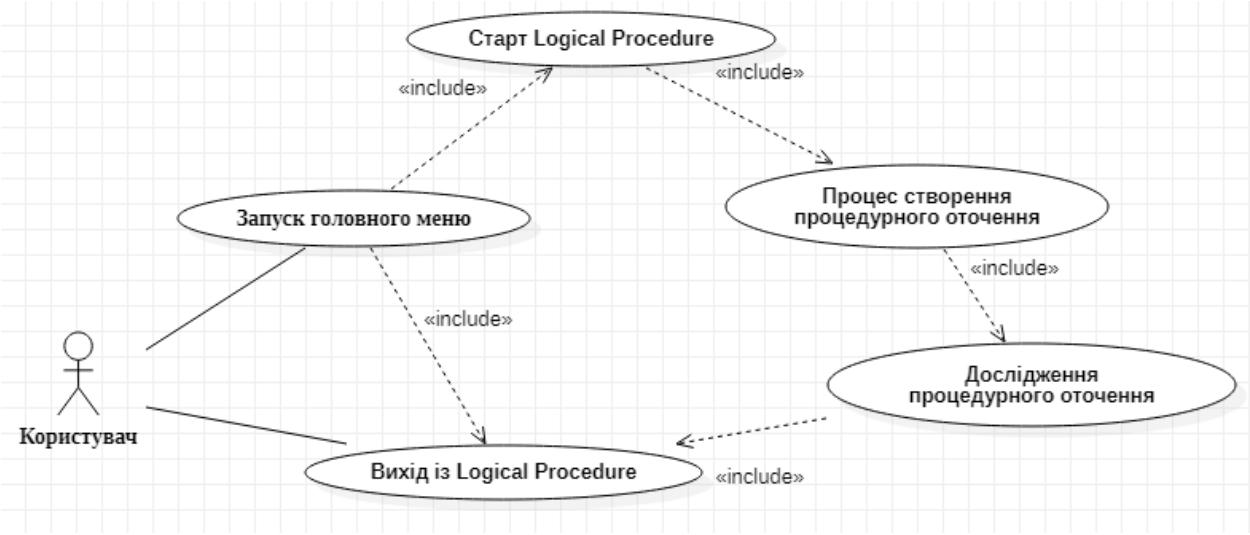

Рисунок 2.1 – Use Case діаграма проекту

2.3 Опис роботи алгоритму WFC

Для початку, потрібно детально описати саму роботу алгоритму WFC, щоб мати наочний приклад, як він працює, на прикладі двомірної графіки.

Сам алгоритм:

1) вважати вхідне бітове зображення і підрахувати кількість патернів N x N;
2) створити масив з розмірами вихідних даних (в джерелах званий «wave»). Кожен елемент цього масиву позначає стан області розміром N x N в вихідних даних. Стан галузі N x N є суперпозицією патернів N x N входять даних з булевими коефіцієнтами (тобто стан пікселя в вихідних даних є суперпозицією входять квітів з речовими коефіцієнтами). Значення false означає, що відповідний патерн заборонений, значення true означає, що відповідний патерн поки не заборонений;
3) формувати хвилю в повністю неспостерігаємому стані, тобто де всі булеві значення мають значення true;
4) повторювати наступні кроки:
   а) спостереження:

1) знайти елемент хвилі з мінімальною ненульовою ентропією. Якщо таких елементів немає (якщо у всіх елементів ентропія нульова або невизначена), то завершити цикл (4) і перейти до кроку (5);
2) колапсувати цей елемент в стан визначеності відповідно до його коефіцієнтами і розподілом патернів N x N входять даних;

b) поширення: поширити інформацію, отриману на попередньому кроці спостереження;

5) до даного моменту всі елементи хвилі або мають повністю бачимо стан (всі коефіцієнти, крім одного, дорівнюють нулю) або знаходяться в стані протиріччя (всі коефіцієнти дорівнюють нулю). У першому випадку повернути вихідні дані. У другому випадку завершити роботу, нічого не повертаючи.

В рамках частини процедурної генерації буде реалізована тривимірна графіка, а значить, буде використано слово "слот" для позначення в тривимірній сітці вокселей, яка може містити блок (або бути порожнім), і буде використано слово "модуль" для блоку, який може займати такий слот.

Алгоритм вибирає, які модулі вибрати для кожного слоту в світі. Масив щілин вважається хвильової функцією в неспостерігаємому стані. Це означає, що в кожному слоті є набір можливих модулів, які можна туди поставити. Мовою квантової механіки можна сказати: “Слот знаходиться в суперпозиції всіх модулів”. Світ починається в абсолютно не спостерігаються стані, коли кожен модуль можливий в будь-якому слоті. Один за іншим кожен слот згортається. Це означає, що випадковим чином вибирається один модуль з набору можливих модулів. Потім слідує етап поширення обмежень. Для кожного модуля дозволяється розміщувати поруч тільки підмножина модулів. Кожен раз, коли слот згортається, необхідно оновити набори модулів, які все ще можна розмістити в сусідніх слотах. Етап поширення обмеження є найбільш витратною в обчислювальному відношенні частиною алгоритму.

Важливий аспект алгоритму - вирішити, який слот згорнути. Алгоритм завжди згортає слот з найменшою ентропією. Це слот, у якого найменше

вибору (або хаосу). Якщо все модулі мають однакову ймовірність, то слот з найменшою кількістю можливих модулів має найменшу ентропію. Як правило, ймовірність вибору модулів різна. Слот з двома можливими модулями з однаковою ймовірністю має більший вибір (більшу ентропію), ніж слот з двома модулями, де один дуже ймовірний, а інший дуже малоймовірний.

Ймовірності модулів і правила суміжності все виставляються вручну (приклад на рисунку 2.2).

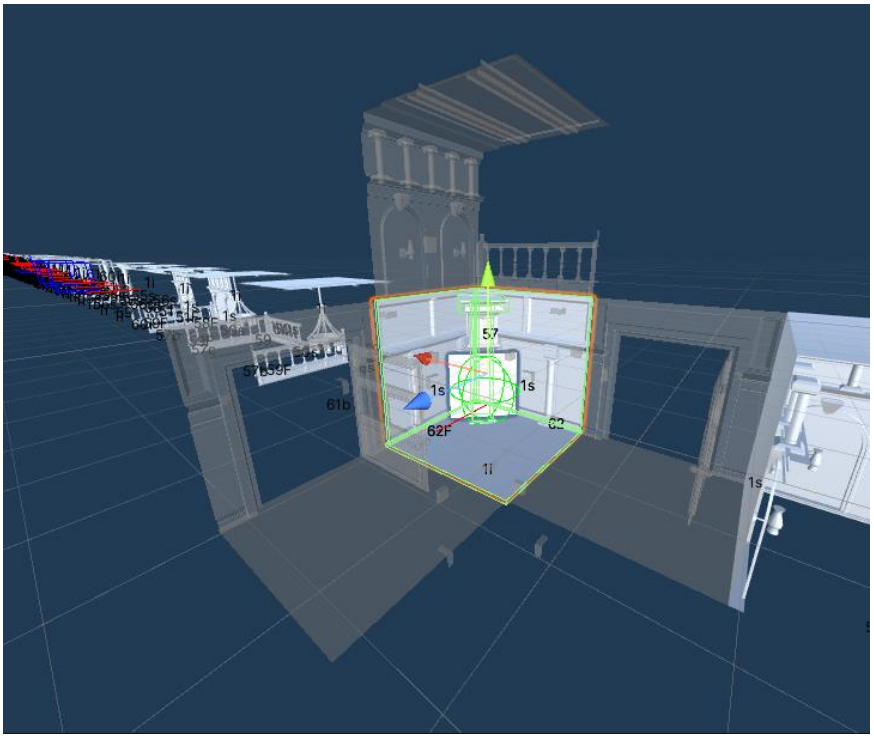

Рисунок 2.2 – Приклад блоку із модулями, який виставлений вручну

2.4 Розбір блоків, прототипів і модулів

Світ створюється з набору з ~ 100 блоків, які частково взяті з Unity Asset Store (сервіс Unity, де можна отримати безкоштовні/небезкоштовні асети різного роду, які потрібні по нужді розробника) і доопрацьовані за допомогою Blender. Самі блоки поки що без текстур (приклад на рисунку 2.3).

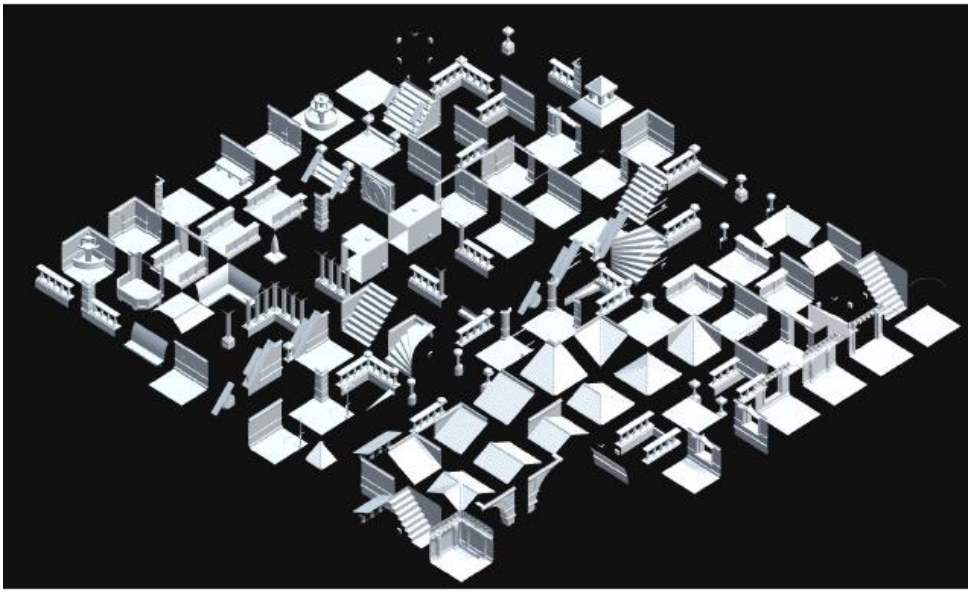

Рисунок 2.3 – Блоки, з яких створюється оточення

Алгоритм повинен знати, які модулі можна розмістити поруч один з одним. Кожен модуль має 6 списків можливих сусідів, по одному для кожного напрямку, по можливості треба уникнути створення цього списку вручну. Також треба мати спосіб автоматично генерувати повернені варіанти блоків.

І того, й іншого можна досягти за допомогою того, що називається прототипами модулів. Це MonoBehaviour, який можна зручно редагувати в редакторі Unity. З них автоматично створюються модулі разом зі списками дозволених сусідів і повернениими варіантами.

Тут виникає складність: як моделювати інформацію про суміжності, щоб цей автоматичний процес працював? Ось, власне кажучи, і саме рішення:

Кожен блок має 6 з'єднувачів, по одному на кожну грань. На роз'ємі є номер. Крім того, горизонтальні роз'єми або перевернуті, або не перевернуті, або симетричні. Вертикальні з'єднувачі або мають індекс повороту від 0 до 3 (b, c, d на знімку екрана), або позначені як інваріантні щодо обертання (рисунок 2.4).

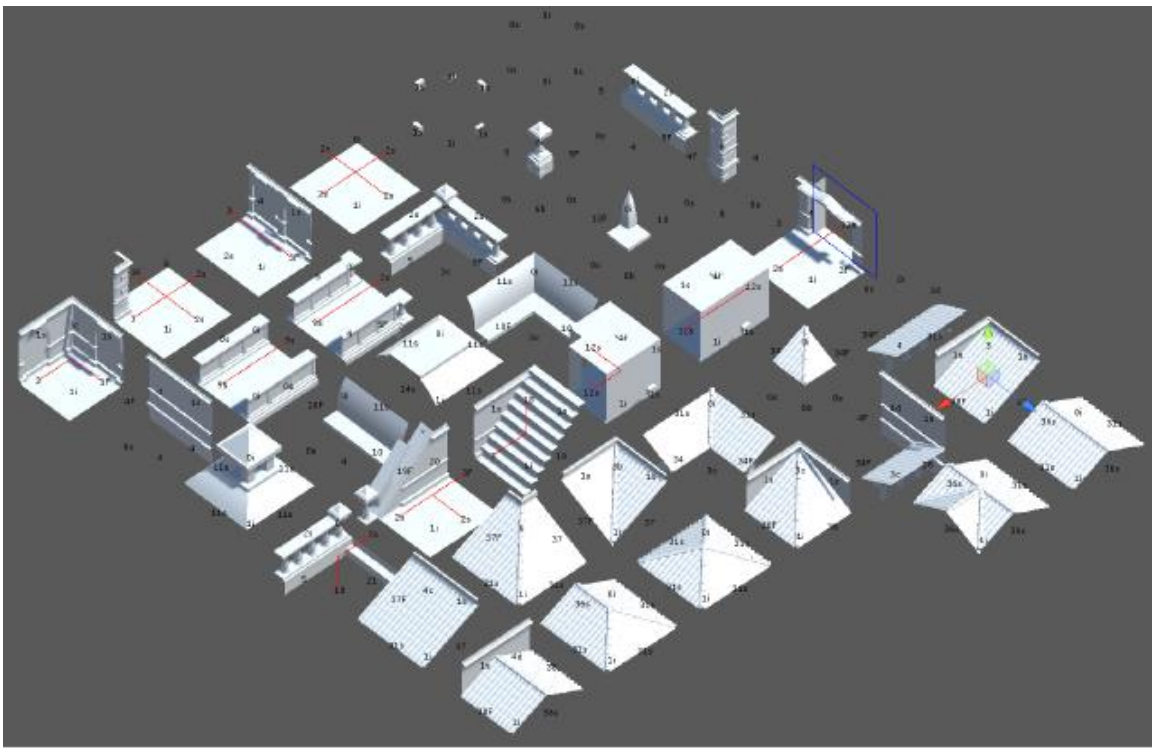

Рисунок 2.4 – Приклад блоків із їх з’єднувачами

Виходячи з цього, можна автоматично перевіряти, які модулі дозволені поруч один з одним. Суміжні модулі повинні мати однаковий номер роз'єму. І їх симетрія повинна збігатися (однаковий індекс обертання по вертикалі, пара перевернутих і не перевернутих по горизонталі) або вони повинні бути симетричними / інваріантними (приклад на рисунку 2.5).

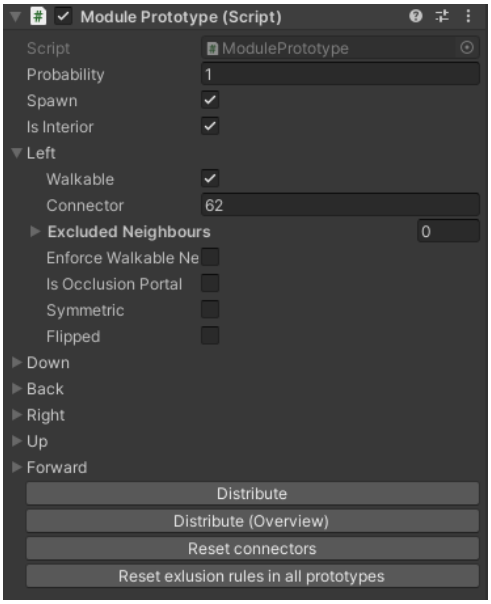

Рисунок 2.5 – Приклад налаштування модулю

Існують правила виключення, які дозволяють забороняти сусідам, які в іншому випадку були б дозволені. Деякі блоки з однаковими роз'ємами просто негарно виглядають поруч один з одним. Ось приклад карти, створеної без правил виключення (рисунок 2.6):

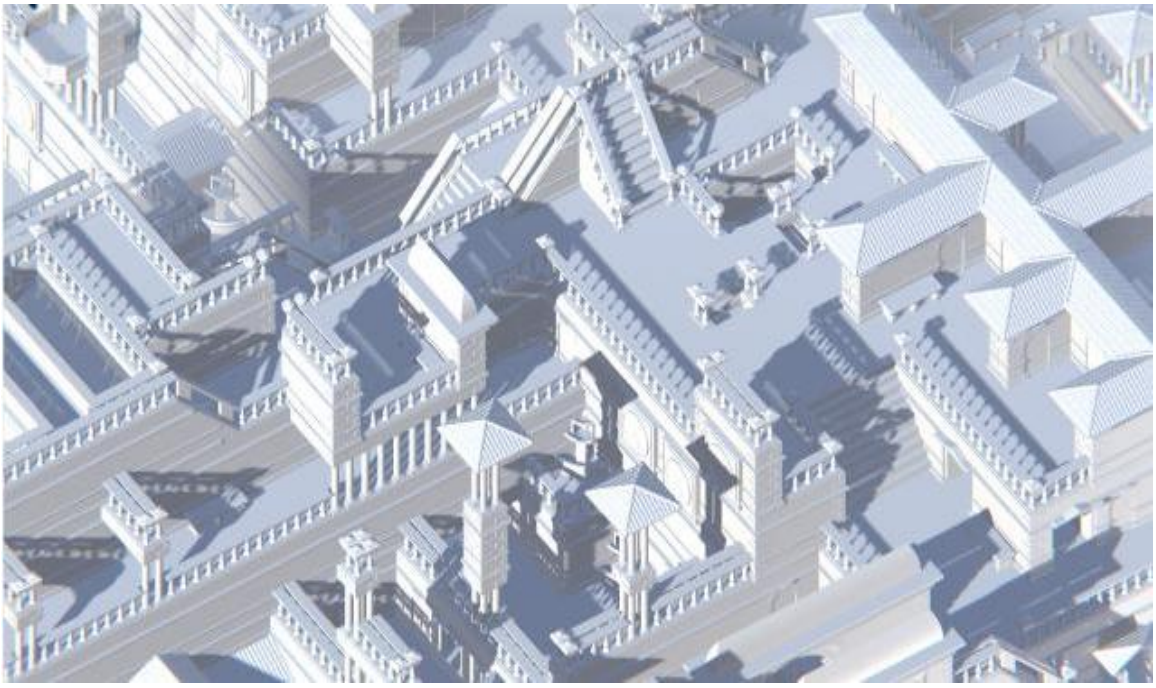

Рисунок 2.6 – Приклад карти процедурного оточення, вигляд зверху

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

Після детального розбору роботи процедурної генерації оточення і побудови правильної архітектури для її роботи можна приступити до розробки програмної частини підсистеми ТЛМ.

Умовно поділимо на етапи розробки підсистеми:

1) реалізація адаптивного інтерфейсу користувача ТЛМ;
2) реалізація процедурної генерації оточення;
3) реалізація комфортної системи пересування готовим 3D персонажем із анімаціями, та налаштування анімацій через Unity Mecanim Animation System.

Для реалізації проекту розроблена система класів, які написані на мові C#.

3.1 Реалізація інтерфейсу користувача

Насамперед, для зручності тестування програми краще відразу реалізувати адаптивний інтерфейс, який підійде до різних ПК з будь-якою роздільною здатністю екрану.

Для цього створена окрема сцена проекту, та доданий компонент сцени (GameObject) – полотно (Canvas), яке являє собою абстрактний простір, в якій відбувається налаштування і відрисовка UI. Все UI-елементи повинні бути нащадками ігрових об'єктів, до яких приєднаний Canvas.

До Canvas додано 4 кнопки:

1) Start Button – кнопка, що відповідає за запуск ТЛМ;
2) Exit Button – кнопка, що закриває додаток;
3) Settings Button – кнопка, що відкриває панель налаштувань ТЛМ;
4) Record Button – кнопка, що відкриває панель рекордів користувача.

На рисунку 3.1, 3.2 та 3.3 можна розглянути, який має вигляд головне меню користувача ТЛМ.

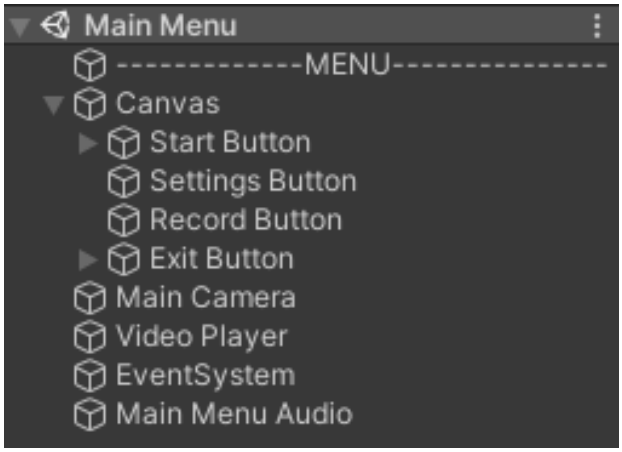

Рисунок 3.1 – Приклад ієрархії сцени Main Menu

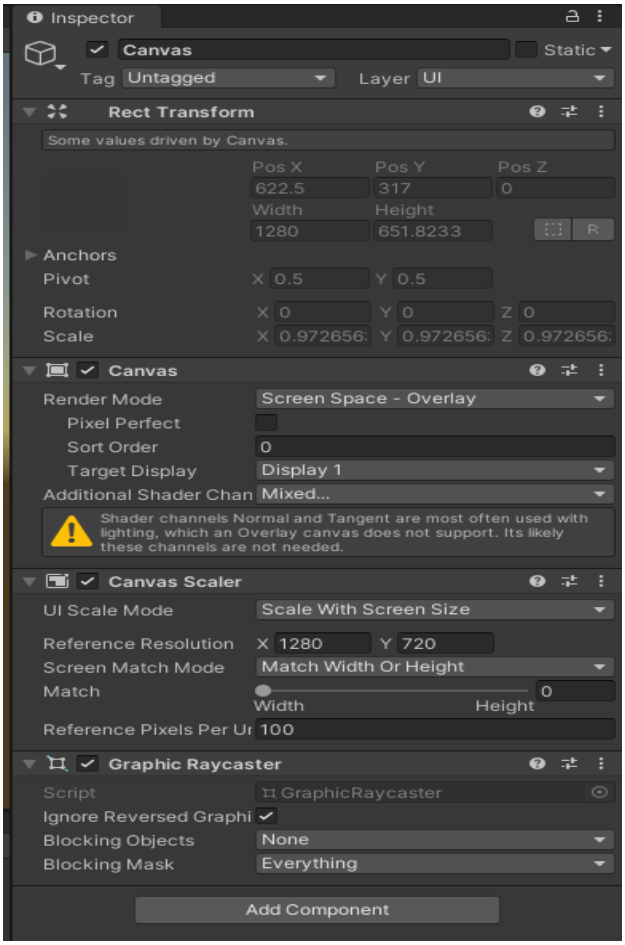

Рисунок 3.2 – Приклад налаштування Canvas для адаптації під будь-який екран ПК

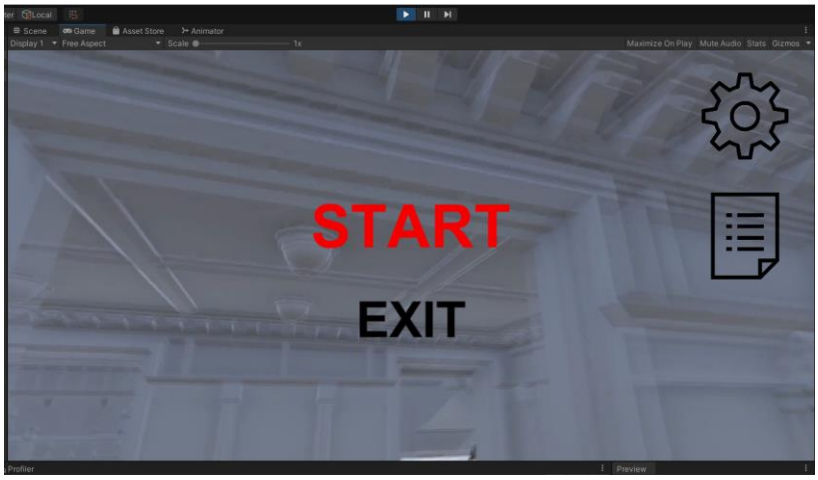

Рисунок 3.3 – Приклад фінального вигляду головного меню ТЛМ

Кнопка Start Button, як і інші, адаптивно-налаштована під різні екрани, та доданий скрипт, який відповідає за запуск симулятора ТЛМ (рисунок 3.4). При наведенні на кнопку курсором, кнопка збільшується в півтора рази і стає червоного кольору, щоб користувач розумів, що кнопка по-справжньому робоча. Після натискання кнопки, відбувається завантаження рівня ТЛМ.

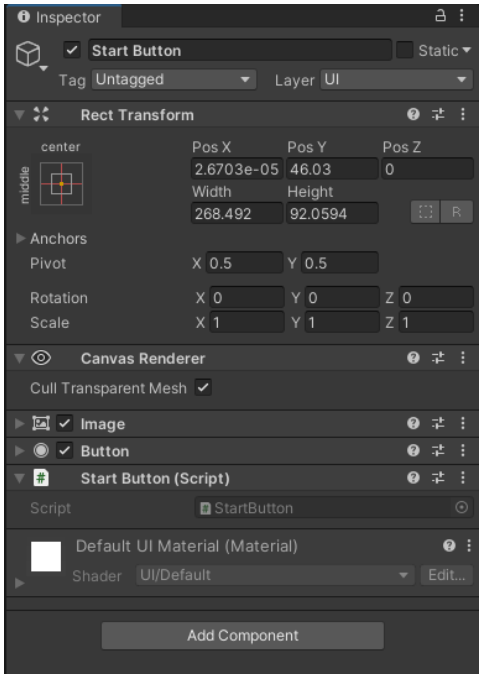

Рисунок 3.4 – Приклад того, з чого складається Start Button кнопка

3.2 Етап додання 3D персонажу з базовими логікою та анімаціями пересування

Для комфортного тестування ТЛМ доданий в сцену об'єкт Player, в сам об'єкт додані такі компоненти: Main Camera і скрипти, що відповідають за пересування персонажа по процедурно-згенерованому оточенню.

У движку Unity існує 2 способи реалізації переміщення фізики: через компонент CharacterController або Rigidbody компонент.

У чому ж відмінність цих двох компонентів можна переглянути в таблиці 3.1.

Таблиця 3.1 - Порівняльна таблиця двох компонентів переміщення об'єктів по згенерованому оточенню

	CharacterController	Rigidbody
Фізика пересування:	Нема, треба реалізувати частину гравітації самостійно	Присутня
Вбудована колізія:	Присутня	Не присутня
Готові методи для реалізації пересування:	Присутні	Не присутні
Складність програмної реалізації:	Середня складність	Складна в реалізації, особливо для новеньких споживачів движку Unity
Універсальність реалізації пересувань персонажу:	Присутня, але не така як у Rigidbody	Чудова універсальність
Економія часу реалізації	Присутня	Не присутня

Тобто, через те що CharacterController трохи більше простий в реалізації і економить достатньо часу програмної реалізації, виберемо його, тому що для завдань реалізації простого пересування персонажа зійде і CharacterController.

Також доданий 3D персонаж з базовими анімаціями пересування, щоб користувач максимально відчував залученість в процес проходження логічних задач (рисунок 3.1). Модель персонажу взята з безкоштовного сервісу 3D моделей персонажів та анімацій Mixamo від знаменитої компанії Adobe.

Рисунок 3.1 – Вигляд 3D персонажу ТЛМ

До написання логіки пересувань персонажу було створено декілька скриптів, детально можна переглянути на рисунку 3.2.

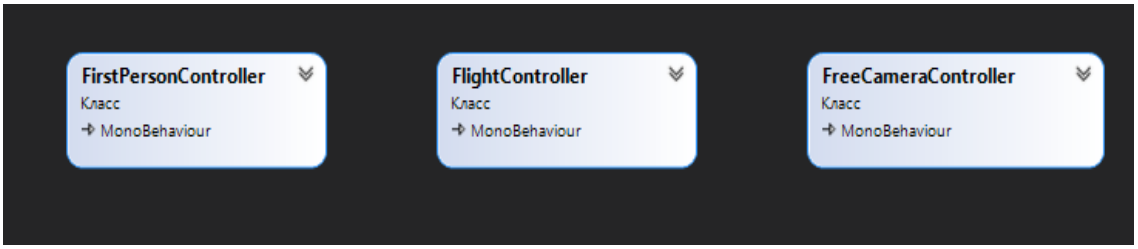

Рисунок 3.2 – Класи для функціонування механіки пересувань персонажу

В класі FirstPersonController реалізований функціонал пересування персонажа, гравітація, логіка визначення поверхні, на якій стоїть персонаж, щоб, в разі стрибка, він не повис в повітрі, також в цьому скрипту реалізована поведінка анімацій пересувань.

В класі FlightController реалізований функціонал польоту персонажа, щоб була можливість у користувача повністю подивитися результат процедурної генерації оточення.

В класі FreeCameraController реалізований функціонал контролю камери персонажа, також залочений градус вертикального повороту камери з 90 градусів до -90 градусів. За межі цього камера, по реалізованої логіці, не виходить.

Щодо анімацій пересувань персонажу використаний також сервіс Mixamo, та налаштований через вбудовану систему комплексного контролю великої кількості анімацій Mecanim в одного персонажа, приклад на рисунку 3.3.

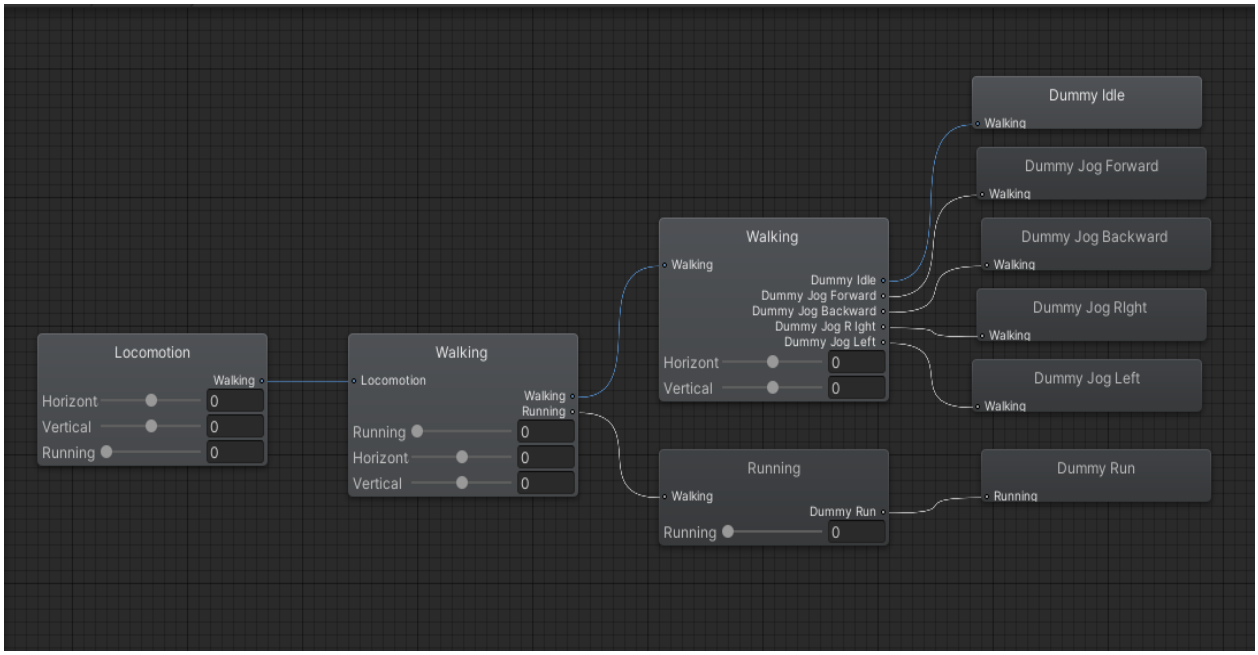

Рисунок 3.3 – Схема роботи пересувань персонажу користувача

3.3 Етап розробки процедурної генерації оточення

За призначенням використовуються в проекті класи, що реалізують механіку процедурної генерації проекту ТЛМ, можна згрупувати наступним чином (рисунок 3.4):

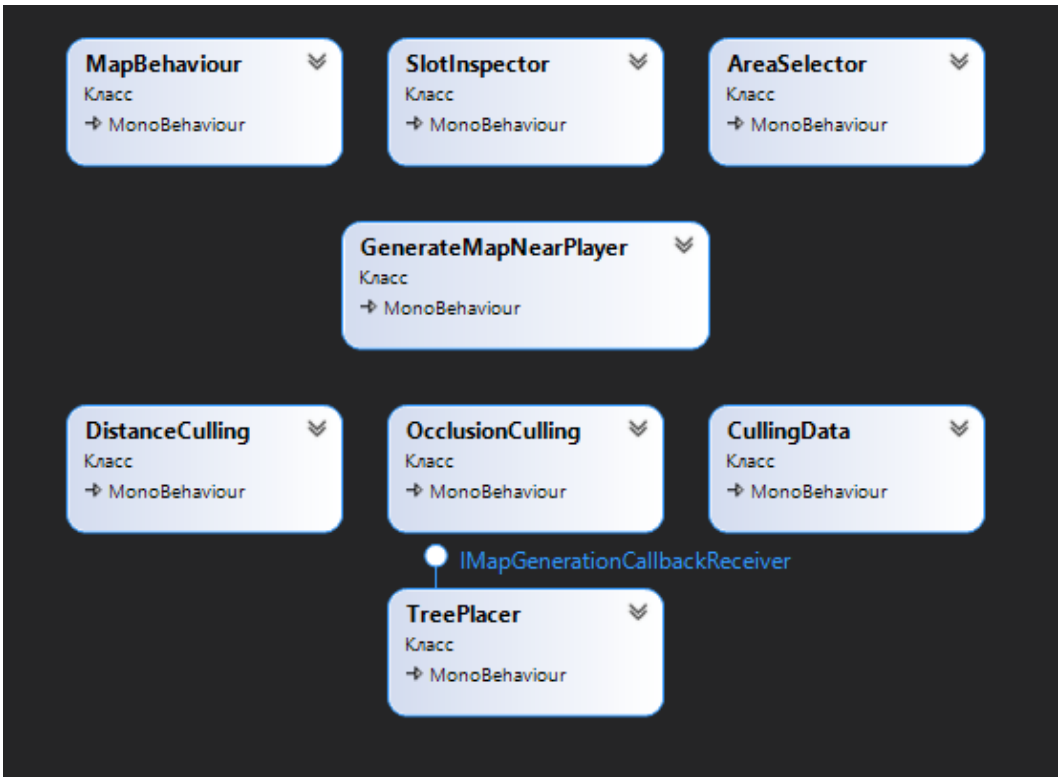

Рисунок 3.4 – Діаграма класів частини процедурної генерації проекту ТЛМ

Клас MapBehaviour відповідає за частину налаштування генерації оточення, як приклад, там можна визначити: якої максимальної висоти може згенеруватися оточення, та інше.

Клас SlotInspector має один статичний метод DrawGizmo(), який відображає Gizmos, поточно натиснутого слоту, приклад на рисунку 2.2, та показує детальну інформацію про одну позицію. Він показує, які модулі можуть бути створені в цій позиції, і дозволяє вибирати модулі вручну приклад на рисунку 3.5.

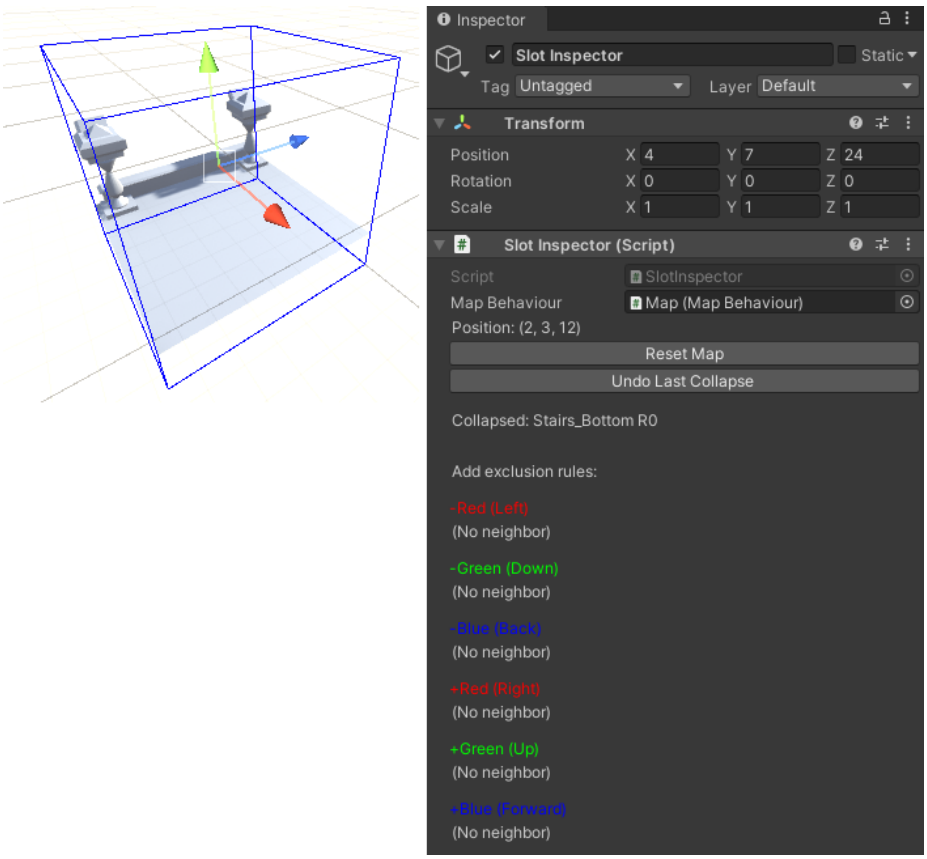

Рисунок 3.5 – Відображення модуля та вибір вручну

Клас AreaSelector генерує тестове оточення на сцені для огляду результату роботи методу WFC, приклад на рисунку 3.6.

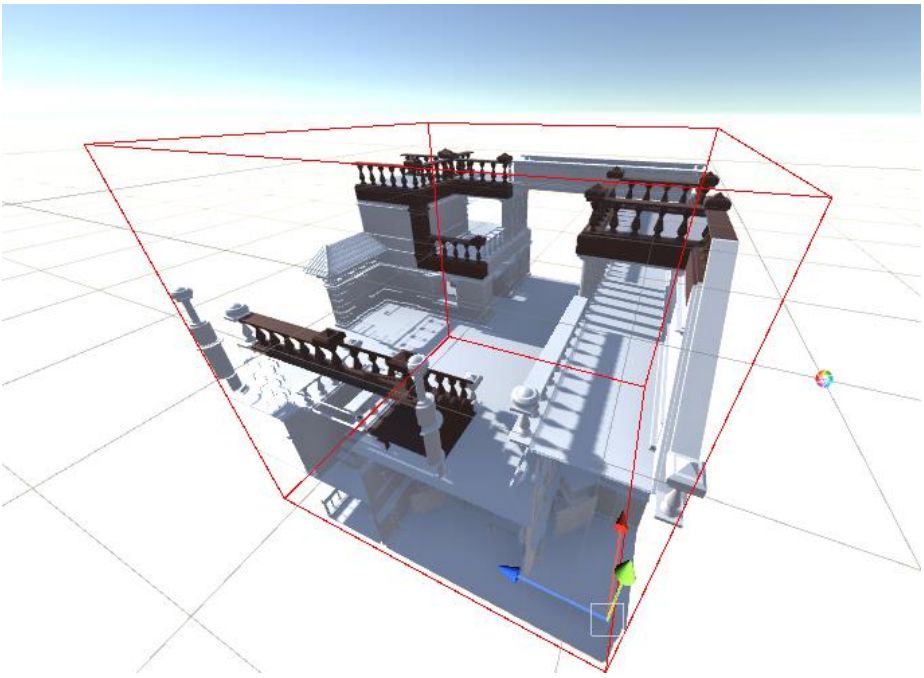

Рисунок 3.6 – Приклад згенерованого оточення завдяки методу WFC

Клас GenerateMapNearPlayer є головним в плані процедурної генерації, також прописана логіка генерації оточення з використовуванням методу WFC навколо персонажа в радіусі 36 юнітів (метрів).

Клас TreePlayer с підключеним інтерфейсом IMapGenerationCallbackReceiver знаходить вільну щілину для росту дерева, та перевіряє чи повністю створена околиця біля дерева, якщо навколишні блоки згенеровані, то починається зріст дерева.

Класи DistanceCulling, OcclusionCulling, CullingData призначені для правильної роботи процедурної генерації оточення.

3.4 Контроль версій проекту ТЛМ

Для контролю версій проекту ТЛМ використаємо Git, заздалегідь створений репозиторій у GitHub за цим посиланням: https://github.com/nural-khalafov/LogicalProcedure. Сам репозиторій є приватним, для того, щоб максимально приховати вихідний код проекту. За бажанням розробника, можна буде зробити репозиторій публічним.

Контроль версій Git призначений для того, щоб по можливості, в разі виникнення критичної помилки проекту повернутися до опублікованих колишніх змін проекту. Також важливо мати доступ до інтернету в цей момент.

3.5 Демонстрація роботи проекту

Після запуску тренувальника логічного мислення відкривається головне меню користувача, де користувач може одразу запустити сам тренувальник, чи закрити його, приклад на рисунку 3.7.

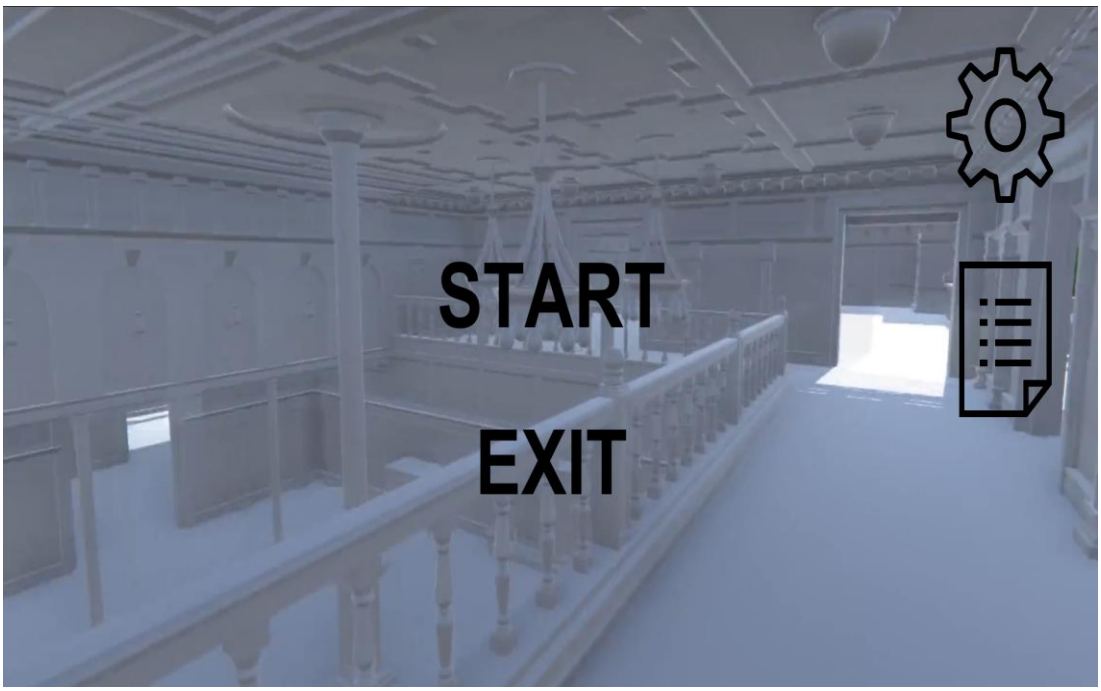

Рисунок 3.7 – Головне меню користувача

Після запуску самого тренувальника логічного мислення, генерується оточення та користувач може керувати персонажем, досліджувати процедурно-генеруєме оточення чи вирішувати логічні завдання, приклад на рисунку 3.8.

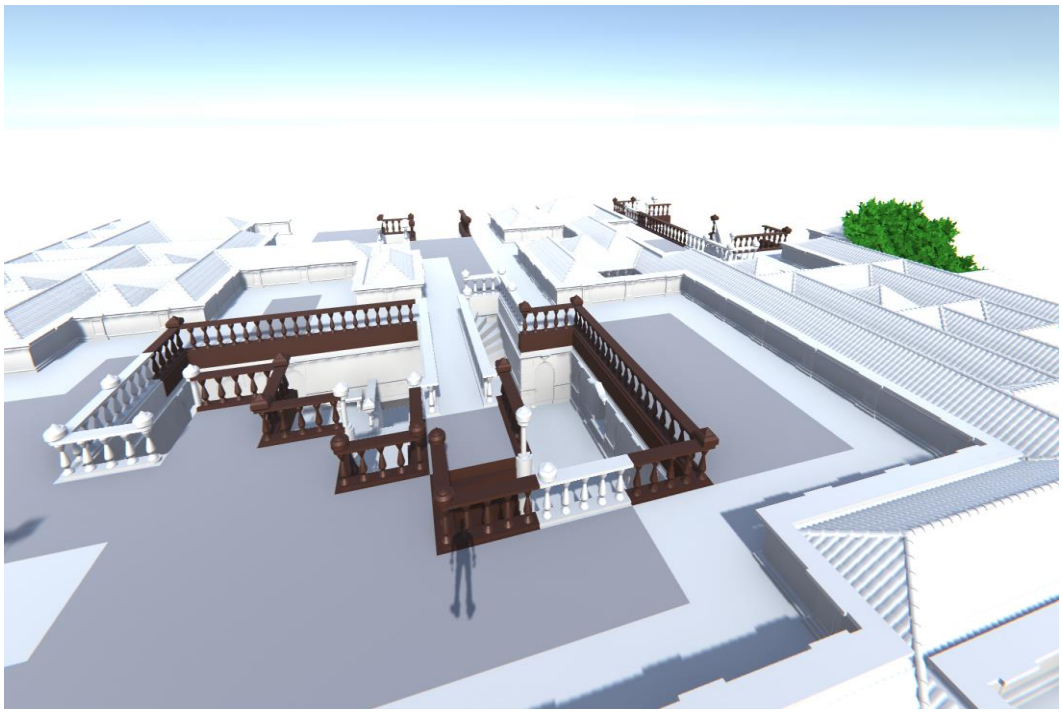

Рисунок 3.8 – Процедурно-генеруєме оточення ТЛМ

Користувач керує персонажем за допомогою клавіш “WASD”, а також керувати камерою персонажа за допомогою миши.

Користувач може:

1) пересуватися;
2) стрибати за допомогою клавіши Space;
3) літати за допомогою клавіши LCtrl;
4) прискорюватися за допомогою клавіши LShift.

Пересування працює переміщення працює навіть в тому випадку, коли персонаж знаходиться в повітрі також як і прискорення персонажа.

Стрибок працює тільки тоді, коли персонаж знаходиться на землі. Механіка польоту працює в будь-яких ситуаціях, коли персонаж на землі або навіть в повітрі.

Також прискорення можна комбінувати з польотом і переміщенням персонажа.

ВИСНОВОК

В ході виконання кваліфікаційної роботи досліджена предметна область процедурної генерації оточення, розібрані різні методи процедурної генерації, в тому числі метод WFC, показані можливості движку Unity. Спроектована загальна архітектура програми та робота процедурної генерації в тривимірній графіці за допомогою методу WFC.

В якості програмної реалізації розроблені система класів для коректної процедурної генерації оточення, керування анімованим 3D персонажем, адаптивний інтерфейс головного меню користувача. Використани движок Unity, система контролю версій Git, сервіс 3D моделей та анімацій Mixamo.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Как процедурная генерация помогает создавать открытые миры [Електронний ресурс] – Режим доступу: https://dtf.ru/gamedev/169117-kak-procedurnaya-generaciya-pomogaet-sozdavat-otkrytye-miry

2. Создаём свою Minecraft: генерация 3D-уровней из кубов [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/472574/

3. How to use BSP trees to generate maps [Електронний ресурс] – Режим доступу: https://gamedevelopment.tutsplus.com/tutorials/how-to-use-bsp-trees-to-generate-game-maps—gamedev-12268

4. Drunken master cave generation [Електронний ресурс] – Режим доступу: https://forums.roguetemple.com//index.php?topic=4128.0

5. Generate random cave levels using cellular automata [Електронний ресурс] – Режим доступу: https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata—gamedev-9664

6. Generating Worlds With Wave Function Collapse [Електронний ресурс] – Режим доступу: https://www.procjam.com/tutorials/wfc/

7. WaveFunctionCollapse Github repository [Електронний ресурс] – Режим доступу: https://github.com/mxgmn/WaveFunctionCollapse

8. Unity User Manual [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/